Empirical Performance of Simple Statistical Inference-Type Models with Feature Extraction in Comparison to Modern Machine Learning Methods Concerning Regression and Classification Problems



Martin P. Dittgen

Christ Church

University of Oxford

A thesis submitted for the degree of

MSc in Mathematical Finance (part-time)

14 April 2019

Acknowledgements

This thesis is not a product of a one-man show, but the result of work in a helpful and encouraging academic environment.

First, I want to thank Dr Ferdinand Graf and Professor Samuel N. Cohen, for supervising this thesis and giving useful feedback during its inception.

Second, I want to thank the Mathematical Institute for providing the computational resources to perform the extensive calculations required for this work.

Third, I want to give acknowledgements to my employer *d-fine* who made it possible for me to attend this part-time programme and supported me throughout the course.

Fourth, I want to thank the many who contributed their part to the open source libraries, in particular the python modules *scikit-learn, numpy, scipy* and *pandas*, which we used extensively in our data processing and calculations.

Last but not least I want to thank the class of 2017 which made the time in Oxford as pleasant and inspiring as it was.

Abstract

While most statistical methods used in the industry are of simple structure – like decision trees and logistic regression – many machine learning competitions are won by modern complex methods, which also outperformed the more simple methods in previous benchmark studies. In these benchmark studies, the results were usually obtained over raw and unprocessed datasets. In practice, when developing a model, a feature extraction is performed first to create non-redundant and useful covariates for the model. This is especially important for the simple methods, which often require approximately linear features without severe outliers.

In this thesis, we investigate the performance of modern machine learning methods on raw data, without any additionally extracted features, in comparison to simpler and more interpretable models on data together with feature extraction. For this purpose we employ certain preprocessing and feature extraction algorithms to objectively approximate this task over 32 different datasets.

Our results show that, for classification problems, the simple methods perform almost as well as the more complex methods. This is true for both binary classification, regarding the metric *area under the ROC curve*, and for multiclass classification with regards to the *weighted F1 Score* metric.

In our regression problems, the modern methods outperform the simpler methods and seem superior to the more interpretable methods in terms of their respective *R-squared* or *explained variance*.

We suspect that the difference between classification and regression problems might be due to the evaluation metrics used; specifically how they react to the monotonicity or linearity of the extracted features, but more research is necessary to investigate this difference.

Contents

List of Figures **vi**

Terms and Abbreviations **viii**

1 Introduction **1**

1.1 Motivation 1

1.2 Discussion of Literature 3

2 Basics **6**

2.1 The Goal of Machine Learning 6

2.2 The Bias–Variance Tradeoff 7

2.3 Cross-Validation 9

2.4 Metrics for Algorithm Evaluation 9

2.4.1 Evaluation of regression algorithms 10

2.4.2 Evaluation of classification algorithms 10

3 Models and Feature Extractors **13**

3.1 Overview 13

3.2 White Box Models 14

3.2.1 Regression 14

3.2.2 Classification 17

3.2.3 Regression and Classification 18

3.2.4 Decision trees 18

3.3 Extractors 19

3.3.1 Compact representation and manifolds methods 19

3.3.2 Extraction via decomposition, projection, and clustering . . 21

3.4 Black Box Models 23

3.4.1 Tree-based methods 24

3.5 Special Topics 26

3.5.1 Predicting multiple classes via softmax 26

3.5.2 The kernel trick 27

4 Datasets, Pipeline and Hyperparameters **29**
4.1 Datasets . . . 29
4.2 Processing Pipeline . . . 31
4.3 Hyperparameter Optimization . . . 33
4.3.1 A note about nested cross validation . . . 34
4.3.2 Setup of the random search . . . 35

5 Comparison of Model Performance **37**
5.1 Methodology of the Comparison . . . 37
5.2 Comparison of Relative Ranking . . . 38
5.3 Comparison of Normalized Scores . . . 42
5.3.1 Normalization of the results . . . 42
5.3.2 Discussion of normalized results . . . 44

6 Conclusion **51**

Appendices

A Supplementary Results **56**
A.1 Medals for Relative Ranking . . . 57
A.2 Normalized Results . . . 61

B Possible Extensions of this Thesis **75**
B.1 Enhance the Extractors . . . 75
B.2 Increase the Scope of the Analysis . . . 75
B.3 Investigate the Monotony Hypothesis . . . 76
B.4 Focus Studies . . . 76

C Details on Datasets and Longlist **77**
C.1 Notes on Specific Datasets . . . 77
C.2 Longlist . . . 79
C.3 Distributions for Hyperparameter Optimization . . . 84

References **86**

List of Figures

2.1 Bias-Variance tradeoff 8

3.1 Feature extractors applied to S-curve 22
3.2 Figure of a neural network 23

4.1 The data processing pipeline 33
4.2 Advantages of random-search hyperparameter optimization 34

5.1 Effect of normalization on random forest 44
5.2 Algorithm overview for regression and means of metric normalized explained variance 45
5.3 Algorithm overview for multiclass classification and mean of metric normalized weighted F1 score 46
5.4 Algorithm overview for binary classification and means of metric normalized Somers' D 47
5.5 Best algorithms for regression and means of metric normalized explained variance 48
5.6 Best algorithms for multiclass classification and mean of metric normalized weighted F1 score 48
5.7 Best algorithms for binary classification and means of metric normalized Somers' D 49

6.1 Sketch of monotonic feature vs linear feature 54

A.1 Algorithm overview for regression and means of metric normalized R-squared 69
A.2 Algorithm overview for multiclass classification and mean of metric normalized accuracy 70
A.3 Algorithm overview for binary classification and means of metric normalized accuracy 71
A.4 Algorithm overview for binary classification and means of metric normalized F1 score 72

A.5 Best algorithms for regression and means of metric normalized R-squared 73
A.6 Best algorithms for multiclass classification and mean of metric normalized accuracy 73
A.7 Best algorithms for binary classification and means of metric normalized accuracy 74
A.8 Best algorithms for binary classification and means of metric normalized F1 score 74

Terms and Abbreviations

Accuracy . . . . . Performance metric for classification tasks, see section 2.4.

AUROC . . . . . Area Under the ROC Curve, a performance metric for classification tasks, see section 2.4.

Black Box Model . Any machine learning model which defies a simple explanation of its inner workings, therefore making it essentially a black box. Compare to white box model.

-Cosine . . . . . Cosine similarity kernel, see 3.5.2.

CV . . . . . Cross Validation, a method to estimate the expected generalization error of a model, see 2.3.

Exp. Var. . . . . . Explained Variance, a performance metric for regression models, see 2.4.

Extractor . . . . . An algorithm applied before a white box model to perform feature extraction, which provides more powerful and useful features to regress, see section 3.3.

F1 Score . . . . . Performance metric for binary classification tasks. Can be used for multiclass classification problems if summed and weighted over the different classes, see section 2.4.

LDA . . . . . Linear Discriminant Analysis, a simple classification model, see section 3.2.

LLE . . . . . Locally Linear Embedding, one of the feature extraction methods, see 3.3.

Logit . . . . . Short for logistic regression model, see section 3.2.

Longlist . . . . . List of all variables piped into the models for a specific dataset. Note that some models might not utilize certain of this variables at all. The Longlists for each dataset can be found in appendix C.

MARS . . . . . Multivariate Adaptive Regression Splines, a regression model, see 3.2.

-k-Means . . . . . A feature extraction based on the k-means clustering algorithm, see 3.3.

PCA . . . . . Principal Component Analysis, one of the feature extraction methods. Also exists in a kernel variant, see 3.3.

-Poly3 . . . . . Third order polynomials, used as a kernel, see 3.5.2.

QDA . . . . . Quadratic Discriminant Analysis, a classification algorithm, see 3.2.

RBF . . . . . Radial Basis Function, used as a kernel, see 3.5.2.

R2 . . . . . R-squared or R^2, a performance metric for regression models, see section 2.4.

ROC-curve . . . . . Performance metric for binary classification tasks, see section 2.4. Also see AUROC.

-Sigmoid . . . . . The sigmoid function, used as a kernel, see 3.5.2.

Somers' D . . . . . Performance metric for binary classification tasks, see section 2.4.

SVC . . . . . Support Vector Classifier, a support vector machine for classification tasks, see chapter 3.

SVM . . . . . Support Vector Machine, a class of models which can be both used in regression and classification tasks, see chapter 3.

SVR . . . . . Support Vector Regression, a support vector machine adapted for regression tasks, see chapter 3.

White Box Model . A statistical or machine learning model which can be put into a form understandable by a human being such that an economic interpretation can be attached to the inner workings of the prediction process. Compare to black box model.

What I cannot create, I do not understand.

— Richard Feynman *from his blackboard at the time of his death*

1 Introduction

Contents

1.1 Motivation 1
1.2 Discussion of Literature 3

1.1 Motivation

On the Kaggle website[1] there are multiple competitions, starting on a regular basis, where machine-learning practitioners compete with their algorithms to achieve the best possible score on a non-public test set. Some competitions focus on unstructured data (like texts and images), others – the data we will focus on here – are based on more structured (spreadsheet like) data; such data that can be found in the databases of many corporations worldwide.

Generally, in most competitions ensemble methods based on trees tend to win, with *random forests* and the boosted tree algorithm *xgboost* usually ranking highest[2].

In the Kaggle survey *The State of ML and Data Science 2017*[3] people were asked what data science methods they use at work. The two most prevalent techniques were *logistic regression* with 63.5% and *decision trees* with 49.9% of

[1] https://www.kaggle.com/
[2] https://www.import.io/post/how-to-win-a-kaggle-competition/
[3] https://www.kaggle.com/surveys/2017

respondents stating that they use these two – both very interpretable – methods in their day to day jobs. The more modern, and harder to interpret, methods *random forests* and *neural networks* followed in 3rd and 4th place with 46.3% and 37.6%. *Boosting* algorithms, though winning in most competitions and benchmarks, ranked only at the 8th place with 23.9%.

Breiman (Leo Breiman et al. 2001) attacked the constrained use of only interpretable models. He argued:

> Accuracy generally requires more complex prediction methods. Simple and interpretable functions do not make the most accurate predictors.
>
> [...]
>
> The dilemma posed in the last section is that the models that best emulate nature in terms of predictive accuracy are also the most complex and inscrutable. But this dilemma can be resolved by realizing the wrong question is being asked. Nature forms the outputs y from the inputs x by means of a black box with complex and unknown interior. Current accurate prediction methods are also complex black boxes. So we are facing two black boxes, where ours seems only slightly less inscrutable than nature's.
>
> [...]
>
> My biostatistician friends tell me, "Doctors can interpret logistic regression." There is no way they can interpret a black box containing fifty trees hooked together. In a choice between accuracy and interpretability, they'll go for interpretability. Framing the question as the choice between accuracy and interpretability is an incorrect interpretation of what the goal of a statistical analysis is.
>
> The point of a model is to get useful information about the relation between the response and predictor variables. Interpretability is a way of getting information. But a model does not have to be simple to provide reliable information about the relation between predictor and response variables; neither does it have to be a data model. The goal is not interpretability, but accurate information.
>
> – (Leo Breiman et al. 2001)

Are most companies effectively losing money by employing simple and interpretable methods? How much worse do simple models perform in comparison to the more modern and complex ones? Looking at some previous benchmark studies (see our discussion in the next section), it seems that most of the simple methods perform significantly worse than the modern methods. However, most

of these benchmark studies focused on raw, unprocessed data only. Usually when constructing a model, one performs a feature extraction first, which creates new features with more relevant information, less redundancy or better linearity regarding the outcome; thus it facilitates the future learning of a model on such data and offers an economic interpretability.

The question of this thesis: Might it be possible that the difference between a manually crafted simple model, in comparison to a modern machine learning method applied to raw and unprocessed data, is not as large as one might naively assume?

In this thesis, we try to answer this question. The next section discusses the previous literature of benchmarking studies. In chapter 2, we discuss some statistical basics of machine learning and model evaluation. Chapter 3 discusses the most common statistical models. As we cannot objectively compare feature extraction done by hand, we have to find a way to (at least approximate) this process algorithmically. These algorithms, which we will call *extractors* in this work, are also discussed there (see section 3.3).

The datasets and their sequence of processing steps, called the *pipeline*, are discussed in chapter 4, together with the methodology of hyperparameter optimization. Chapter 5 describes the final aggregation and discussion of the results, leading to our conclusion in chapter 6.

1.2 Discussion of Literature

The first large scale study comparing learning algorithms was STALOG (King, Feng, and Sutherland 1995). The comparison focused on classification problems only and, due to the time of publication, did not include some of the modern tree-based and support vector machine (SVM) algorithms.

A study including these algorithms was published (Caruana and Niculescu-Mizil 2006), focusing again on classification. The 11 datasets were each downsampled to 4000 observations used for training due to the high computational demand. An additional 1000 observations were used to re-calibrate the algorithms (see our comment in section 4.3.2). The results were evaluated over 8 different performance

metrics. The algorithms giving the best average performance across all metrics and datasets were bagged trees, random forests, and neural networks.[4]

In a subsequent study (Caruana, Karampatziakis, and Yessenalina 2008) a similar evaluation was performed focusing on 11 high-dimensional binary classification datasets. They noted that the previous work (Caruana and Niculescu-Mizil 2006) suffered from a high correlation of the evaluated performance metrics. They decided to focus on three evaluation metrics: The threshold metric accuracy, the probability (calibration) metric squared error, and the ordering metric area under the ROC curve.

> To our surprise, the method that performs consistently well across all dimensions is random forests, followed by neural nets, boosted trees, and SVMs.
>
> – (Caruana, Karampatziakis, and Yessenalina 2008)

Lim et al. (Lim, Loh, and Shih 2000) published a comparison over 16 classification datasets. Their work focused on different kinds of tree algorithms and included a comparison of the training times of the algorithms.

A 2014 conference paper (Bischl, Schiffner, and Weihs 2014) compared the performance of 8 models on 36 classification problems. A principal focus of this study was on the evaluation of pre-processing steps like outlier removal, principal component analysis[5] and variable selection methods. Note that, contrary to our approach, they did not intend this to correspond to a manual human feature extraction but rather to benchmark the impact of these preprocessing steps in an automated pipeline.

> It appears that it is possible to considerably improve the performance by data pre-processing in some cases. However, for the majority of the investigated classification problems pre-processing did not result in improvements. We can also see that for different classifiers different pre-processing options are beneficial and that some classifiers profit much more from the pre-processing steps in this investigation than others. It was especially hard to improve upon the best performing basic method per data set.
>
> – (Bischl, Schiffner, and Weihs 2014)

[4] This ordering refers to the results without calibration.
[5] No kernels were used; only the standard linear variant of PCA was considered.

Recently (Y. Zhang et al. 2017) published a comparison over 7 algorithms on 12 classification datasets. They found that random forests performed best over all datasets, followed by the boosting algorithm *AdaBoost*. Note that in this work, possibly due to limited computational resources, no hyperparameter optimization was performed. Instead, they used "default" hyperparameter values for each algorithm, thus effectively biasing the comparison with regards to the quality of these "default" values on a given dataset.

To summarize the described literature: They all focused exclusively on classification problems and, with the exception of (Bischl, Schiffner, and Weihs 2014), did not use any feature extraction prior to fitting the models. They all agreed on the fact that, on average, modern non-interpretable machine learning methods performed better than simple methods.

This work aims to further contribute to the previous research by including regression datasets in the benchmark as well. Further, we will test how simple methods perform when they are allowed to use a wide array of feature extraction methods on their input.

If your experiment needs statistics, you ought to have done a better experiment.

— Ernest Rutherford

2 Basics

Contents

2.1 The Goal of Machine Learning 6
2.2 The Bias–Variance Tradeoff 7
2.3 Cross-Validation 9
2.4 Metrics for Algorithm Evaluation 9
2.4.1 Evaluation of regression algorithms 10
2.4.2 Evaluation of classification algorithms 10

In the following, we will discuss the basics of statistical modelling and machine learning.

2.1 The Goal of Machine Learning

In supervised machine learning we start with a training dataset containing *features*, also called *covariates* or *independent variables* together with a *target*, also called *dependent variable* or *label*. The whole matrix of features is usually denoted by $X = x_{i,j}$ where i corresponds to a specific observation and j to a specific feature of this observation. The target is equivalently given by y_i for observation i. The total number of observations will be denoted by n and the number of covariates by m. This dataset can now be used to construct a function or algorithm $\hat{f}$ and its parametrization $\hat{\theta}$, which can be used to assess new data points for which we do not know the target yet and estimate it to be:

$$\hat{y} = \hat{f}(x; \hat{\theta}) \tag{2.1}$$

The target can either be a numeric value (e.g. when regressing a house's value based on its features like number of rooms, square meters, location etc.) or a class membership where y might be the probability of belonging to a class or not (e.g. if a corporate debtor will default within the next year based on its current balance sheet). The former is called a *regression problem*, the latter a *classification problem* where we differentiate between a *binary classification problem* and a *multiclass classification problem* depending on how many categories are available (we will discuss the handling of multiclass problems later in section 3.5.1).

In order to assess how good a model fits we want to know how it would perform on new observations, which were not part of the training set. The error rate on unseen test data is usually called *generalization error* or *out-of-sample error*. In order to assess it we often retain a certain fraction of the data (for which we know the targets) as a test set and use the rest of the data for estimating $\hat{f}$ and its parametrization $\hat{\theta}$[1]. We evaluate the predictions of an algorithm against the actual labels for this set, subject to a loss function or metric[2]. For example for regression problems we might evaluate our algorithms in terms of the Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.2}$$

2.2 The Bias–Variance Tradeoff

In regression problems the mean squared error is often used as a loss-function

$$\text{MSE} = \text{E}\left[\left(y - \hat{f}(x)\right)^2\right] = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.3}$$

where y_i is the actual target and $\hat{f}(x_i) = \hat{y}_i$ the prediction for observation i. The MSE can be decomposed into three components:

$$\text{E}\left[\left(y - \hat{f}(x)\right)^2\right] = \left(\text{Bias}\left[\hat{f}(x)\right]\right)^2 + \text{Var}\left[\hat{f}(x)\right] + \sigma^2 \tag{2.4}$$

[1] This process if often called *training* or *fitting the model*.

[2] Many algorithms already use a loss function or similar criterion during training as a target for their optimization.

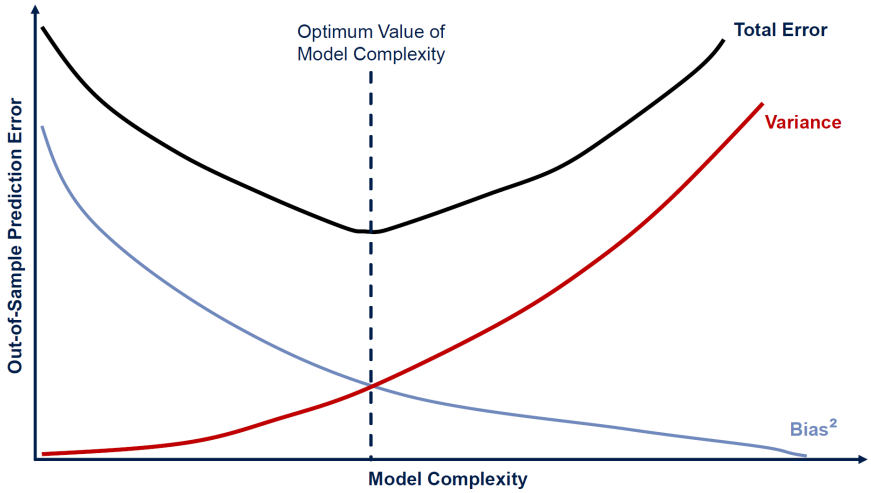

Figure 2.1: Sketch of the bias-variance tradeoff. If there are hyperparameters controlling for model complexity we see a U-shape in model performance. The optimal value of the hyperparameters is found at the minimum of the total error, where the sum of bias and variance is lowest. Note that on empirical data we can only assess the total error. Bias and variance can only be measured independently if the data generating process is known, e.g. the data was generated synthetically by a specific formula.

The bias

$$\text{Bias}\left[\hat{f}(x)\right] = \text{E}\left[\hat{f}(x) - f(x)\right] \tag{2.5}$$

is related to the number and amount of erroneous assumptions in the algorithm (e.g. that the relationship is linear).

The variance

$$\text{Var}\left[\hat{f}(x)\right] = \text{E}[\hat{f}(x)^2] - \Big(\text{E}[\hat{f}(x)]\Big)^2 \tag{2.6}$$

essentially measures the instability of the algorithm due to overfitting.

The irreducible error σ^2 corresponds to the inherant idiosyncratic stochastic noise which bounds the best prediction possible by any algorithm or procedure, given a set of covariates.

For classification problems the same deconstruction can be applied to the squared error of the probabilities. Analogous decompositions exist for the misclassifcation rate, which is defined as $1 - \texttt{accuracy}$ (Domingos 2000; Valentini and Dietterich 2004).

Generally more complex models have a tendency to overfit (thus having high variance) and, due to their flexibility, have a low bias. More simple models tend to

be more stable but can sometimes 'underfit' due to their erronous assumptions (e.g. linearity of the features). Typically, when we have a model with a hyperparameter controlling its flexibility (e.g. lasso) the out-of-sample error of the model forms a U-shape, meaning that there is an optimum flexibility were bias and variance are minimal (see figure 2.1). Empirically the same behaviour is observed for the area under the ROC curve or the mean absolute error; though in this case no explicit decomposition like above is possible.

See (T. Hastie, R. Tibshirani, and J. Friedman 2009) for a more thorough discussion.

2.3 Cross-Validation

k-fold cross-validation is a procedure for evaluating the expected *generalization error* of an algorithm (T. Hastie, R. Tibshirani, and J. Friedman 2009). It splits the data into k nonoverlapping equal-sized subsets, which are called folds. The algorithm to be evaluated is then trained on $(k-1)$ folds and evaluated on the remaining one, not used for training. This is repeated k-times with each fold acting as the *test set* once. Cross validation remedies the need to separate one specific test set from the data and provides a more stable estimate of prediction error when the k-losses are averaged. Often 5 to 10-fold cross validation is recommended (Leo Breiman and Spector 1992; Kohavi et al. 1995). Additionally the k-fold cross validation can be repeated multiple times using different splits which we will use for evaluating our algorithms (see chapter 5).

2.4 Metrics for Algorithm Evaluation

We already discussed the metric mean squared error (MSE) above. In the following we will briefly discuss all metrics used in this work (including the ones used for the supplementary results in the appendix). We will stick as closely as possible to the equations from the scikit-learn documentation, as this is the framework we use for implementing the algorithms and their evaluations (Pedregosa et al. 2011). The metrics will be displayed as taken over n_{test}, which will denote the respective test set or test cv-fold. Note that it is possible to measure the metrics presented

here on the training set as well. Throughout this work, when we speak about the metrics, this will always refer to test set results.

2.4.1 Evaluation of regression algorithms

The R-squared or $\texttt{R}^2$

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n_{\text{test}}}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n_{\text{test}}}(y_i - \bar{y})^2} = \frac{\mathrm{E}\left[(y - \hat{y})^2\right]}{\mathrm{Var}\left[y\right]} \tag{2.7}$$

where $\bar{y}$ denotes the mean target of the sample $\bar{y} = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} y_i$.

The R^2 ranges from 1.0, for a perfect prediction, to a dummy level of 0.0 if for all observations the expected value of y is predicted (evaluated on the test set), without considering the covariates at all. Note that it has no lower bound and can become infinitely negative if a model performs worse than this dummy prediction.

The explained variance (Exp. Var.)

$$explained_variance(y, \hat{y}) = 1 - \frac{Var\left[y - \hat{y}\right]}{Var\left[y\right]} \tag{2.8}$$

The explained variance ranges from 1.0 for a perfect prediction, explaining all the variance, to arbitrarily low values for bad predictions. It is closely related to the R^2 but the numerator is slightly different and measures the centralized second moment called variance. Compare this to R^2, which uses the raw second moment, thus replacing $\mathrm{Var}(X) = \mathrm{E}\left[(X - \mu)^2\right]$ with $\mathrm{E}\left[X^2\right]$, hence disregarding the mean of $(y - \hat{y})$.

2.4.2 Evaluation of classification algorithms

The accuracy

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \mathbf{1}(\hat{y}_i = y_i) \tag{2.9}$$

The accuracy counts the fraction of correct predictions, without considering with which rank ordering or certainty these predictions were made. This loss function is sometimes also called *0-1 loss*; in this context the normalization by n_{test} is often omitted.

		True condition		
		Positive	Negative	Total
Prediction	Positive	*True positive*	*False positive*	$tp + fp$
	Negative	*False negative*	*True negative*	$fn + tn$
	Total	$tp + fn$	$fp + tn$	N

Table 2.1: The confusion matrix

Area under the ROC curve and Somers' D The area under the ROC curve (abbreviated as AUROC) is a ranking metric, which summarizes the likelihood of a model predicting a higher probability for true positive cases than true negative cases. For example when talking about a credit risk model predicting default probabilities for customers of a given portfolio: In this case the AUROC corresponds to the probability that a randomly drawn non-defaulted customer is assigned with a better score than a randomly drawn defaulted customer. By means of a sheer random assignment one would obtain an AUROC of 50% which is thus the low baseline for any algorithm.

The AUROC is directly proportional to Somers' D:

$$\text{Somers' D} = 2 \cdot \text{AUROC} - 1 \tag{2.10}$$

In this metric the dummy performance of a random rank ordering is 0%, allowing for a better interpretation of the results. We convert all AUROC values to Somers' D in this thesis.

Confusion matrix For a binary classification problem we can categorize the predictions with regards to the actual conditions in a confusion matrix, see 2.1. Contrary to the ranking metrics the algorithm predicts the class to which it assigned the highest probability. Based on this matrix we can define the two metrics *recall* and *precision*.

The recall is defined as the fraction of all observations with an actual positive condition which the algorithm predicted correctly:

$$\text{Recall} = \frac{\text{True positive}}{\text{True positive} + \text{False negative}} \tag{2.11}$$

The precision quantifies the fraction which the algorithm correctly classified as positive as compared to all instances for which it predicted a positive condition:

$$\text{Precision} = \frac{\text{True positive}}{\text{True positive} + \text{False positive}} \tag{2.12}$$

F1 Score and weighted F1 Score The *F1 score* is now given as the harmonic mean of the recall and precision:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{2.13}$$

The harmonic mean penalizes a single low value more than the arithmetic mean would (Sasaki et al. 2007).

The F1 score can be generalized to multiclass classification settings in various ways. We will use the *weighted* F1 score, where we determine the F1 score for each class $l \in L$ independently by reducing it to a binary classification problem for this class only. We then average all F1 scores obtained using the number of observations in each class, $|\hat{y}_l|$, as a weight:

$$\frac{1}{\sum_{l \in L} |\hat{y}_l|} \sum_{l \in L} |\hat{y}_l| \, F_1(y_l, \hat{y}_l) \tag{2.14}$$

The terms y_l and $\hat{y}_l$ denote the predictions and targets for class l, which are converted to the binary problem of l vs non-l.

We think that in multiclass classification settings one is often not interested in a relative rank ordering between all the classes but more in an accurate labelling; e.g. when classifying products into different categories. The weighted F1 measure is therefore well suited for this task. In contrast for most binary classification problems the rank order is of practical concern; for example we might want to know which insurance claims are most likely to be fraudulent. The cases could then be ordered by their likeliness of fraud, as predicted by the algorithm, and a team could assess the cases in detail, starting with the cases most likely to be fraudulent and continuing in order through this list.

Since all models are wrong the scientist cannot obtain a "correct" one by excessive elaboration. On the contrary following William of Occam he should seek an economical description of natural phenomena. Just as the ability to devise simple but evocative models is the signature of the great scientist so overelaboration and overparameterization is often the mark of mediocrity.

— George E. P. Box (Box 1976)

3 Models and Feature Extractors

Contents

3.1 Overview 13
3.2 White Box Models 14
3.2.1 Regression 14
3.2.2 Classification 17
3.2.3 Regression and Classification 18
3.2.4 Decision trees 18
3.3 Extractors 19
3.3.1 Compact representation and manifolds methods 19
3.3.2 Extraction via decomposition, projection, and clustering 21
3.4 Black Box Models 23
3.4.1 Tree-based methods 24
3.5 Special Topics 26
3.5.1 Predicting multiple classes via softmax 26
3.5.2 The kernel trick 27

3.1 Overview

In this chapter we will discuss the models and the algorithms used to perform the feature extraction. Methods which are interpretable in terms of their inner workings will be called *white box methods*. This contrasts to the many modern non-linear prediction-type models such as neural networks and random forests. Most of these methods fit a general hyperplane to the data, which defies any simple economic interpretation. We will call these methods *black box models*.

The methods used for feature extraction together with the white box models are called *extractors*. We believe that these methods roughly approximate the manual steps of extracting features for a model. However, they most likely underestimate what could be achieved with a canny manual modelling and will potentially underestimate the performance of white box models.

Table 3.1 lists all algorithms used in this work. The technical name given in this table will be used in subsequent chapters to denote this model in plots and tables. In this chapter all algorithms in the table will be discussed in more detail.

3.2 White Box Models

3.2.1 Regression

Linear regression connects the target variable linearly to the features.

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j \tag{3.1}$$

The parameters β[2] are usually optimized by minimizing the *Residual Sum of Squares*:

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - f(x_i))^2 \tag{3.2}$$

In the case of linear regression this expression can be explicitly solved via the *method of least squares*. Thus a solution can be obtained which does not involve any recursive optimization procedures, as is the case with most other algorithms. Using this explicit formula linear regression was already used in the pre-computing age, starting with Legendre and Gauss who apparently both discovered the *method of least squares* independently.

Ridge regression, instead of minimizing the *residual sum of squares*, imposes a Euclidean norm penalty term:

$$\hat{\beta}^{ridge} = \underset{\beta}{\operatorname{argmin}} \; RSS(\beta) + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{3.3}$$

This effectively reduces the number of degrees of freedom for the algorithm, which often improves performance by alleviating correlations between variables and

[2]Note that we follow the usual convention for linear regression methods in denoting the parameters with β instead of the more general θ, introduced earlier.

Interpretability	Type	Name	Technical Name
White box	Regression	Linear Regression	*LinearRegression*
White box	Regression	Ridge Regression	*Ridge*
White box	Regression	Lasso	*Lasso*
White box	Regression	Elastic Net	*ElasticNet*
White box	Regression	Linear Support Vector Machine for Regression	*LinearSVR*
White box	Regression	k-Nearest Neighbors Algorithm	*KNeighborsRegressor*
White box	Regression	Decision Tree (Regression)	*DecisionTreeRegressor*
White box	Regression	Multivariate Adaptive Regression Splines (MARS)[1]	*MARS-Earth*
White box	Classification	Logistic Regression	*Logit*
White box	Classification	Logistic Ridge	*Logit Ridge*
White box	Classification	Logistic Lasso	*Logit Lasso*
White box	Classification	Naive Bayes with Gaussian density	*NaiveBayesGaussian*
White box	Classification	Linear Discriminant Analysis (LDA)	*LDA*
White box	Classification	Quadratic Discriminant Analysis (QDA)	*QDA*
White box	Classification	Linear Support Vector Classifier	*LinearSVC*
White box	Classification	k-Nearest Neighbors Algorithm	*KNeighborsClassifier*
White box	Classification	Decision Tree (Classification)	*DecisionTreeClassifier*
Black box	Regression	Kernel Support Vector Machine for Classification	*KernelSVC*
Black box	Regression	Neural Network, multilayer perceptron with one or two hidden layers	*NeuroNetRegressor*
Black box	Regression	Bagged Trees	*BaggedTrees*
Black box	Regression	Random Forest	*RandomForest*
Black box	Regression	Extra Trees	*ExtraTreesRegressor*
Black box	Regression	XGBoost	*XGBRegressor*
Black box	Classification	Kernel Support Vector Machine for Regression	*KernelSVR*
Black box	Classification	Neural Network, multilayer perceptron with one or two hidden layers	*NeuroNetClassifier*
Black box	Classification	Bagged Trees	*BaggedTrees*
Black box	Classification	Random Forest	*RandomForest*
Black box	Classification	Extra Trees	*ExtraTreesClassifier*
Black box	Classification	XGBoost	*XGBClassifier*
Extractor	All	None (Taking the raw features, without any extraction)	*None*
Extractor	All	Principal Component Analysis (PCA)	*PCA*
Extractor	All	Kernel Principal Component Analysis with the four different kernels, see 3.5.2	*KernelPCA-poly3, KernelPCA-rbf, KernelPCA-sigmoid, KernelPCA-cosine*
Extractor	All	Locally Linear Embedding (LLE)	*LLE-standard*
Extractor	All	Modified Locally Linear Embedding	*LLE-Modified*
Extractor	All	Isometric Mapping	*Isomap*
Extractor	All	Gaussian Random Projection	*GaussRandProj*
Extractor	All	K-Means Transformation	*k-Means*

Table 3.1: List of all algorithms along with their technical name, used later when plotting the results.

restricting the influence of outliers. It is also known as Tikhonov regularization after Andrey Tikhonov. The weight of the penalty term λ is usually determined using cross-validation.

The Lasso, short for least absolute shrinkage and selection operator, replaces the Euclidean norm with an absolute-value norm penalty term:

$$\hat{\beta}^{lasso} = \underset{\beta}{\operatorname{argmin}} \; RSS(\beta) + \lambda \sum_{j=1}^{p} |\beta_j| \tag{3.4}$$

This small replacement has substantial ramifications for the resulting parameters β. Instead of shrinking all parameters continuously the lasso penalty pushes certain variables to a zero weight, thus restricting the prediction to a subset of all available predictors[3]. Note the difference to ridge regression where all parameters usually remain present but may have a vanishingly small weight and impact on the prediction.

Elastic net combines the previous two algorithms and uses both penalties simultaneously (Zou and Trevor Hastie 2005):

$$\hat{\beta}^{elasticNet} = \underset{\beta}{\operatorname{argmin}} \; RSS(\beta) + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2 \tag{3.5}$$

The motivation is to combine the advantages of both lasso and ridge regression, thus using two hyperparameters λ_1 and λ_2 to weight the respective penalty terms.

Multivariate Adaptive Regression Splines (MARS) improve on the linearity assumption of linear regression by using *hinge* functions on the original input. Hinge functions are equal to its argument when the argument is positive:

$$\mathrm{h}\,(x - c) = \max(0, x - c) = \begin{cases} x - c, & x > c \\ 0, & x \leq c \end{cases} \tag{3.6}$$

For the variant increasing in positve directions, or

$$\mathrm{h}\,(c - x) = \max(0, c - x) \tag{3.7}$$

[3]Both Lasso and ridge regression essentially replace procedures for subset selection and provide a higher performance in terms of predictive power. For this reason we do not include subset selection algorithms in our evaluation.

for the variant increasing in negative directions. The location of the kink of the function is determined by c. Note that multiple hinge functions can be combined with different values of c to approximate arbitrarily non-linear functions.

We use the *py-earth* implementation of MARS, which is based on the original paper by Friedmann (J. H. Friedman 1991). MARS models are more flexible to non-linear input while still yielding an interpretable model. In this work we consider MARS to be a white box model which already includes a feature extractor and therefore don't combine it with the extractors discussed further below.

3.2.2 Classification

Logistic regression (also known as logistic model, or logit model) estimates the probability p of a given class membership by linearly approximating the log-odds of belonging to a given class:

$$\text{logit}(p) = \ln \frac{p}{1-p} = \beta_0 + \sum_{j=1}^{p} X_j \beta_j \tag{3.8}$$

Due to its strong ties to probability it is often used when well calibrated probabilities for estimating class-membership are required.

Logistic Lasso and Logistic Ridge: The same penalties used for linear regression models can also be applied to the logistic model. In this case the penalty terms are added to the log-likelihood, while fitting the model, instead of the RSS as for linear regression (note that for a linear regression model the RSS is equivalent to its log-likelihood under the squared-loss).

Naive Bayes assumes that all features are Gaussian distributed and are independent from each other (corresponding to a diagonal covariance matrix). It then utilizes Bayes' theorem to obtain a probability estimate for a given class.

Linear Discriminant Analysis (LDA) was developed by Sir Ronald Fisher to classify Iris flowers (Fisher 1936). It extents the naive Bayes model by allowing for off-diagonal elements in the covariance matrix. It assumes that the covariance is independent of a specific class and thus estimates the matrix over all observations. LDA results in a linear discrimination boundary between classes.

Quadratic Discriminant Analysis (QDA) allows for a different covariance matrix for each class. This significantly increases the number of parameters and allows for greater flexibility in the model, which sometimes improves but also often destabilizes the estimates (especially on small datasets) due to the potential over-fitting which such flexibility entails (see section 2.2).

3.2.3 Regression and Classification

***k*-Nearest Neighbours algorithm (k-NN)** makes a prediction based on the values of the k-training observations which are closest in feature space to the new observation to be evaluated. Of these values the average is taken to either determine a numeric value for a regression setting or the probability of class membership for classification problems. The average can either be taken uniformly or based on the inverse of the distance of a neighbour to the new observation, which is also determined during the hyperparameter optimization.

We determine the number of neighbours k during hyperparameter optimization, where we randomly select a value between 1 and half the number of observations, with a maximum of 500 possible neighbours for larger datasets.

3.2.4 Decision trees

A tree can be imagined as a set of convoluted if-then statements; thus, a single decision tree is generally easy to understand and interpret. It has been argued that they even resemble human decision making (James et al. 2013). Trees can be linearized into unnested sets of if-then statements with multiple conditions, which provides a different angle to interpret the tree rules (Quinlan 1987). In this arrangement their non-linearity and flexibility becomes especially apparent.

A decision tree can be used for classification or regression problems. In this context Breiman et al. promoted the term *Classification And Regression Tree (CART)* in their book (L. Breiman et al. 1984) together with a comprehensive algorithm to compute them.

The construction of an optimal decision tree is NP-hard, meaning that it cannot be found in polynomial time with regards to the number of observations n and number of variables m (Laurent and Rivest 1976). This is due to the fact that all splits of the tree depend on each other. For example it might be beneficial

to make a seemingly sub-optimal split on top to enable more powerful splits at the lower branches of the tree. As a remedy multiple procedures were developed to approximate a tree using certain 'greedy' assumptions, which speed up the otherwise nested search (T. Hastie, R. Tibshirani, and J. Friedman 2009). The CART algorithm seems to be the most prominent decision tree algorithm, which is also the one used by scikit-learn and therefore the one used in this thesis. Further all constructed trees are restricted to a maximum depth of 50, to prohibit the construction of overly-complex trees.

The structure of a single decision tree is often unstable, such that a small change in the data leads to a large change in tree-structure. In order to remedy these downsides (while still retaining the flexibility of trees) multiple algorithms were developed which use decision trees as their fundamental building blocks. This aggregation of multiple trees leads to a loss of interpretability which is the reason why we dedicated a separate section 3.4.1 on the tree-based methods. We consider a single tree to be an interpretable white box model, but all aggregations of multiple trees to belong to the class of black box models.

3.3 Extractors

3.3.1 Compact representation and manifolds methods

High dimensional data is usually not evenly spread across all possible values, but rather is embedded on a sub-dimensional manifold. This is especially apparent for images, where a completely "random image" looks like white noise from an old TV screen, and has little resemblance to a photograph of a beach or a person. Therefore the set of all photographs is just a small subset/manifold of all (mathematical possible) images (Brand 2003). Discovering such a manifold can lead us to useful features and ease the work required by the machine learning methods. For example consider a motorway: If we wish to refer to a certain position in the road we usually do this by specifying a road with a distance (e.g. as on a driver location sign "M40 A 75.8"). Actually a road is not a one dimensional object but rather a 2-dimensional object on a map or a 3-dimensional object with regards to the landscape[4] (Goodfellow et al. 2016). If we would train an algorithm to predict

[4]or even a 4-dimensional structure if we would include temporal aspects in our consideration

traffic jams on such a road, but input the data by means of (x,y,z) coordinates, the algorithm will probably struggle to make useful predictions as it might spend most of its training time trying to assess the exact road structure.

In applying these methods we tried to reduce the dimensionality by just a small amount, as we are not dealing with particularly high dimensional data, and we rather intend them to provide an additional layer of useful feature extraction. Often these methods are used for a reduction to only two or three dimensions for visualization of high dimensional datasets. As to our knowledge, we are the first to use some of these methods as a mean of feature extraction on structured (spreadsheet like) datasets.

Isomap is an extension of multidimensional scaling (MDS). First, it constructs a graph representation of the data by creating an edge between two points in a certain neighbourhood. It then searches a low dimensional representation of the data which retains most of the geodesic distance properties[5] of the original data (Tenenbaum, De Silva, and Langford 2000). In this work we use 10 neighbours for each neighbourhood and reduce the dimensionality by 20% of the original number of features.

Locally Linear Embedding (LLE) starts with a neighbourhood of the closest points for each observation. Each observation is then essentially interpolated by its neighbouring observations. The lower-dimensional representation is chosen in a way to retain most of this interpolational fit (Roweis and Saul 2000). As for isomap we use 10 neighbours for each neighbourhood and reduced the dimensionality by 20% of the original number of features.

Modified LLE We also employ a modified variant of LLE, which claims to be more stable than the original method (Z. Zhang and Wang 2007). We attempt to use a dataset dependent neighbourhood of half the number of features[6] and reduce the dimensionality by 50%.

[5]Geodesic distance is the minimum number of edges which connect two points.
[6]In this case this number is evaluated before the one-hot encoding of categorical features.

3.3.2 Extraction via decomposition, projection, and clustering

One might argue that the manifold methods are not well suited for our task at hand. Often feature extraction can be achieved in a simpler way. For example it might be the case that a feature, instead of being linear, follows a U-curve with regards to the target. In this case the feature has to be linearized, e.g. by transforming it with a low-order polynomial. A practical example of this would be the annual growth in turnover of a corporation. While creditors with a low rate of growth are obviously at risk a corporation might also be risky if it is growing too fast (Henking, Bluhm, and Fahrmeir 2006). Often features can also be hidden "in-between" the variables, e.g. the monetary amounts of equity might only be weakly correlated with bankruptcy risk while the equity ratio (its quotient with regards to the total balance sheet sum) shows a strong correlation. We hypothesize that the principal component based methods should resonate more with these common transformations.

Principal Component Analysis (PCA) is a linear transformation which maps a set of features to a linearly uncorrelated set of new features, called principal components (Shlens 2014). The principal components are ordered by the relative variance of the features they can explain. In order to reduce the dimension of the new features only the first principal components are retained, until a sufficient variance of the data can be explained by these components. In our implementation we determine the number by the threshold of at least 95% explained variance[7].

Kernel-PCA: The kernel trick (see section 3.5.2) can be applied to PCA (Schölkopf, Smola, and Müller 1998). In this way a non-linear decomposition/feature extraction is possible. We retain all resulting components which have a non-zero variance.

Gaussian random projection constructs a random projection matrix to project the original features to a new space. Often the method for constructing the random projection matrix is based on a Gaussian distribution[8] (Bingham and

[7]Note that this refers to the explained variance of the covariates X. We will later evaluate algorithms based on the explained variance of the target y compared to its prediction $\hat{y}$.

[8]There exist another method which constructs sparse matrices. As our data has a lower feature-dimension compared to text and image data we did not attempt such an extreme reduction as in the sparse method.

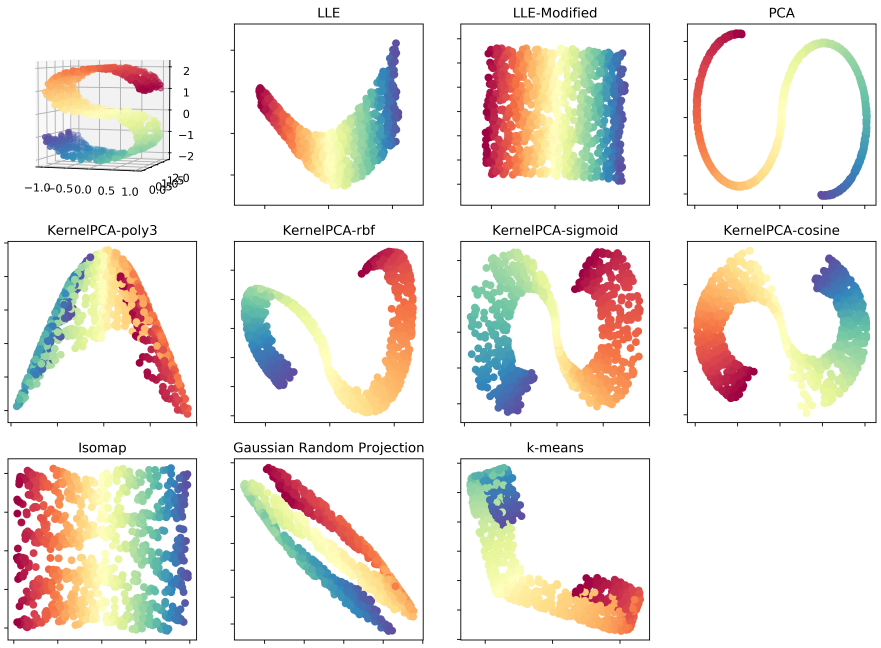

Figure 3.1: The feature extractors used in this thesis are applied to a 3-dimensional S-curve. For this example all feature extractors are reconfigured to extract exactly 2 components. Via colour coding the effect of the transformation can be visualized. This figure was inspired by a similar plot from Jake Vanderplas.

Mannila 2001; Dasgupta 2013). We reduced the dimensionality by 20% of the original number of features.

***k*-means clustering** is one of the most common clustering algorithms used. A certain number of random points is drawn in feature space, each representing the centroid of a potential cluster. The points are then moved with respect to the data until convergence is achieved[9]. The algorithm often gets stuck in local minima; therefore the algorithm is usually run multiple times with different random centroid initialisations and the clustering with the lowest inertia (sum of squared distances of samples to their closest cluster centre) is retained.

The algorithm can also be used as a feature extractor by calculating the distance of each observation to each of the k-cluster centroids and using these as new features

[9]As convergence is not guaranteed and oscillations are possible the procedure is stopped after 300 iterations.

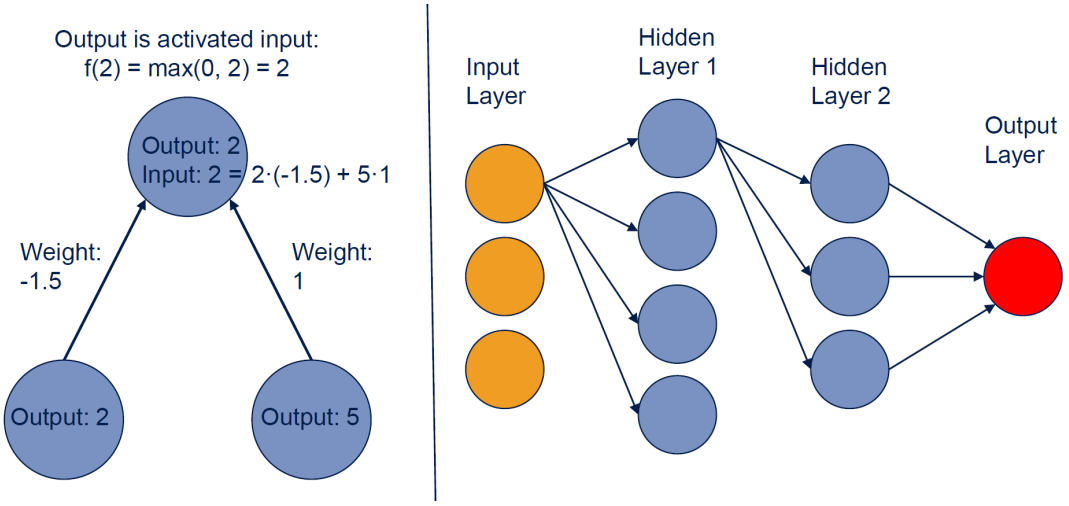

Figure 3.2: Figure of a neural network: On the left we show a simple neuron interaction using the 'ReLU' activation function. The right side depicts the structure of a simple feed forward network with two hidden layers. Each neuron is connected to all neurons of the following layer. In the figure only the connections for one neuron are shown in order to avoid an occlusion of the arrows.

(Coates and Ng 2012). We used 8 clusters/centroids on all dataset where we used the elbow method to check that these values are in a reasonable range for every datasets. Each observation is then mapped to the 8-dimensional space of centroid-distances.

3.4 Black Box Models

Neural networks comprise a class of algorithms vaguely inspired by biological brain functions. Their functional building blocks are artificial neurons. Each neuron sums its inputs, which are coming from other neurons, and are weighted by a connection strength, that is specific for each neuron pair and can also be negative. To this summed input it applies a non-linear transformation, called *activation function*.

Usually the neurons are organized in layers which take the previous layer as its input and connect to the next layer as its output. Figure 3.2 shows the structure of a simple feed forward network. The input layer is the first layer of a network; its values are set by the features of an observation, one variable represented by each input neuron. The output layer, often just a single neuron, contains the prediction of the neural network. The parameters of this model are its connections and their weights. They are usually trained using the backpropagation algorithm (Rumelhart, Hinton, and Williams 1986).

In this work we use multilayer perceptrons (MLP), which are simple feed forward networks with at least one hidden layer (T. Hastie, R. Tibshirani, and J. Friedman 2009). We use the high performing and popular rectifier 'ReLU' activation function $f(x) = \max(0, x)$, where x is the input to a neuron (Goodfellow et al. 2016). During the hyperparameter optimization different network structures are considered and it is determined whether the network should have one or two hidden layers and how many neurons they should each contain. Generally it is a rule of thumb that (in case of an MLP on structured datasets) one should rather increase the number of hidden units in each layer instead of adding more layers. Additionally we use an *l2* regularization parameter 'alpha' to prevent overfitting, which is also part of the hyperparameter optimization.

Support Vector Machines (SVM) try to find the best separating hyperplane between two classes – in this context often called *margin* (Vapnik and Chervonenkis 1974). Kernel methods (see 3.5.2) can be used to make the hyperplanes non-linear in feature space (Boser, Guyon, and Vapnik 1992). This greatly increases the flexibility of the algorithm while defying any easy interpretation of the resulting hyperplane discrimination boundary. It can be shown that there is a close connection between the logit model and the linear SVM (T. Hastie, R. Tibshirani, and J. Friedman 2009). We therefore consider the linear-SVM to be a white box and the non-linear kernel SVMs to be black box models.

While originally SVMs where only designed for classification problems the algorithm can be adapted for regression settings (Drucker et al. 1997).

In case of multiclass classification problems we train the kernel SVMs one-against-one, and the linear SVMs with one-against-rest as recommended by the sklearn default settings.

SVMs are usually not well calibrated (Platt et al. 1999). However as long as we only use a ranking-order metric we can use the closest distance to the decision boundary as the relative ranking metric.

3.4.1 Tree-based methods

In the following we will discuss the different methods for aggregating multiple trees.

Bagged trees, where bagging is short for bootstrap aggregation, combines the prediction of multiple trees. First conceived by Leo Breimann (Leo Breiman 1996a) it trains each tree on a different bootstrap sample of the dataset. In order to make a prediction a new observation is evaluated with all trees and these predictions are then averaged[10] in order to obtain an aggregated forecast.

Random forests, additionally to fitting each tree on a bootstrap sample, restrict the considered predictors for each split in a tree to a certain fixed fraction (Leo Breiman 2001). This lowers the bias while increasing the variance of the trees and ensures that all predictors are utilized[11]. As it is the variance which is reduced through the bootstrap the lower bias usually results in better algorithms.

Extremely randomized trees often abbreviated as extra trees, try to increase the variance of its constructed trees, just like a random forest, but using a different source of randomness. For each variable a random point is chosen for a potential split. The best variable–random-split pair is then selected based on a performance metric (often Gini impurity). Compare this to the usual CART algorithm, which tries to find the best split for each variable and then selects the optimal variable–best-split pair. In its original version the tree construction is performed over the whole training set (Geurts, Ernst, and Wehenkel 2006). Similarly to random forests only a fraction of the total variables is considered at each split.

Boosted trees: Boosting starts with a weak model (e.g. a shallow tree or tree stump) as a fundamental building block. A weak models is defined as a model which produces an estimate which is only weakly correlated with the target variable, thus the name weak learner. The idea is to fit a chain of multiple such models to the data, where each subsequent model tries to compensate for the weaknesses of the others (T. Hastie, R. Tibshirani, and J. Friedman 2009). This can either be achieved by giving a weight to each observations which is then increased for entries with a higher remaining error, and decreased for observations with a lower error. This idea is called Adaptive Boosting, AdaBoost for short, and was first proposed

[10] In our context, we either average the value for regression problems or the probability in case of classification problems.

[11] Otherwise a strong predictor might always be the one chosen for the first split by the greedy tree-construction algorithm

by (Freund and Schapire 1997). Another possibility is to fit a subsequent model to the residuals of the previous one. This is called Gradient Boosting, with the first procedure proposed by (J. H. Friedman 2001; J. H. Friedman 2002).

Note that this dependency of the learners can lead to overfitting if too many models are constructed. For boosting this number is therefore determined via hyperparameter optimization. Compare this to bagging (and its derivatives extra trees and random forest), where increasing the number of trees cannot overfit, as the trees are independent from each other.

In this work we use the *xgboost* implementation (Chen and Guestrin 2016) of gradient boosting. It is one of the most popular boosting algorithms and is one of the top methods used in Kaggle competitions, after its debut in winning the *Higgs Machine Learning Challenge*[12] (Chen and He 2015).

Number of trees for bagged trees, random forests, and extra trees: In order to assess a sufficient number of trees, we calculate the out-of-bag scores while continually increasing the number of trees (Leo Breiman 1996b) for all methods over all datasets (using recommended default values for the hyperparameters of random forest and extra trees). Including a high margin of safety we set the number of trees to 2,000 for all three methods and all datasets.

3.5 Special Topics

3.5.1 Predicting multiple classes via softmax

If, in a *multiclass classification problem*, more than two classes have to be predicted the softmax function is used to determine the predicted probabilities for each class:

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\top \mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\top \mathbf{w}_k}} \tag{3.9}$$

where X is either the vector of covariates or the output of the previous hidden layer (in case of neural networks).

For example in logistic regression $\mathbf{w}_k$ corresponds to the weights of a single logit model, which is fitted via modelling one-vs-rest (k vs non-k). In this way one logit model is fitted for each class and then the final prediction of class probabilities is

[12] https://atlas.cern/updates/atlas-news/machine-learning-wins-higgs-challenge

calculated with the above expression. In case of a neural network the network has multiple terminal nodes, one for each class, each connected with different weight vectors $\mathbf{w}_k$ to the same hidden layer, which provides the vector $\mathbf{x}$.

Note that for support vector machines we do not use the softmax function, see section 3.4. Also tree based models are capable of directly estimating multiple classes and do not require multiple models to be fitted.

3.5.2 The kernel trick

The kernel trick can be used to effectively increase the feature space in which a possible solution can be found. It is usually applied to support vector machine and principal component analysis. Both of these models can be easily be reformulated to contain linear kernels:

$$k(x_i, x_\nu) = {x_i}^\top x_\nu \tag{3.10}$$

where x_ν is the new point to be evaluated and x_i a known point from the training set. The whole SVM or PCA can be expressed as a weighted sum over all linear kernels for each training observation i.

We can now replace the linear kernel by a different non-linear kernel, which might be better at approximating the data. Several choices are available:

Polynomial kernel:

$$k(x_i, x_\nu) = (\gamma {x_i}^\top x_\nu + c_0)^d \tag{3.11}$$

Sigmoid kernel:

$$k(x_i, x_\nu) = \tanh(\gamma {x_i}^\top x_\nu + c_0) \tag{3.12}$$

Radial basis function (RBF) kernel:

$$k(x_i, x_\nu) = \exp(-\gamma \|x_i - x_\nu\|^2) \tag{3.13}$$

where for the kernel parameter γ we take the fixed value $\frac{1}{n_{features}}$ and, following sklearn's default values we take $c_0 = 0.0$ for the support vector machines and $c_0 = 1.0$ for the kernel-PCA[13]. For the degree of the polynomial kernel we take $d = 3$.

[13]Here $n_{features}$ is taken after one-hot encoding of the categorical features. The variables are all standardized (with mean zero and variance one) before the kernel is applied. For more details on the processing steps see chapter 4.

Cosine similarity kernel:

$$k(x_i, x_\nu) = \frac{x_i x_\nu^\top}{\|x_i\| \|x_\nu\|} \tag{3.14}$$

This kernel computes the vector similarity of two observations. It is used in the feature extractor *KernelPCA-cosine*.

Alles Gescheite ist schon gedacht worden.
Man muss nur versuchen, es noch einmal zu denken.

All intelligent thoughts have already been thought;
what is necessary is only to try to think them again.

— Johann Wolfgang von Goethe – Wilhelm Meister's Wanderjahre (Journeyman Years) (1829)

4

Datasets, Pipeline and Hyperparameters

Contents

4.1 Datasets 29
4.2 Processing Pipeline 31
4.3 Hyperparameter Optimization 33
4.3.1 A note about nested cross validation 34
4.3.2 Setup of the random search 35

4.1 Datasets

We use 32 datasets mostly from the *UCI Machine Learning Repository* (Dheeru and Karra Taniskidou 2017). Some are (training) datasets from Kaggle[1]. The *AmesHousing* dataset was taken directly from (De Cock 2011). The *BostonHousing2* dataset contains corrected values and additional spatial information compared to the original *BostonHousing* dataset and was taken from the StatLib website[2].

Regression datasets The datasets span a wide range of topics: Six of the regression datasets are conernced with predicting house prices (*Housing* as part of the name), two with the prediction of automobile prices or qualities (*Automobile,*

[1]https://www.kaggle.com/

[2]http://math.furman.edu/ dcs/courses/math47/R/library/mlbench/html/BostonHousing.html, http://lib.stat.cmu.edu/datasets/

AutoMPG), one with the prediction of wine quality (*WineQuality*), one predicting the compressive strenght of concrete based on its components (*ConcreteCompressiveStrength*), and another one with determining the age of 'Abalone' sea snails (*Abalone*).

Binary classification datasets Eight of the binary classification problems have a direct financial context: Predicting if a credit is bad (*GermanCredit*), if a credit card application was approved (*AustralianCredit*), identifying fraudulent credit card transactions (*CreditCardFraud*), predicting whether a customer will default on his credit card debt (*TaiwanCreditCardDefault*), if a customer will buy insurance for his caravan (*CaravanInsurance*), if a person is a high earner (*AdultHighEarnerUCI*), the success of a banks marketing campaign (*BankMarketingUCI*), and whether a driver will initiate an auto insurance claim in the next year (*PortoSeguro*).

Four have a more biological context with predicting heart disease (*HeartStatlog*), if a horse has a colic (*HorseColic*), if a breast cancer is benign (*BreastCancerWisconsinDiagnostic*), and if a mushroom is poisonous or edible (*Mushroom*).

Another four are about predicting if a Titanic passenger survived (*TitanicKaggle*), if a chess game can still be won by white (*ChessKRKPA7*), if a heavy Scania truck will have an air pressure system failure (*AirScaniaTrucks*), and if the former NBA player Kobe Bryant was able to score a certain basketball shot (*KobeBryantShots*).

Multiclass classification datasets The five multiclass classification datasets are about categorizing: The type of wine based on its chemical components (*Wine*), types of glass based on its chemical components (*Glass*), if a scale will be in balance or swing to the left or right (*BalanceScale*), the quality of a used car (*CarEvaluation*), and categorizing products from the online retailer Otto into different categories (*OttoGroupProductClassChallenge*).

Some further notes on the datasets and the exact longlist of variables used for the models can be found in appendix C.

In order to decrease the computational costs required for our comparison we downsample each larger dataset to 10,000 observations by randomly drawing from the original dataset without replacement[3].

[3]We could have used the remaining observations to form a test set. Nevertheless we chose to just use the downsampled data and apply cross-validation to it (as is described later). This

	Problem Type	Dataset	Observations	Metric Features	Categorical Features
1	Binary Classif.	HeartStatlog	270	6	7
2	Binary Classif.	HorseColic	366	7	14
3	Binary Classif.	BreastCancerWisconsinDiagnostic	569	30	0
4	Binary Classif.	AustralianCredit	690	6	8
5	Binary Classif.	TitanicKaggle	891	4	3
6	Binary Classif.	GermanCredit	1000	6	14
7	Binary Classif.	ChessKRKPA7	3196	0	36
8	Binary Classif.	Mushroom	8124	0	22
9	Binary Classif.	CreditCardFraud	9492	29	0
10	Binary Classif.	CaravanInsurance	9822	84	1
11	Binary Classif.	AirScaniaTrucks	10000	170	0
12	Binary Classif.	KobeBryantShots	25697	8	8
13	Binary Classif.	TaiwanCreditCardDefault	30000	20	3
14	Binary Classif.	AdultHighEarnerUCI	32561	6	8
15	Binary Classif.	BankMarketingUCI	45211	7	9
16	Binary Classif.	PortoSeguro	595212	26	31
17	Multiclass Classif.	Wine	178	13	0
18	Multiclass Classif.	Glass	214	9	0
19	Multiclass Classif.	BalanceScale	635	4	0
20	Multiclass Classif.	CarEvaluation	1728	0	6
21	Multiclass Classif.	OttoGroupProductClassChallenge	61878	93	0
22	Regression	Automobile	205	15	9
23	Regression	AutoMPG	398	5	2
24	Regression	BostonHousing2	506	14	3
25	Regression	ConcreteCompressiveStrength	1030	8	0
26	Regression	AmesHousing	2930	27	52
27	Regression	Abalone	4177	7	1
28	Regression	WineQuality	6497	11	1
29	Regression	MelbourneHousing	8895	8	3
30	Regression	CaliforniaHousing	20640	8	0
31	Regression	KingCountyHousing	21613	16	1
32	Regression	SberbankRussianHousingMarket	30471	7	4

Table 4.1: List of all datasets with their key characteristics.

4.2 Processing Pipeline

We set up a preprocessing pipeline (Géron 2017), using the python framework scikit-learn (Pedregosa et al. 2011), as summarized in figure 4.1. Note that if we speak about categorical variables this includes all ordinal variables as well. Ordinal variables are categories which can be ordered (e.g. low, medium and high). They are treated in exactly the same fashion as categories which cannot be ordered (e.g. colours).

guarantees that all results, which are later aggregated, were obtained with the same methodology.

The metric variables are first imputed with regards to missing values by the median. The median is used here to dampen the effect potential outliers might have on values such as the mean. After imputing the values are standardized to mean zero and variance one.

The categorical variables are one-hot encoded. One-hot encoding was chosen due to its simplicity and because every extractor or model is able to deal with one-hot encoded values. Many algorithms automatically convert categorical variables to effectively one-hot encoded categories anyway (e.g. linear regression or the logit model). For tree based methods there seems to be no large difference between the two methods (Kuhn and Johnson 2013), while one-hot encoding ensures that a split is evaluated for each different category, as each category comprises its own new feature.

The one-hot encoded categorical variables are also standardized to make them compatible with the enhancers and models. In this way we can ensure that the categorical variables don't dominate the extractors and model coefficients, but are on an equal footing with the metric features (Robert Tibshirani 1997)[4]

Extractor stage: The two processing streams are merged again into one dataset. The extractor, if applied at all, is applied now and afterwards the transformed features are passed on to the respective model.

The input for the pipeline is each respective training set (or in the case of cross validation all different training fold variants). In this way it can be ensured that even the features x of the test set do not enter the evaluation by providing a better estimate of the median-imputing or for the extractors used. In case of the categorical and ordinal variables a certain value of a feature might not be present on a certain CV-fold and lead to different effective numbers of parameters of the model (by means of the one-hot encoding), thus effectively testing the feature extractors and structures of the dataset for robustness.

[4]As already stated in the cited work which first promoted this procedure: *"The lasso method requires initial standardization of the regressors, so that the penalization scheme is fair to all regressors. For categorical regressors, one codes the regressor with dummy variables and then standardizes the dummy variables. As pointed out by a referee, however, the relative scaling between continuous and categorical variables in this scheme can be somewhat arbitrary."* In our case this step is especially critical, because otherwise the extractor algorithms would all be dominated by the categorical variables.

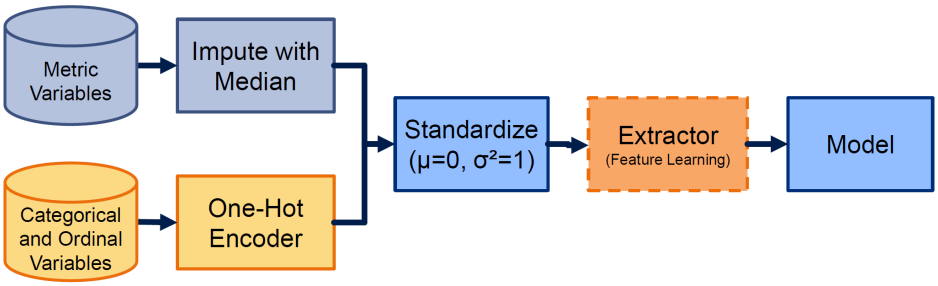

Figure 4.1: The data processing pipeline. Each metric feature is first imputed with the median and then standardized. Each categorical feature is one-hot encoded and then also standardized to put it on an equal footing with the metric features. The metric and categorical features are merged again, a feature extractor is used and then the model is applied to the data.

4.3 Hyperparameter Optimization

Hyperparameter optimization is concerned with finding appropriate and useful hyperparameters for an algorithm on a given dataset. The evaluation of a given set of hyperparameters is usually performed on a separate test set or via CV (Claesen and De Moor 2015). Usually the hyperparameters to be evaluated are either chosen

i) via manual guidance

ii) by grid search, which evaluates each point on a defined grid structure of potentially reasonable values

iii) or by a random search, which randomly draws values of a reasonable distribution for each hyperparameter.

Random search generally appears to be more efficient than grid search, can be more easily scaled with the given computational power, and still functions well if low-impact hyperparameters are added to the search, see figure 4.2, (Bergstra and Bengio 2012; Mantovani et al. 2015). As a manually guided search might potentially bias our comparison and make it hard to reproduce, we pursue randomized search. It was shown by (Bergstra and Bengio 2012) that evaluating 60 sets of hyperparameters via random search has a 95%-probability of finding hyperparameters within the top-5% optimum. We therefore also run our optimization over 60 iterations of randomly drawn hyperparameters.

The distributions from which we draw the hyperparameters are described in appendix C.

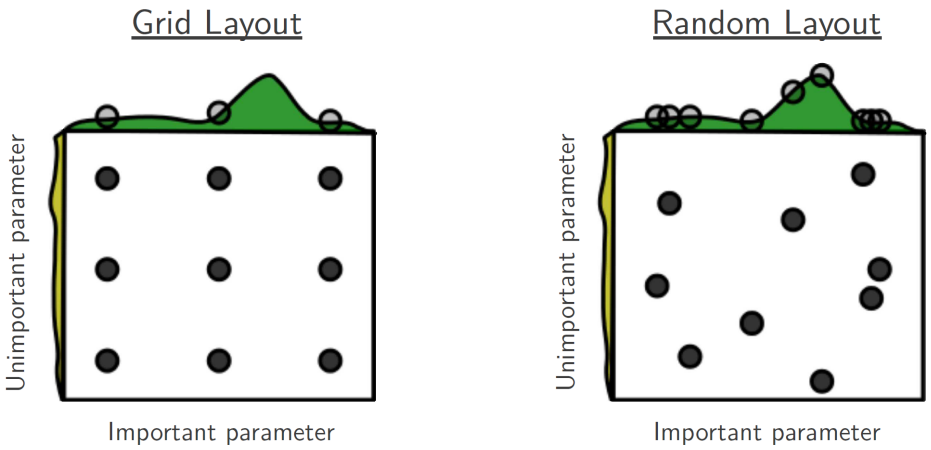

Figure 4.2: An example of the advantages of using random search. Even when one hyperparameter is present in the search with low impact on predictive performance no computational cost is lost, as still 9 different values are evaluated for the high-impact hyperparameter. Compare this to the grid search where only 3 values for the high-impact hyperparameter are evaluated. This figure was taken from (Bergstra and Bengio 2012).

4.3.1 A note about nested cross validation

As pointed out by (Varma and Simon 2006; Cawley and Talbot 2010) evaluating a classifier on the same test set or CV-folds as the hyperparameter optimization can lead to a leakage of the test set into the training data with the hyperparameters acting as conveyors of knowledge. (Varma and Simon 2006) proposed the computationally intensive remedy of nested cross validation, which essentially squares the needed computations for hyperparameter optimization. However, as was pointed out by (Kuhn and Johnson 2013) and can be seen in the analysis of (Krstajic et al. 2014), the effect becomes almost unobservable for large datasets and seems to be only significant for problems where $m > n$ or[5] $m \approx n$, i.e. data which has a similar number of features and observations. This situation often occurs in genetic microarray studies (T. Hastie, R. Tibshirani, and J. Friedman 2009). As datasets with these properties are not present in our analysis we do not consider it necessary to use nested cross validation. In order to account for the possibility that the hyperparameters might be overfitted on the test-set or CV-splits of the hyperparameter optimization we use different splits for the final model evaluation.

[5]As a reminder: n is the number of observations in the dataset, m is the number of features.

4.3.2 Setup of the random search

We evaluate the random search on

- A 5-fold cross validation for datasets with less than 2,000 observations
- On a 80/20 train-test-split for larger datasets

The last rule ensures that at least 400 observations are present in the test set and therefore the results are more stable (by the law of large numbers), whereas the former rule flattens out fluctuations over all folds for small datasets. As for the purpose of hyperparameter optimization, we are only interested in the shape of the parameter landscape (finding the optimal point). If the whole landscape is shifted, e.g. by the test set containing more challenging points, this would not change the value of the optimal hyperparameter while still allowing us to save $\frac{4}{5}$ of the computational power otherwise required.

The splits are drawn using stratification for classification problems, as suggested by (Kohavi et al. 1995).

The metric used to determine the best set of hyperparameters is:

- Explained variance for regression problems
- Area under the ROC curve for binary classification problems, which we convert to Somers' D
- F1-score evaluated and weighted by each class and its count for multiclass classification problems.

The metrics are discussed in section 2.4. For a more detailed discussion of evaluation metrics see (Kuhn and Johnson 2013; T. Hastie, R. Tibshirani, and J. Friedman 2009). The same metrics are used for the final evaluation of the models.

The area under the ROC curve (AUROC) is rescaled to the *Somers' D* coefficient via (Somers' D $= 2 \cdot \text{AUROC} - 1$). This ensures that 0% is the dummy performance for binary classification and makes the results more accessible.

In our comparison of binary classification algorithms we only focus on the relative ordering of outcomes. In practice one is often (also) interested in the exact assignment of probabilities for each class outcome. Some algorithms already produce

well calibrated probabilities while others require an additional calibration, e.g. by using using the standard calibration techniques of *isotonic regression* and *Platt scaling* (Niculescu-Mizil and Caruana 2005). The influence of such calibration on the rank ordering metrics is usually negligible (as can be seen in the result tables of (Caruana and Niculescu-Mizil 2006)). For this reason we do include any calibration in this work but compare the models in terms of their ordering abilities only.

After the search for an algorithm on a dataset is completed the best value of hyperaparameters is used for its final evaluation, as described in the next chapter.

"Let the jury consider their verdict," the King said, for about the twentieth time that day.
"No, no!" said the Queen. "Sentence first—verdict afterwards."
"Stuff and nonsense!" said Alice loudly. "The idea of having the sentence first!"
"Hold your tongue!" said the Queen, turning purple.
"I won't!" said Alice.
"Off with her head!" the Queen shouted at the top of her voice. Nobody moved.
"Who cares for you?" said Alice, (she had grown to her full size by this time.) "You're nothing but a pack of cards!"

— Lewis Carroll – Alice's Adventures in Wonderland (1865)

5 Comparison of Model Performance

Contents

5.1 Methodology of the Comparison 37
5.2 Comparison of Relative Ranking 38
5.3 Comparison of Normalized Scores 42
5.3.1 Normalization of the results 42
5.3.2 Discussion of normalized results 44

5.1 Methodology of the Comparison

We evaluate each algorithm on 10 repeats of 10-fold cross-validation. We chose repeated cross validation based on the studies by (Molinaro, Simon, and Pfeiffer 2005) and (Kim 2009), which both compared different model evaluation strategies. We thus obtain 100 values of each metric per datasets. The structure can be seen in table 5.1. Each algorithm is configured with the set of optimal hyperparameters (as discussed in the last chapter).

In order to assess the performance of the algorithms we use two comparisons inspired by the previous literature: First we perform a comparison of all algorithms based on their relative ranking per dataset, presented in the next section 5.2. The subsequent section 5.3 presents a comparison based on a normalized score for each

Dataset	Algorithm	Extractor	Metric	Fold-id	Result
A	LinearRegression	None	r2	1	0.876
			r2	...	...
			r2	100	0.678
		PCA	r2	1	0.714
			r2	...	...
			r2	100	0.721
	...	...	...	...	...
	RandomForest	None	r2	1	0.913
			r2	...	...
			r2	100	0.789
B	LinearRegression	None	r2	1	0.876
...	...	...	...	...	...

Table 5.1: Example structure of the final CV-evaluation over all datasets, algorithms, extractors and CV-folds.

metric. This normalization harmonizes the scores over all datasets and makes an aggregation and comparison possible.

5.2 Comparison of Relative Ranking

In order to first assess the algorithms we award medals per dataset to the models which performed best. The model with the best mean performance over all folds on a dataset is awarded a gold medal. The second and third are rewarded a silver and bronze medal. The medal table over all models can be seen in table 5.2. The algorithms are sorted by medal counts sorting for gold medals first, followed by silver and bronze[1]. All algorithms which won at least one medal are shown.

For regression problems the xgboost and extra trees dominate the results with 4 gold medals each. The three white box models ridge regression, lasso and linear SVM were able to score medals with ridge regression even winning one gold medal. The random forest showed a very consistent performance, scoring 7 silver and 1 bronze medal. Out of the 33 medals awarded for regression problems 5 were won by white box algorithms, yielding a rate of 15%, and 28 by black box methods. In terms of the medal count regression problems were clearly dominated by the black box methods, which won 85% of all medals.

[1]This is similar to the sorting used in sport events like the Olympic Games.

Problem Type	Metric	Model	BlackBox	Gold	Silver	Bronze
Regression	Exp. Var	XGBRegressor	y	4	1	1
Regression	Exp. Var	ExtraTreesRegressor	y	4	0	1
Regression	Exp. Var	KernelSVR	y	2	1	1
Regression	Exp. Var	Ridge	n	1	0	1
Regression	Exp. Var	RandomForest	y	0	7	1
Regression	Exp. Var	LinearSVR	n	0	1	1
Regression	Exp. Var	Lasso	n	0	1	0
Regression	Exp. Var	BaggedTrees	y	0	0	4
Regression	Exp. Var	NeuroNetRegressor	y	0	0	1
Multiclass Classif.	F1 Weighted	NeuroNetClassifier	y	1	1	0
Multiclass Classif.	F1 Weighted	LinearSVC	n	1	0	1
Multiclass Classif.	F1 Weighted	QDA	n	1	0	0
Multiclass Classif.	F1 Weighted	RandomForest	y	1	0	0
Multiclass Classif.	F1 Weighted	XGBClassifier	y	1	0	0
Multiclass Classif.	F1 Weighted	ExtraTreesClassifier	y	0	1	2
Multiclass Classif.	F1 Weighted	BaggedTrees	y	0	1	0
Multiclass Classif.	F1 Weighted	KernelSVC	y	0	1	0
Multiclass Classif.	F1 Weighted	LDA	n	0	1	0
Multiclass Classif.	F1 Weighted	Logit Lasso	n	0	0	1
Multiclass Classif.	F1 Weighted	Logit Ridge	n	0	0	1
Binary Classif.	Somers' D	XGBClassifier	y	7	1	1
Binary Classif.	Somers' D	RandomForest	y	3	3	2
Binary Classif.	Somers' D	LinearSVC	n	3	2	5
Binary Classif.	Somers' D	Logit Ridge	n	1	4	4
Binary Classif.	Somers' D	BaggedTrees	y	1	2	2
Binary Classif.	Somers' D	ExtraTreesClassifier	y	1	0	1
Binary Classif.	Somers' D	NeuroNetClassifier	y	0	2	0
Binary Classif.	Somers' D	KNeighborsClassifier	n	0	1	0
Binary Classif.	Somers' D	Logit	n	0	1	0
Binary Classif.	Somers' D	KernelSVC	y	0	0	1

Table 5.2: Overall medals

For multiclass classification 15 medals were awarded with 6 going to white box methods, yielding a rate of 40%, and 9 going to black box methods. The two white box methods QDA and linear SVC each won 1 gold medal.

Out of the 48 models awarded for binary classification 21 were won by white box, yielding a rate of 44%, and 27 by black box models. The linear SVC and logistic ridge methods performed well and scored 19 of the 21 medals awarded for interpretable models. Thus for binary and multiclass classification the gap between white box and black box methods was close in terms of medal count.

The dominance of xgboost for regression and binary classification appears to be weaker for multiclass problems. Interestingly, if we award medals for binary classification based on the F1 score, xgboost only wins 2 gold and 1 bronze medal

Problem Type	Metric	Extractor	Gold	Silver	Bronze
Regression	Exp. Var	KernelPCA-rbf	5	6	7
Regression	Exp. Var	None	3	2	2
Regression	Exp. Var	KernelPCA-poly3	2	3	2
Regression	Exp. Var	KernelPCA-sigmoid	1	0	0
Multiclass Classification	F1 Weighted	KernelPCA-poly3	2	3	2
Multiclass Classification	F1 Weighted	KernelPCA-rbf	2	1	1
Multiclass Classification	F1 Weighted	None	1	1	0
Multiclass Classification	F1 Weighted	KernelPCA-cosine	0	0	1
Multiclass Classification	F1 Weighted	PCA	0	0	1
Binary Classification	Somers' D	None	5	5	3
Binary Classification	Somers' D	KernelPCA-poly3	5	3	1
Binary Classification	Somers' D	KernelPCA-rbf	3	6	7
Binary Classification	Somers' D	KernelPCA-cosine	2	2	2
Binary Classification	Somers' D	PCA	1	0	1
Binary Classification	Somers' D	KernelPCA-sigmoid	0	0	2

Table 5.3: Best extractors medals

(see appendix A for supplementary results). It might therefore be (partly) an effect of the F1 metric. However, if we distribute medals based on the metric accuracy xgboost is the highest ranking model again for binary classification with 3 gold, 1 silver and 2 bronze medals. For multiclass classification it ranks again in the middle of the medal table. Note that these comparisons should be taken with caution as the hyperparameter optimization was performed for the metrics reported in table 5.2.

We also award medals to the best extractors per dataset (assessed on the white box models only), as displayed in table 5.3. We see that, besides using the raw features (None), only the PCA and Kernel-PCA methods won any medals.

As the overall medal tables are all dominated by black box algorithms we also perform a ranking on the white box models only: Table 5.4 shows the ranking of white box algorithms if no additional extractor is used. In table 5.5 the same comparison is shown if all combinations with feature extractors are included. Note that we also list MARS in table 5.4, though we consider it to have the equivalent of a feature extractor build into the algorithm itself. This probably explains why MARS is dominating the table when no feature extraction methods are included but ranks lower when the other white box models are allowed to utilize a feature extraction algorithm.

We see that some algorithms such as elastic net perform strongly on the white box models alone but then disappear in the overall table. This is due to the fact that they perform well on the datasets where the black box algorithms performed even better.

Problem Type	Metric	Model	Gold	Silver	Bronze
Regression	Exp. Var	MARS-Earth	5	1	0
Regression	Exp. Var	KNeighborsRegressor	2	5	3
Regression	Exp. Var	DecisionTreeRegressor	2	1	2
Regression	Exp. Var	Lasso	1	1	2
Regression	Exp. Var	ElasticNet	1	0	0
Regression	Exp. Var	LinearRegression	0	3	2
Regression	Exp. Var	Ridge	0	0	2
Multiclass Classification	F1 Weighted	QDA	2	0	0
Multiclass Classification	F1 Weighted	KNeighborsClassifier	1	1	1
Multiclass Classification	F1 Weighted	DecisionTreeClassifier	1	0	0
Multiclass Classification	F1 Weighted	Logit	1	0	0
Multiclass Classification	F1 Weighted	Logit Lasso	0	2	1
Multiclass Classification	F1 Weighted	LDA	0	1	0
Multiclass Classification	F1 Weighted	NaiveBayesGaussian	0	1	0
Multiclass Classification	F1 Weighted	Logit Ridge	0	0	3
Binary Classification	Somers' D	Logit Ridge	7	3	3
Binary Classification	Somers' D	LinearSVC	5	7	3
Binary Classification	Somers' D	KNeighborsClassifier	3	0	0
Binary Classification	Somers' D	Logit Lasso	1	4	8
Binary Classification	Somers' D	LDA	0	1	1
Binary Classification	Somers' D	DecisionTreeClassifier	0	1	0
Binary Classification	Somers' D	NaiveBayesGaussian	0	0	1

Table 5.4: Medals for white box models without any extractor

Problem Type	Metric	Model	Gold	Silver	Bronze
Regression	Exp. Var	ElasticNet	3	2	4
Regression	Exp. Var	LinearSVR	3	1	0
Regression	Exp. Var	Lasso	2	2	3
Regression	Exp. Var	Ridge	1	4	3
Regression	Exp. Var	KNeighborsRegressor	1	0	1
Regression	Exp. Var	MARS-Earth	1	0	0
Regression	Exp. Var	DecisionTreeRegressor	0	1	0
Regression	Exp. Var	LinearRegression	0	1	0
Multiclass Classification	F1 Weighted	LinearSVC	2	1	0
Multiclass Classification	F1 Weighted	Logit Ridge	1	2	1
Multiclass Classification	F1 Weighted	Logit Lasso	1	1	3
Multiclass Classification	F1 Weighted	QDA	1	0	1
Multiclass Classification	F1 Weighted	LDA	0	1	0
Binary Classification	Somers' D	Logit Ridge	7	7	5
Binary Classification	Somers' D	LinearSVC	5	7	7
Binary Classification	Somers' D	Logit	2	1	2
Binary Classification	Somers' D	Logit Lasso	1	1	2
Binary Classification	Somers' D	KNeighborsClassifier	1	0	0

Table 5.5: Medals for white box models including all combinations with extractors

For multiclass classification the linear SVC does not appear in the table at all if no feature extraction is applied but wins most medals with extractor methods.

5.3 Comparison of Normalized Scores

5.3.1 Normalization of the results

In this section we want to aggregate the exact numeric results and see how large any differences between the algorithms actually are. The results over the different datasets and algorithms have to be normalized in order to make them comparable to each other. For this purpose we process the data with the following three steps:

1. Remove unstable algorithms: Some algorithm and extractor combinations were unstable, meaning that they either yielded results way below the dummy performance or already aborted during training. The combination of kernel-PCA with linear regression was unstable on almost all regression datasets and therefore removed from this analysis[2]. Also the QDA model together with the LLE-modified was unstable on 7 binary classification problems and thus was completely omitted. Some algorithms extractor combinations were only unstable on one dataset. In order not to bias the result, we remove any algorithm for each dataset, where it has a median equal or below -0.5[3]. A list of the concerned entries can be found in table 5.6.

2. Winsorize worst possible performance for regression problems: The metric for the binary classification problems are bound from -1.0 (absolute worst performance by always giving the best observations the worst scores) to +1.0 (perfect prediction). Similarly the weighted F1 score of the multiclass classification ranges from 0.0 to 1.0. Contrary the regression metrics can go from minus infinity to +1.0. Some regression algorithms were only unstable on a small number of folds and therefore not removed in the previous step. In order not to allow any single outlier to influence the total result by a significant amount and to put the comparison between regression and classification on a more equal footing we wanted to restrict the lower bound for the regression metrics to -1.0. For this reason

[2]The linear regression methods with a penalty-term did not suffer from this instability.

[3]Note that this value is not possible for the weighted F1 score, as it is bound at the value zero. However, the data for multiclass classification problems did not show any severe instabilities and thus no algorithm was removed

Problem Type	Model and Extractor	Unstable on dataset
Binary Classification	LDA_KernelPCA-sigmoid	Mushroom
	QDA_KernelPCA-sigmoid	AirScaniaTrucks
	QDA_LLE-Modified	Unstable on 7 datasets. Removed completely
	QDA_None	TitanicKaggle
Regression	LinearRegression_KernelPCA-cosine	Unstable on almost all datasets. Removed completely.
	LinearRegression_KernelPCA-poly3	Unstable on almost all datasets. Removed completely.
	LinearRegression_KernelPCA-rbf	Unstable on almost all datasets. Removed completely.
	LinearRegression_KernelPCA-sigmoid	Unstable on almost all datasets. Removed completely.
	LinearRegression_None	AmesHousing

Table 5.6: List of all algorithms (model, extractor pairs), which had to be removed due to instabilities.

we winsorize the metric for each fold of the regression problems by setting every value which is lower to be equal to -1.0.

3. Normalize for idiosyncratic effects: For each dataset, the difficulty in predicting new values is different. The reason lies in the different magnitude of the idiosyncratic effects on each observation. In order to make the scores for each datasets more comparable the following normalization is performed: For each combination of dataset and metric we determine the 95% percentile of the metric, taken over all algorithms and folds. Each CV-fold of the dataset and metric combination is then normalized by dividing its result through this percentile.

$$\text{NormalizedMetric}_{d,a,c} = \frac{\text{Metric}_{d,a,c}}{Q95_d} \tag{5.1}$$

where d denotes each dataset, a each algorithm extractor pair and c stands for a specific CV-fold. The percentile is taken over all algorithms and all folds, per metric and dataset. After this normalization each algorithm should show a similar result structure over all datasets. We can confirm that this is indeed the case with a simple boxplot (see figure 5.1). Note that this normalization can lead to values which are above +1.0.

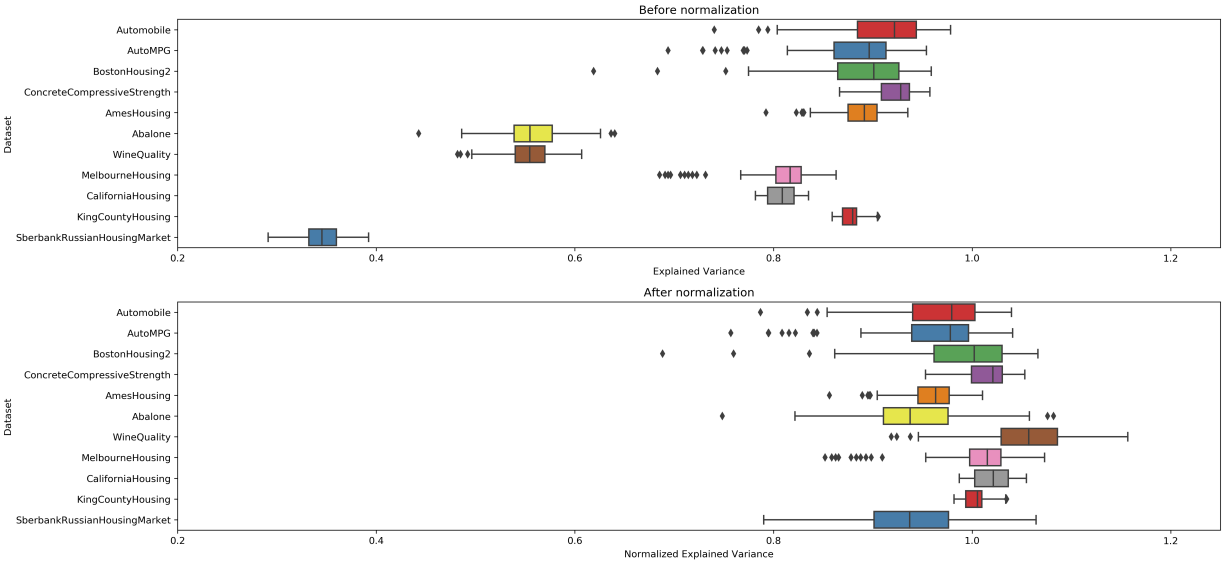

Figure 5.1: Results for the regression random forest before and after normalization for the metric explained variance by dataset. We see that after the normalization the results are of similar range and an aggregation of the results over all datasets is now possible.

Next, we aggregate all the normalized scores over all datasets, calculate the mean scores per algorithm and rank the algorithms accordingly. In order to obtain confidence intervals we bootstrap the normalized scores per algorithm and use the pivotal confidence intervals (Wasserman 2013):

$$(2\hat{\theta} - \hat{\theta}^*_{(1-\alpha/2)}, 2\hat{\theta} - \hat{\theta}^*_{(\alpha/2)}) \tag{5.2}$$

where $\hat{\theta}_{(1-\alpha/2)}$ denotes the $1 - \alpha/2$ percentile of the bootstrapped coefficients $\hat{\theta}^*$. Note that the cross validation results can be correlated (Bengio and Grandvalet 2004), therefore the 95% confidence interval should be interpreted cautiously and taken with a grain of salt.

5.3.2 Discussion of normalized results

The results over all algorithms can be seen in figures 5.2, 5.3 and 5.4. We also show more focused plots in figures 5.5, 5.6 and 5.7, which display only the best white box algorithms[4] with a more confined x-axis. The table on which this plots are based can be found in appendix A together with supplementary information on the performance regarding additional metrics.

[4]As discussed previously, the MARS model is shown in red, as we consider it to have a build-in feature extractor.

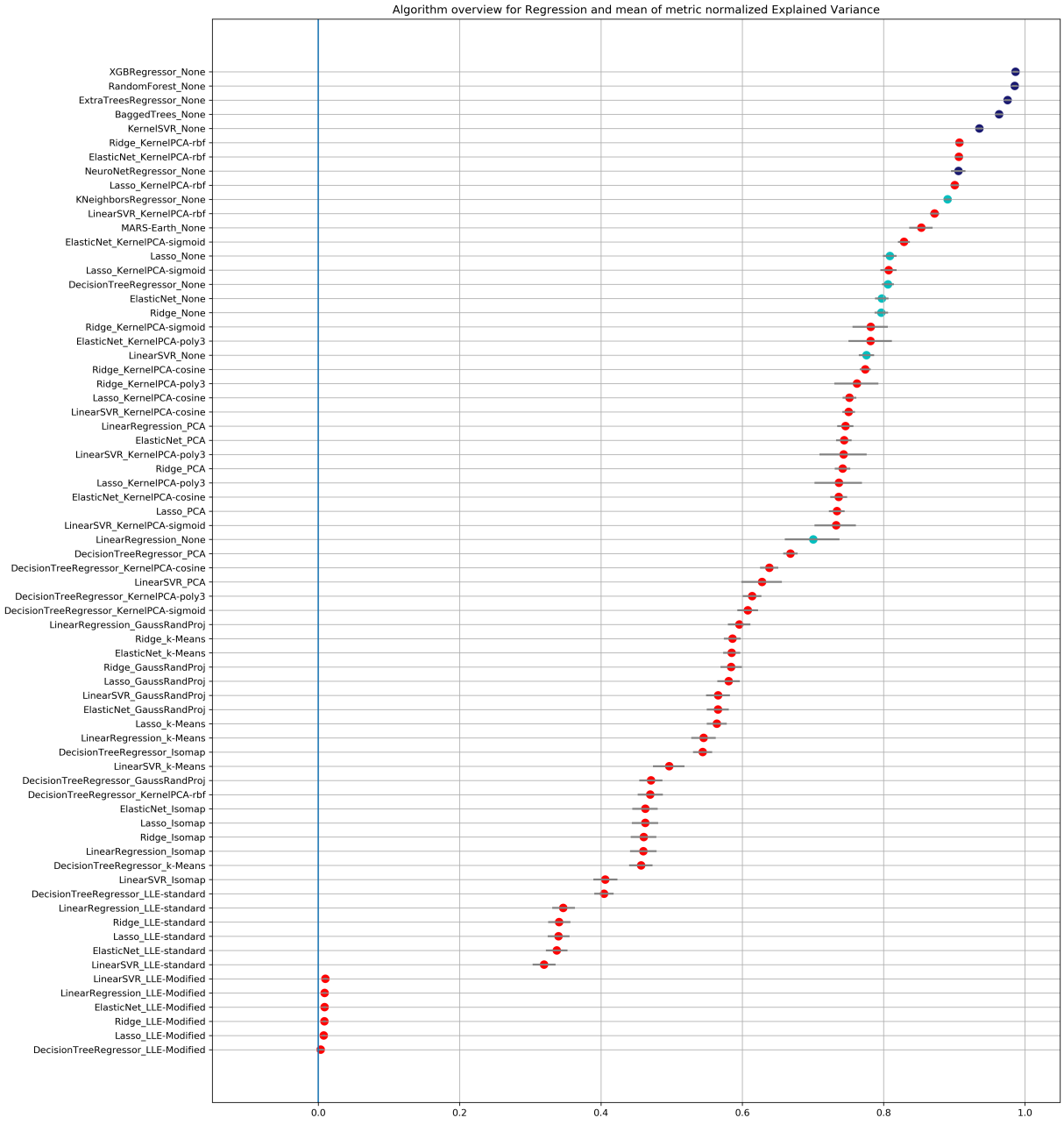

Figure 5.2: Comparison of all algorithms for regression with regards to the mean normalized explained variance. Black box algorithms are displayed in dark blue. White box algorithms are shown in cyan if applied to the raw data, and in red if any extractor is used. Confidence intervals are shown with grey bars.

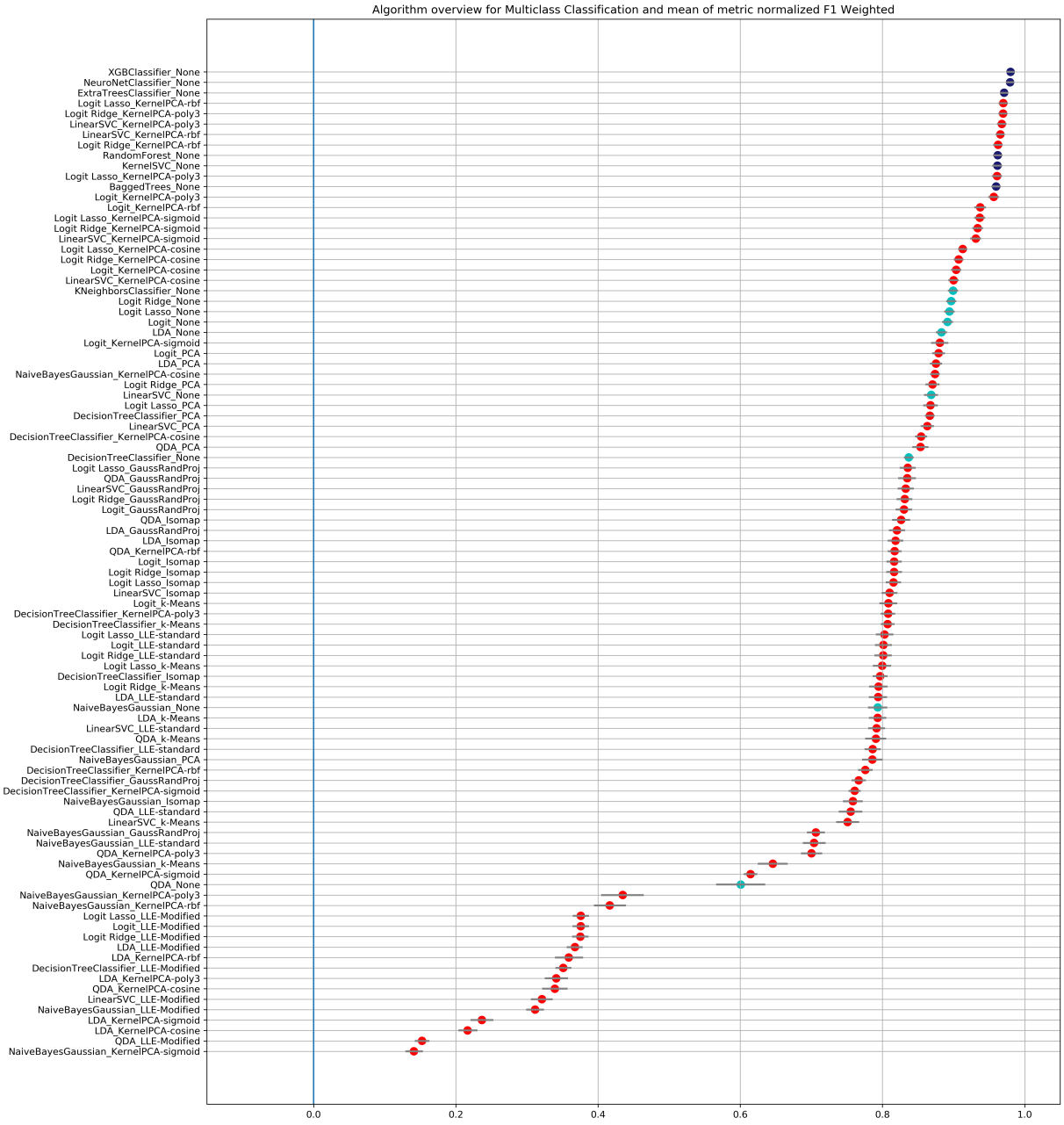

Figure 5.3: Comparison of all algorithms for multiclass classification with regards to the mean normalized weighted F1 score. Black box algorithms are displayed in dark blue. White box algorithms are shown in cyan if applied to the raw data, and in red if any extractor is used. Confidence intervals are shown with grey bars.

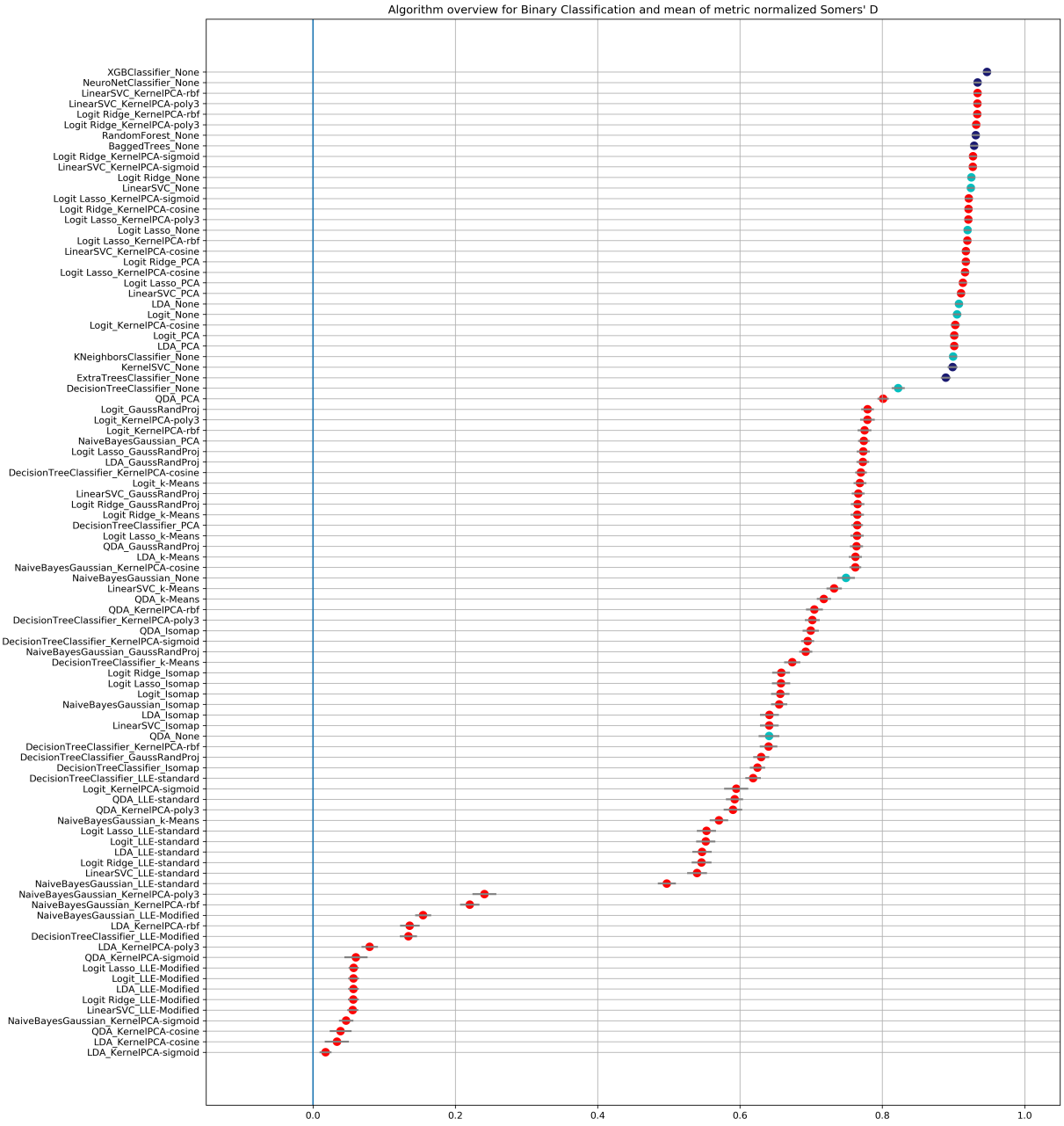

Figure 5.4: Comparison of all algorithms for binary classification with regards to the mean normalized Somers' D value. Black box algorithms are displayed in dark blue. White box algorithms are shown in cyan if applied to the raw data, and in red if any extractor is used. Confidence intervals are shown with grey bars.

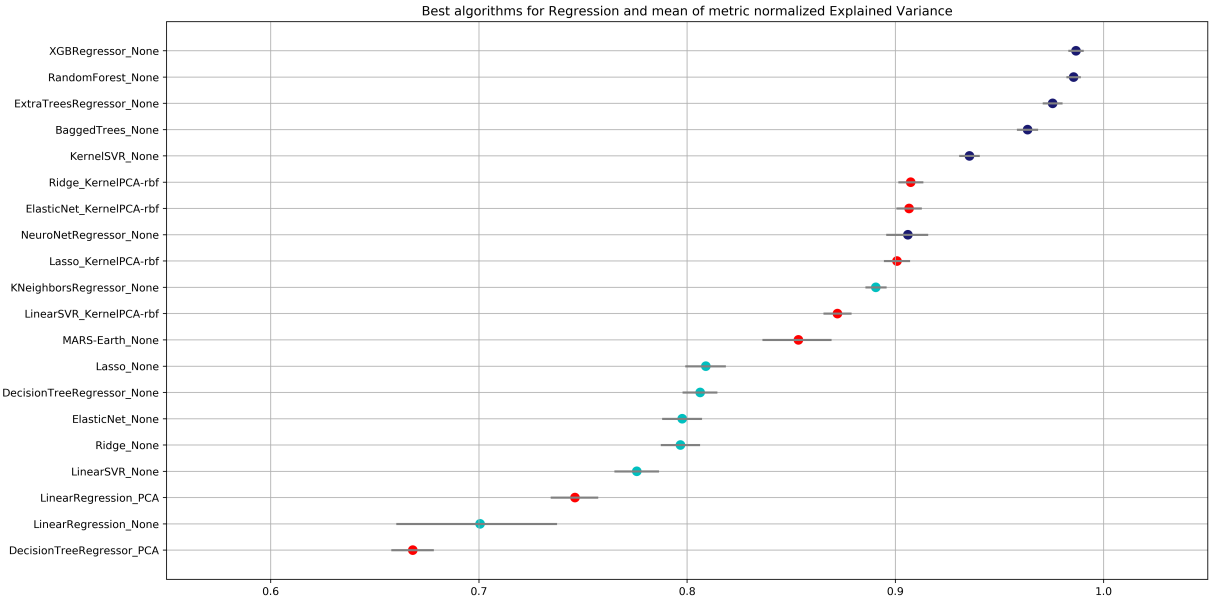

Figure 5.5: Selection of the best regression algorithms with regards to the normalized explained variance. In this plot only the best white box extractor duo is shown together with the results without any extractors. Black box algorithms are displayed in dark blue. White box algorithms are shown in cyan if applied to the raw data, and in red if any extractor is used. Confidence intervals are shown with grey bars.

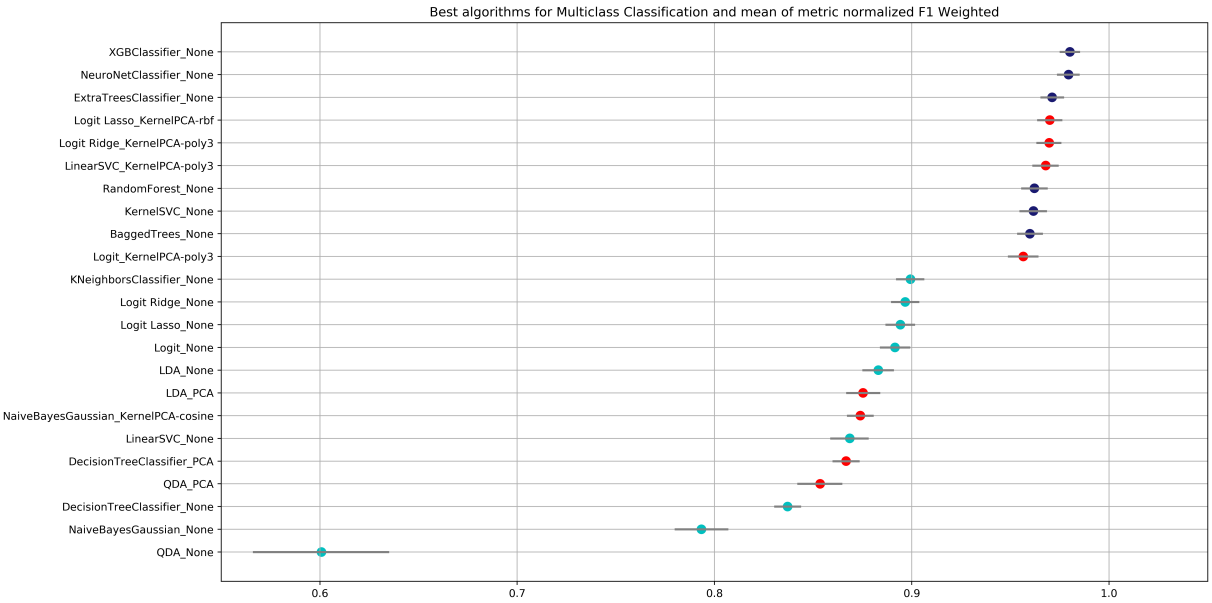

Figure 5.6: Selection of the best multiclass classification algorithms with regards to the normalized weighted F1 score. In this plot only the best white box extractor duo is shown together with the results without any extractors. Black box algorithms are displayed in dark blue. White box algorithms are shown in cyan if applied to the raw data, and in red if any extractor is used. Confidence intervals are shown with grey bars.

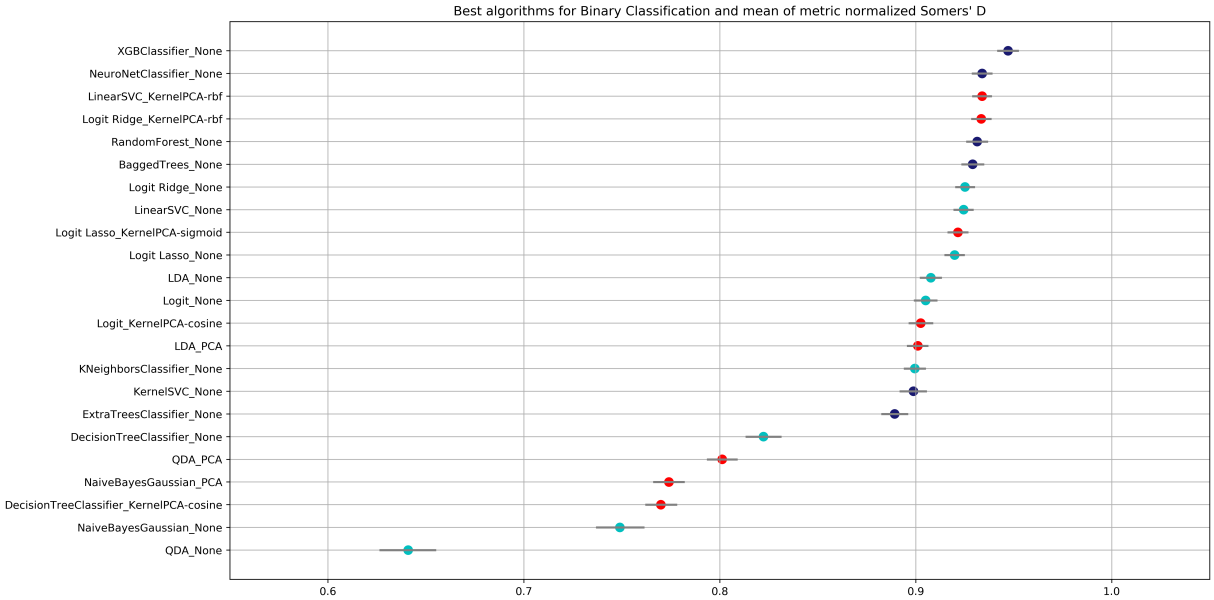

Figure 5.7: Selection of the best binary classification algorithms with regards to the normalized Somers' D value. In this plot only the best white box extractor duo is shown together with the results without any extractors. Black box algorithms are displayed in dark blue. White box algorithms are shown in cyan if applied to the raw data, and in red if any extractor is used. Confidence intervals are shown with grey bars.

For regression problems we see that all black box algorithms, with the exception of the neural network, performed significantly better than the white box methods. The xgboost algorithm was the best performing algorithm. It was able to explain almost 10% more variance than the best white box model ridge regression[5] (with RBF based kernel-PCA as its extractor).

For multiclass classification we see that all white box algorithms without any extractor performed significantly worse and cannot compete with the other methods. While xgboost and the neural network have the highest mean scores they only perform roughly 1% better than the three best white box algorithms logistic regression, logistic lasso, and liner SVC. These three even outperformed the three black box methods random forest, kernel-SVC, and bagged trees.

For binary classification the xgboost algorithm achieved a 1% higher Somers' D than the second best model, which was a linear SVC with kernel-PCA of type

[5]The explained variance was 98.7% for xgboost and 90.7% for ridge, as can be found in appendix A.

RBF as its extractor. The following methods appear to cluster in a similar range (as can be seen in figure 5.7) starting from a normalized Somers' D of 0.93 for the linear SVC up to the extra trees classifier with 0.89. They are comprised of the other black box methods (besides xgboost) and many other white box algorithms, all with extractors. The next best algorithm (a decision tree without any extractor) already shows a significant drop in performance to 0.82.

Overall we see that xgboost was the best algorithm over all problem types with regards to the normalized scores. The neural network performed poorly on the regression problems in comparison to the other black box algorithms. It performed well on the classification tasks, being especially competitive with xgboost for multiclass problems.

The results of the normalized scores again confirm that the PCA based methods were the best performing extractors (for all three problem-types) and able to enhance many of the white box algorithms. The manifold methods performed poorly. The k-means transformation performed somewhere in the mid-range.

We can see that the regularized white box methods outperformed the unregularized linear regression: The linear regression (with a PCA extractor) achieved a normalized explained variance of 0.75, while the ridge regression achieved a score of 0.91. However this appears to be related to the fact that normal linear regression was unstable with regards to the kernel PCA methods, which showed the best performance on the regression datasets. This behaviour was not apparent for the classification problem types where the results were stable and logistic regression performed similarly to the logistic lasso and ridge.

Far better an approximate answer to the right question, which is often vague, than an exact answer to the wrong question, which can always be made precise.

— John Tukey (Tukey 1962)

6 Conclusion

In this study we benchmarked the performance of black box machine learning methods, which are hard to interpret but can approximate non-linear relationships in the data, with more interpretable white box models. These white box models gain their interpretability by making stronger assumptions about the data, like assuming a linear dependency of the variables to the prediction target. Contrary to previous studies we allowed the white box models to use additional feature extraction methods. For this purpose algorithms based on principal component analysis, manifold methods, k-means clustering transformations, and random projections were applied. We belief that these methods, at least partly, resemble and approximate the process of manual feature extraction, performed when developing a white box model. This approximation should be viewed as a potentially underestimated baseline, since these methods would most likely be surpassed by a proper manual feature extraction.

We evaluated the performance of the algorithms for regression and classification problems on 32 real-world datasets. Our assessment of model-performance is based on their ranking per dataset and by means of a normalized score, which attempts to remedy the idiosyncratic effects specific to each dataset. From our empirical results we draw the following six conclusions:

Extractors increase the performance of white box models: For most algorithms their use results in a substantial gain in performance. The increase was

highest for multiclass classification and regression problems. For binary classification problems the gain was smaller, possibly due to the already better performance of the algorithms without extractors.

Principal component analysis, including its kernel variants, were the best feature extractors: For many datasets, especially for binary classification, using the raw features without any extractor produced competitive results for white box models as well. Other methods based on manifold methods, k-means and random projections performed worse. We speculate that the manifold methods might achieve a better extraction on unstructured datasets (texts, images) for which they were originally developed; it appears that on structured data the more straightforward transformations of PCA are more appropriate.

The best black box algorithms: The boosting algorithm xgboost performed best in our analysis. This might explain its ubiquity in machine learning competitions. Generally all black box algorithms were well performing and achieved similar scores. The Neural network performed well on classification tasks, but showed lower performance on regression datasets[1]. The support vector machine showed a lower performance on regression and binary classification problems than most other black box algorithms. The extra trees were in a similarly lower range for the binary classification problems.

The best white box models: The models of linear structure, like linear and logistic regression, performed best. Often the regularized methods (lasso, ridge, elastic-net) performed better than their un-regularized versions. Linear support vector machines performed well on classification problems, but their regression version performed worse. LDA, though structurally similar to logistic regression, was not able to show a similarly high performance on classification tasks. The methods of non-linear structure naive Bayes, QDA, and a single tree were not able to compete. It appears that they were unable to fully utilize the extracted features; sometimes all versions using an extractor performed worse than using the raw features directly.

[1]Potentially this might be due to the chosen network structure and parameters, so no premature general conclusions should be drawn.

White box models with feature extraction were able to compete with black box models on classification problems: While typically black box models showed the best performance the top white box models (including feature extraction) had just marginally lower scores. This was apparent for both types of binary and multiclass classification. Referring back to the question of this thesis, we can conclude that for these types of problems a simple model with proper (ideally manual) feature extraction can compete with a black box model. As a practical suggestion we would recommend to benchmark a manually constructed simple model against black box models. This can provide an indication whether the manual feature extraction is adequate or could be improved.

On regression problems the black box algorithms appeared superior: The white box methods were on average not able to compete with the tree-based black box methods. We speculate that this might be related to the feature extraction and metrics used. Classification problems are evaluated using rank ordering metrics; thus if a feature extraction creates new features monotone with the target, a simple linear method (e.g. logit) will perform well on such metrics. For regression problems, considering their value focused metrics, it is more important that the feature is not just monotone, but linear in dependence to the target. Figure 6.1 shows a sketch of this difference. The PCA methods might have been better at producing new features which are monotone but not linear to the target variable, as they do not consider the target during their construction. This would explain the difference in performance between the classification and regression problems.

However, considering the initial question of this thesis, it will be more difficult to obtain linear instead of just monotone features even if feature extraction is performed manually. When constructing a desired simple regression model we would strongly suggest to benchmark its performance against black box models. Any significant deviation in performance of the manual white box model, compared to the black box models, should be scrutinized and – hopefully – be resolved with a more careful feature extraction.

Potential extensions and improvements of this work are discussed in appendix B.

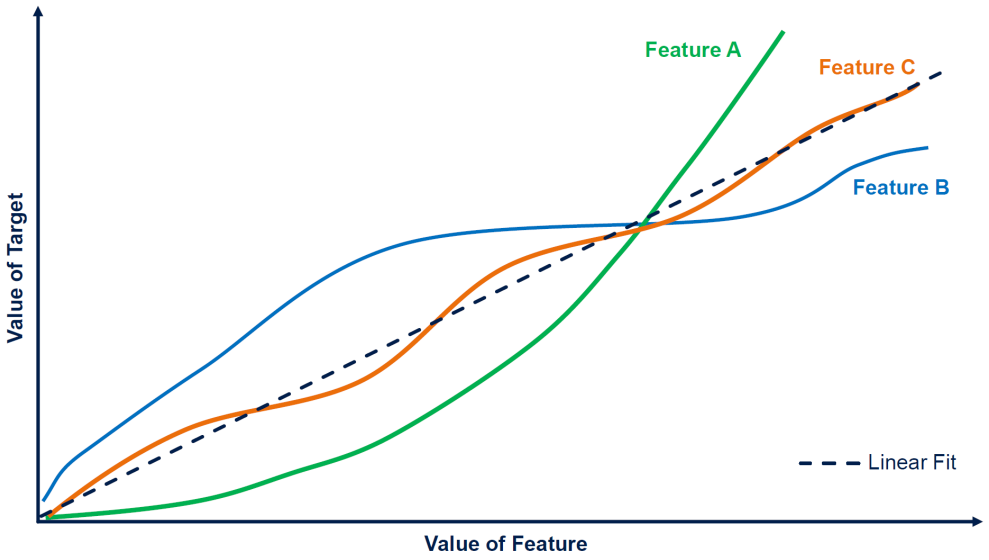

Figure 6.1: This figure shows the difference between features which are merely monotone (A and B) to a rather linear feature (C). We can imagine that these features are coming from one of our feature extraction algorithms. If the algorithm is evaluated on rank metrics the merely monotone features might perform nearly as well as the linear features. Evaluated in terms of squared differences to the target the linear feature will outperform the less linear features. Naturally it will be harder to produce truly linear features compared to extracting covariates which are only monotone.

Appendices

It is a capital mistake to theorize before you have all the evidence. It biases the judgment.

— Sherlock Holmes in *A Study in Scarlet (1887), by Sir Arthur Conan Doyle*

A Supplementary Results

Contents

A.1 Medals for Relative Ranking 57
A.2 Normalized Results 61

Section A.1 will show the medal based relative ranking for supplementary metrics not presented in the main text.

Section A.2 presents a table containing all the normalized scores, as described in section 5.3 of the main text. This includes all metrics from the main text and supplementary metrics. Figures depicting the results for these supplementary metrics are presented as well.

The supplementary metrics are:

- R-squared for regression problems
- F1 score for binary classification
- Accuracy for both multiclass and binary classification

Note that these metrics were not the ones for which the hyperparameter optimization was performed (see 4.3.2). The results of these metrics should thus be viewed with caution, in context of the actual main metric used to determine the hyperparameters.

A.1 Medals for Relative Ranking

Problem Type	Metric	Model	BlackBox	Gold	Silver	Bronze
Regression	R-squared	XGBRegressor	y	4	2	0
Regression	R-squared	ExtraTreesRegressor	y	4	0	1
Regression	R-squared	KernelSVR	y	2	0	1
Regression	R-squared	Ridge	n	1	1	0
Regression	R-squared	RandomForest	y	0	7	1
Regression	R-squared	Lasso	n	0	1	0
Regression	R-squared	BaggedTrees	y	0	0	4
Regression	R-squared	ElasticNet	n	0	0	2
Regression	R-squared	LinearSVR	n	0	0	2
Multiclass Classif.	Accuracy	RandomForest	y	1	0	2
Multiclass Classif.	Accuracy	LinearSVC	n	1	0	1
Multiclass Classif.	Accuracy	NeuroNetClassifier	y	1	0	0
Multiclass Classif.	Accuracy	QDA	n	1	0	0
Multiclass Classif.	Accuracy	XGBClassifier	y	1	0	0
Multiclass Classif.	Accuracy	ExtraTreesClassifier	y	0	1	1
Multiclass Classif.	Accuracy	BaggedTrees	y	0	1	0
Multiclass Classif.	Accuracy	KernelSVC	y	0	1	0
Multiclass Classif.	Accuracy	LDA	n	0	1	0
Multiclass Classif.	Accuracy	Logit Lasso	n	0	1	0
Multiclass Classif.	Accuracy	Logit Ridge	n	0	0	1
Binary Classif.	Accuracy	XGBClassifier	y	3	1	2
Binary Classif.	Accuracy	ExtraTreesClassifier	y	3	0	0
Binary Classif.	Accuracy	Logit Ridge	n	2	4	2
Binary Classif.	Accuracy	RandomForest	y	2	3	3
Binary Classif.	Accuracy	BaggedTrees	y	2	3	0
Binary Classif.	Accuracy	LinearSVC	n	1	2	7
Binary Classif.	Accuracy	NeuroNetClassifier	y	1	1	1
Binary Classif.	Accuracy	KNeighborsClassifier	n	1	0	0
Binary Classif.	Accuracy	Logit	n	1	0	0
Binary Classif.	Accuracy	Logit Lasso	n	0	1	1
Binary Classif.	Accuracy	DecisionTreeClassifier	n	0	1	0
Binary Classif.	F1 Score	ExtraTreesClassifier	y	3	1	0
Binary Classif.	F1 Score	NaiveBayesGaussian	n	2	1	1
Binary Classif.	F1 Score	BaggedTrees	y	2	0	6
Binary Classif.	F1 Score	XGBClassifier	y	2	0	1
Binary Classif.	F1 Score	DecisionTreeClassifier	n	2	0	0
Binary Classif.	F1 Score	RandomForest	y	1	7	1
Binary Classif.	F1 Score	Logit Ridge	n	1	3	1
Binary Classif.	F1 Score	LinearSVC	n	1	1	4
Binary Classif.	F1 Score	QDA	n	1	1	1
Binary Classif.	F1 Score	NeuroNetClassifier	y	1	1	0
Binary Classif.	F1 Score	Logit Lasso	n	0	1	0
Binary Classif.	F1 Score	LDA	n	0	0	1

Table A.1: Overall medals for supplementary metrics

Problem Type	Metric	Extractor	Gold	Silver	Bronze
Regression	R-squared	KernelPCA-rbf	5	6	6
Regression	R-squared	None	3	2	3
Regression	R-squared	KernelPCA-poly3	2	3	2
Regression	R-squared	KernelPCA-sigmoid	1	0	0
Multiclass Classification	Accuracy	KernelPCA-poly3	2	2	4
Multiclass Classification	Accuracy	KernelPCA-rbf	2	2	0
Multiclass Classification	Accuracy	None	1	1	0
Multiclass Classification	Accuracy	PCA	0	0	1
Binary Classification	Accuracy	None	5	3	2
Binary Classification	Accuracy	KernelPCA-poly3	3	2	4
Binary Classification	Accuracy	KernelPCA-rbf	2	6	4
Binary Classification	Accuracy	KernelPCA-cosine	2	1	3
Binary Classification	Accuracy	PCA	2	1	1
Binary Classification	Accuracy	KernelPCA-sigmoid	1	3	2
Binary Classification	Accuracy	LLE-Modified	1	0	0
Binary Classification	F1 Score	None	5	2	3
Binary Classification	F1 Score	KernelPCA-rbf	3	3	2
Binary Classification	F1 Score	KernelPCA-sigmoid	3	2	0
Binary Classification	F1 Score	Isomap	2	4	0
Binary Classification	F1 Score	KernelPCA-poly3	1	2	5
Binary Classification	F1 Score	LLE-Modified	1	1	1
Binary Classification	F1 Score	GaussRandProj	1	0	0
Binary Classification	F1 Score	PCA	0	1	4
Binary Classification	F1 Score	KernelPCA-cosine	0	1	1

Table A.2: Best extractors medals for supplementary metrics

Problem Type	Metric	Model	Gold	Silver	Bronze
Regression	R-squared	MARS-Earth	5	1	0
Regression	R-squared	KNeighborsRegressor	2	4	3
Regression	R-squared	DecisionTreeRegressor	2	1	2
Regression	R-squared	LinearRegression	1	3	1
Regression	R-squared	ElasticNet	1	0	1
Regression	R-squared	Lasso	0	2	2
Regression	R-squared	Ridge	0	0	2
Multiclass Classification	Accuracy	QDA	2	0	0
Multiclass Classification	Accuracy	Logit Lasso	1	1	1
Multiclass Classification	Accuracy	KNeighborsClassifier	1	0	1
Multiclass Classification	Accuracy	Logit	1	0	1
Multiclass Classification	Accuracy	Logit Ridge	0	1	2
Multiclass Classification	Accuracy	DecisionTreeClassifier	0	1	0
Multiclass Classification	Accuracy	LDA	0	1	0
Multiclass Classification	Accuracy	NaiveBayesGaussian	0	1	0
Binary Classification	Accuracy	LinearSVC	4	5	3
Binary Classification	Accuracy	Logit Ridge	4	4	3
Binary Classification	Accuracy	Logit	2	3	1
Binary Classification	Accuracy	Logit Lasso	2	2	5
Binary Classification	Accuracy	KNeighborsClassifier	2	1	0
Binary Classification	Accuracy	DecisionTreeClassifier	2	0	1
Binary Classification	Accuracy	LDA	0	1	2
Binary Classification	Accuracy	QDA	0	0	1
Binary Classification	F1 Score	LinearSVC	4	2	2
Binary Classification	F1 Score	Logit	3	3	2
Binary Classification	F1 Score	Logit Ridge	3	2	4
Binary Classification	F1 Score	NaiveBayesGaussian	3	1	0
Binary Classification	F1 Score	Logit Lasso	2	3	4
Binary Classification	F1 Score	DecisionTreeClassifier	1	2	0
Binary Classification	F1 Score	LDA	0	2	3
Binary Classification	F1 Score	QDA	0	1	1

Table A.3: Medals for white box models without any extractor for supplementary metrics

Problem Type	Metric	Model	Gold	Silver	Bronze
Regression	R-squared	ElasticNet	4	1	4
Regression	R-squared	Ridge	2	4	2
Regression	R-squared	Lasso	1	3	3
Regression	R-squared	LinearSVR	1	2	0
Regression	R-squared	KNeighborsRegressor	1	0	1
Regression	R-squared	MARS-Earth	1	0	1
Regression	R-squared	LinearRegression	1	0	0
Regression	R-squared	DecisionTreeRegressor	0	1	0
Multiclass Classification	Accuracy	LinearSVC	3	1	0
Multiclass Classification	Accuracy	Logit Lasso	1	1	1
Multiclass Classification	Accuracy	QDA	1	0	1
Multiclass Classification	Accuracy	Logit Ridge	0	2	3
Multiclass Classification	Accuracy	LDA	0	1	0
Binary Classification	Accuracy	LinearSVC	6	6	5
Binary Classification	Accuracy	Logit Ridge	4	9	5
Binary Classification	Accuracy	Logit	2	1	1
Binary Classification	Accuracy	Logit Lasso	2	0	5
Binary Classification	Accuracy	DecisionTreeClassifier	1	0	0
Binary Classification	Accuracy	KNeighborsClassifier	1	0	0
Binary Classification	F1 Score	LinearSVC	5	3	6
Binary Classification	F1 Score	Logit Ridge	3	7	2
Binary Classification	F1 Score	Logit Lasso	2	2	2
Binary Classification	F1 Score	NaiveBayesGaussian	2	1	1
Binary Classification	F1 Score	DecisionTreeClassifier	2	0	0
Binary Classification	F1 Score	QDA	1	2	2
Binary Classification	F1 Score	Logit	1	1	2
Binary Classification	F1 Score	LDA	0	0	1

Table A.4: Medals for white box models including all combinations with extractors for supplementary metrics

A.2 Normalized Results

Problem Type	Metric	Model	Blackbox	Extractor	Mean	Mean CI 0.025	Mean CI 0.975	Median
Regression	R-squared	XGBRegressor	y	None	0.98838	0.98467	0.99208	1.00161
Regression	R-squared	RandomForest	y	None	0.98697	0.98340	0.99048	0.99677
Regression	R-squared	ExtraTreesRegressor	y	None	0.97688	0.97220	0.98153	0.99095
Regression	R-squared	BaggedTrees	y	None	0.96465	0.95964	0.96966	0.99059
Regression	R-squared	KernelSVR	y	None	0.92559	0.92047	0.93081	0.94983
Regression	R-squared	NeuroNetRegressor	y	None	0.89132	0.87481	0.91002	0.94443
Regression	R-squared	KNeighborsRegressor	n	None	0.88696	0.88182	0.89227	0.90402
Regression	R-squared	MARS-Earth	n	None	0.85284	0.83683	0.87034	0.92250
Regression	R-squared	Ridge	n	KernelPCA-rbf	0.83152	0.81294	0.85142	0.94341
Regression	R-squared	ElasticNet	n	KernelPCA-rbf	0.83091	0.81246	0.85041	0.94180
Regression	R-squared	ElasticNet	n	KernelPCA-sigmoid	0.82634	0.81774	0.83531	0.86116
Regression	R-squared	Lasso	n	KernelPCA-rbf	0.82419	0.80572	0.84375	0.94120
Regression	R-squared	Lasso	n	None	0.80729	0.79743	0.81746	0.84591
Regression	R-squared	DecisionTreeRegressor	n	None	0.80582	0.79775	0.81419	0.84111
Regression	R-squared	Lasso	n	KernelPCA-sigmoid	0.80337	0.79199	0.81518	0.85415
Regression	R-squared	ElasticNet	n	None	0.79462	0.78531	0.80449	0.82569
Regression	R-squared	Ridge	n	None	0.79381	0.78408	0.80362	0.82698
Regression	R-squared	LinearSVR	n	KernelPCA-rbf	0.78169	0.76323	0.80076	0.89063
Regression	R-squared	ElasticNet	n	KernelPCA-poly3	0.78126	0.75107	0.81403	0.91806
Regression	R-squared	Ridge	n	KernelPCA-sigmoid	0.77884	0.75558	0.80535	0.85861
Regression	R-squared	Ridge	n	KernelPCA-cosine	0.76901	0.76060	0.77747	0.79491
Regression	R-squared	Ridge	n	KernelPCA-poly3	0.76166	0.73025	0.79436	0.90897
Regression	R-squared	LinearSVR	n	None	0.75389	0.74221	0.76570	0.80048
Regression	R-squared	Lasso	n	KernelPCA-cosine	0.74460	0.73415	0.75522	0.79268
Regression	R-squared	LinearRegression	n	PCA	0.74440	0.73348	0.75607	0.76951
Regression	R-squared	ElasticNet	n	PCA	0.74155	0.73044	0.75320	0.76921
Regression	R-squared	Ridge	n	PCA	0.73909	0.72809	0.75067	0.76550
Regression	R-squared	Lasso	n	KernelPCA-poly3	0.73615	0.70300	0.77053	0.90557
Regression	R-squared	LinearSVR	n	KernelPCA-poly3	0.73181	0.69933	0.76602	0.89231
Regression	R-squared	ElasticNet	n	KernelPCA-cosine	0.72985	0.71747	0.74218	0.79322
Regression	R-squared	Lasso	n	PCA	0.72969	0.71823	0.74120	0.75001
Regression	R-squared	LinearSVR	n	KernelPCA-cosine	0.72618	0.71581	0.73679	0.76715
Regression	R-squared	LinearSVR	n	KernelPCA-sigmoid	0.71584	0.68658	0.74709	0.82949
Regression	R-squared	LinearRegression	n	None	0.69854	0.66082	0.73903	0.84235
Regression	R-squared	DecisionTreeRegressor	n	PCA	0.66306	0.65226	0.67417	0.69405
Regression	R-squared	DecisionTreeRegressor	n	KernelPCA-cosine	0.63339	0.61995	0.64756	0.70446
Regression	R-squared	DecisionTreeRegressor	n	KernelPCA-poly3	0.60670	0.59290	0.62090	0.66079
Regression	R-squared	DecisionTreeRegressor	n	KernelPCA-sigmoid	0.60329	0.58817	0.61888	0.66769
Regression	R-squared	LinearSVR	n	PCA	0.59345	0.56421	0.62350	0.71563
Regression	R-squared	LinearRegression	n	GaussRandProj	0.59068	0.57440	0.60699	0.66897
Regression	R-squared	Ridge	n	GaussRandProj	0.57668	0.56117	0.59297	0.65918
Regression	R-squared	Lasso	n	GaussRandProj	0.57324	0.55713	0.58987	0.64968
Regression	R-squared	ElasticNet	n	GaussRandProj	0.55698	0.54113	0.57365	0.62166
Regression	R-squared	DecisionTreeRegressor	n	Isomap	0.53834	0.52477	0.55235	0.60194
Regression	R-squared	LinearRegression	n	k-Means	0.50325	0.48052	0.52702	0.63040
Regression	R-squared	Lasso	n	k-Means	0.49602	0.47314	0.52023	0.62599
Regression	R-squared	ElasticNet	n	k-Means	0.47638	0.45087	0.50319	0.62955
Regression	R-squared	Ridge	n	k-Means	0.46643	0.43988	0.49414	0.62983
Regression	R-squared	DecisionTreeRegressor	n	GaussRandProj	0.46204	0.44473	0.47937	0.51532
Regression	R-squared	LinearSVR	n	GaussRandProj	0.45861	0.43171	0.48644	0.60832
Regression	R-squared	ElasticNet	n	Isomap	0.45486	0.43674	0.47302	0.46106
Regression	R-squared	Lasso	n	Isomap	0.45485	0.43657	0.47352	0.45753
Regression	R-squared	LinearRegression	n	Isomap	0.45308	0.43390	0.47244	0.45760
Regression	R-squared	Ridge	n	Isomap	0.45296	0.43453	0.47154	0.45844
Regression	R-squared	DecisionTreeRegressor	n	k-Means	0.41056	0.38746	0.43470	0.54294
Regression	R-squared	DecisionTreeRegressor	n	KernelPCA-rbf	0.40455	0.37842	0.43135	0.49072
Regression	R-squared	DecisionTreeRegressor	n	LLE-standard	0.39762	0.38327	0.41201	0.39364
Regression	R-squared	LinearRegression	n	LLE-standard	0.33967	0.32336	0.35570	0.24551
Regression	R-squared	Ridge	n	LLE-standard	0.33336	0.31804	0.34862	0.23361
Regression	R-squared	Lasso	n	LLE-standard	0.33229	0.31699	0.34767	0.25126
Regression	R-squared	ElasticNet	n	LLE-standard	0.32975	0.31467	0.34490	0.24321
Regression	R-squared	LinearSVR	n	Isomap	0.13411	0.09697	0.17215	0.31982
Regression	R-squared	LinearSVR	n	k-Means	0.03770	-0.01212	0.08855	0.42894
Regression	R-squared	LinearSVR	n	LLE-standard	-0.00736	-0.04917	0.03688	0.13242
Regression	R-squared	LinearRegression	n	LLE-Modified	-0.00869	-0.01258	-0.00444	-0.00146
Regression	R-squared	ElasticNet	n	LLE-Modified	-0.00873	-0.01231	-0.00476	-0.00083
Regression	R-squared	Ridge	n	LLE-Modified	-0.00897	-0.01265	-0.00486	-0.00085
Regression	R-squared	Lasso	n	LLE-Modified	-0.00992	-0.01359	-0.00584	-0.00126
Regression	R-squared	DecisionTreeRegressor	n	LLE-Modified	-0.01449	-0.02035	-0.00827	-0.00182
Regression	R-squared	LinearSVR	n	LLE-Modified	-0.63165	-0.67575	-0.58708	-0.12299
Regression	Exp. Var.	XGBRegressor	y	None	0.98675	0.98310	0.99043	0.99874
Regression	Exp. Var.	RandomForest	y	None	0.98561	0.98215	0.98913	0.99639
Regression	Exp. Var.	ExtraTreesRegressor	y	None	0.97553	0.97081	0.98029	0.99126
Regression	Exp. Var.	BaggedTrees	y	None	0.96350	0.95846	0.96868	0.98901
Regression	Exp. Var.	KernelSVR	y	None	0.93559	0.93071	0.94057	0.95420
Regression	Exp. Var.	Ridge	n	KernelPCA-rbf	0.90737	0.90128	0.91332	0.94033
Regression	Exp. Var.	ElasticNet	n	KernelPCA-rbf	0.90659	0.90053	0.91270	0.93881
Regression	Exp. Var.	NeuroNetRegressor	y	None	0.90601	0.89627	0.91648	0.94480
Regression	Exp. Var.	Lasso	n	KernelPCA-rbf	0.90078	0.89452	0.90709	0.94022
Regression	Exp. Var.	KNeighborsRegressor	n	None	0.89061	0.88550	0.89565	0.90525

Continued on next page

Problem Type	Metric	Model	Blackbox	Extractor	Mean	Mean CI 0.025	Mean CI 0.975	Median
Regression	Exp. Var.	LinearSVR	n	KernelPCA-rbf	0.87219	0.86547	0.87895	0.89785
Regression	Exp. Var.	MARS-Earth	n	None	0.85344	0.83759	0.87072	0.91994
Regression	Exp. Var.	ElasticNet	n	KernelPCA-sigmoid	0.82898	0.82064	0.83752	0.86097
Regression	Exp. Var.	Lasso	n	None	0.80898	0.79941	0.81890	0.84818
Regression	Exp. Var.	Lasso	n	KernelPCA-sigmoid	0.80725	0.79605	0.81895	0.85445
Regression	Exp. Var.	DecisionTreeRegressor	n	None	0.80626	0.79800	0.81477	0.84081
Regression	Exp. Var.	ElasticNet	n	None	0.79763	0.78817	0.80731	0.82941
Regression	Exp. Var.	Ridge	n	None	0.79679	0.78737	0.80629	0.83079
Regression	Exp. Var.	Ridge	n	KernelPCA-sigmoid	0.78188	0.75788	0.80769	0.86115
Regression	Exp. Var.	ElasticNet	n	KernelPCA-poly3	0.78160	0.75131	0.81301	0.91831
Regression	Exp. Var.	LinearSVR	n	None	0.77582	0.76513	0.78660	0.82116
Regression	Exp. Var.	Ridge	n	KernelPCA-cosine	0.77407	0.76618	0.78195	0.79459
Regression	Exp. Var.	Ridge	n	KernelPCA-poly3	0.76237	0.73210	0.79449	0.90629
Regression	Exp. Var.	Lasso	n	KernelPCA-cosine	0.75171	0.74209	0.76148	0.79248
Regression	Exp. Var.	LinearSVR	n	KernelPCA-cosine	0.75062	0.74181	0.75963	0.78340
Regression	Exp. Var.	LinearRegression	n	PCA	0.74613	0.73497	0.75785	0.76803
Regression	Exp. Var.	ElasticNet	n	PCA	0.74425	0.73358	0.75582	0.77106
Regression	Exp. Var.	LinearSVR	n	KernelPCA-poly3	0.74345	0.71077	0.77773	0.90467
Regression	Exp. Var.	Ridge	n	PCA	0.74214	0.73142	0.75336	0.76735
Regression	Exp. Var.	Lasso	n	KernelPCA-poly3	0.73687	0.70431	0.77143	0.90633
Regression	Exp. Var.	ElasticNet	n	KernelPCA-cosine	0.73656	0.72480	0.74864	0.79440
Regression	Exp. Var.	Lasso	n	PCA	0.73409	0.72321	0.74569	0.75428
Regression	Exp. Var.	LinearSVR	n	KernelPCA-sigmoid	0.73290	0.70506	0.76372	0.84092
Regression	Exp. Var.	LinearRegression	n	None	0.70056	0.66365	0.74086	0.84361
Regression	Exp. Var.	DecisionTreeRegressor	n	PCA	0.66824	0.65819	0.67860	0.69429
Regression	Exp. Var.	DecisionTreeRegressor	n	KernelPCA-cosine	0.63841	0.62578	0.65163	0.70414
Regression	Exp. Var.	LinearSVR	n	PCA	0.62804	0.59994	0.65721	0.74486
Regression	Exp. Var.	DecisionTreeRegressor	n	KernelPCA-poly3	0.61405	0.60108	0.62747	0.66448
Regression	Exp. Var.	DecisionTreeRegressor	n	KernelPCA-sigmoid	0.60788	0.59335	0.62284	0.67295
Regression	Exp. Var.	LinearRegression	n	GaussRandProj	0.59590	0.58016	0.61212	0.66919
Regression	Exp. Var.	Ridge	n	k-Means	0.58624	0.57453	0.59857	0.63492
Regression	Exp. Var.	ElasticNet	n	k-Means	0.58505	0.57303	0.59715	0.63425
Regression	Exp. Var.	Ridge	n	GaussRandProj	0.58430	0.56910	0.59958	0.66735
Regression	Exp. Var.	Lasso	n	GaussRandProj	0.58080	0.56522	0.59687	0.65512
Regression	Exp. Var.	LinearSVR	n	GaussRandProj	0.56588	0.54928	0.58313	0.64603
Regression	Exp. Var.	ElasticNet	n	GaussRandProj	0.56568	0.55045	0.58177	0.62799
Regression	Exp. Var.	Lasso	n	k-Means	0.56407	0.55022	0.57829	0.63022
Regression	Exp. Var.	LinearRegression	n	k-Means	0.54536	0.52819	0.56304	0.63279
Regression	Exp. Var.	DecisionTreeRegressor	n	Isomap	0.54403	0.53067	0.55769	0.60579
Regression	Exp. Var.	LinearSVR	n	k-Means	0.49666	0.47500	0.51963	0.60076
Regression	Exp. Var.	DecisionTreeRegressor	n	GaussRandProj	0.47090	0.45480	0.48761	0.52106
Regression	Exp. Var.	DecisionTreeRegressor	n	KernelPCA-rbf	0.46976	0.45198	0.48755	0.49328
Regression	Exp. Var.	ElasticNet	n	Isomap	0.46285	0.44523	0.48137	0.46697
Regression	Exp. Var.	Lasso	n	Isomap	0.46279	0.44472	0.48172	0.46152
Regression	Exp. Var.	Ridge	n	Isomap	0.46064	0.44276	0.47909	0.46323
Regression	Exp. Var.	LinearRegression	n	Isomap	0.46007	0.44147	0.47895	0.46089
Regression	Exp. Var.	DecisionTreeRegressor	n	k-Means	0.45689	0.44080	0.47388	0.54727
Regression	Exp. Var.	LinearSVR	n	Isomap	0.40612	0.38896	0.42308	0.39793
Regression	Exp. Var.	DecisionTreeRegressor	n	LLE-standard	0.40445	0.39082	0.41828	0.40679
Regression	Exp. Var.	LinearRegression	n	LLE-standard	0.34666	0.33025	0.36229	0.26056
Regression	Exp. Var.	Ridge	n	LLE-standard	0.34081	0.32491	0.35637	0.25757
Regression	Exp. Var.	Lasso	n	LLE-standard	0.33997	0.32448	0.35540	0.26267
Regression	Exp. Var.	ElasticNet	n	LLE-standard	0.33754	0.32247	0.35292	0.26355
Regression	Exp. Var.	LinearSVR	n	LLE-standard	0.31953	0.30310	0.33557	0.21251
Regression	Exp. Var.	LinearSVR	n	LLE-Modified	0.01005	0.00718	0.01278	0.00000
Regression	Exp. Var.	LinearRegression	n	LLE-Modified	0.00898	0.00657	0.01131	0.00040
Regression	Exp. Var.	ElasticNet	n	LLE-Modified	0.00892	0.00682	0.01091	0.00035
Regression	Exp. Var.	Ridge	n	LLE-Modified	0.00868	0.00659	0.01074	0.00035
Regression	Exp. Var.	Lasso	n	LLE-Modified	0.00774	0.00563	0.00971	0.00000
Regression	Exp. Var.	DecisionTreeRegressor	n	LLE-Modified	0.00343	-0.00108	0.00855	0.00000
Multiclass Classif.	F1 Weighted	XGBClassifier	y	None	0.98016	0.97503	0.98545	0.99320
Multiclass Classif.	F1 Weighted	NeuroNetClassifier	y	None	0.97948	0.97393	0.98536	1.00000
Multiclass Classif.	F1 Weighted	ExtraTreesClassifier	y	None	0.97113	0.96511	0.97711	0.98871
Multiclass Classif.	F1 Weighted	Logit Lasso	n	KernelPCA-rbf	0.96996	0.96360	0.97645	0.99786
Multiclass Classif.	F1 Weighted	Logit Ridge	n	KernelPCA-poly3	0.96967	0.96352	0.97626	0.99968
Multiclass Classif.	F1 Weighted	LinearSVC	n	KernelPCA-poly3	0.96789	0.96138	0.97473	0.99735
Multiclass Classif.	F1 Weighted	LinearSVC	n	KernelPCA-rbf	0.96583	0.95978	0.97238	0.99397
Multiclass Classif.	F1 Weighted	Logit Ridge	n	KernelPCA-rbf	0.96263	0.95625	0.96935	0.99417
Multiclass Classif.	F1 Weighted	RandomForest	y	None	0.96216	0.95545	0.96890	0.98211
Multiclass Classif.	F1 Weighted	KernelSVC	y	None	0.96168	0.95489	0.96884	0.99424
Multiclass Classif.	F1 Weighted	Logit Lasso	n	KernelPCA-poly3	0.96116	0.95431	0.96841	0.99241
Multiclass Classif.	F1 Weighted	BaggedTrees	y	None	0.95988	0.95331	0.96643	0.97607
Multiclass Classif.	F1 Weighted	Logit	n	KernelPCA-poly3	0.95653	0.94891	0.96439	0.99389
Multiclass Classif.	F1 Weighted	Logit	n	KernelPCA-rbf	0.93754	0.92961	0.94587	0.96500
Multiclass Classif.	F1 Weighted	Logit Lasso	n	KernelPCA-sigmoid	0.93690	0.92957	0.94470	0.95742
Multiclass Classif.	F1 Weighted	Logit Ridge	n	KernelPCA-sigmoid	0.93388	0.92634	0.94151	0.96258
Multiclass Classif.	F1 Weighted	LinearSVC	n	KernelPCA-sigmoid	0.93133	0.92394	0.93908	0.95577
Multiclass Classif.	F1 Weighted	Logit Lasso	n	KernelPCA-cosine	0.91290	0.90650	0.91950	0.92027
Multiclass Classif.	F1 Weighted	Logit Ridge	n	KernelPCA-cosine	0.90723	0.90046	0.91417	0.91835
Multiclass Classif.	F1 Weighted	Logit	n	KernelPCA-cosine	0.90369	0.89651	0.91100	0.91835
Multiclass Classif.	F1 Weighted	LinearSVC	n	KernelPCA-cosine	0.90006	0.89297	0.90733	0.90844
Multiclass Classif.	F1 Weighted	KNeighborsClassifier	n	None	0.89934	0.89233	0.90671	0.89480
Multiclass Classif.	F1 Weighted	Logit Ridge	n	None	0.89669	0.88955	0.90397	0.90849
Multiclass Classif.	F1 Weighted	Logit Lasso	n	None	0.89424	0.88680	0.90186	0.90812
Multiclass Classif.	F1 Weighted	Logit	n	None	0.89149	0.88374	0.89923	0.90762

Continued on next page

Problem Type	Metric	Model	Blackbox	Extractor	Mean	Mean CI 0.025	Mean CI 0.975	Median
Multiclass Classif.	F1 Weighted	LDA	n	None	0.88304	0.87516	0.89116	0.89438
Multiclass Classif.	F1 Weighted	Logit	n	KernelPCA-sigmoid	0.88073	0.86865	0.89314	0.90636
Multiclass Classif.	F1 Weighted	Logit	n	PCA	0.87897	0.87011	0.88815	0.90179
Multiclass Classif.	F1 Weighted	LDA	n	PCA	0.87531	0.86665	0.88388	0.88501
Multiclass Classif.	F1 Weighted	NaiveBayesGaussian	n	KernelPCA-cosine	0.87393	0.86719	0.88075	0.86260
Multiclass Classif.	F1 Weighted	Logit Ridge	n	PCA	0.87044	0.86061	0.88063	0.89579
Multiclass Classif.	F1 Weighted	LinearSVC	n	None	0.86858	0.85904	0.87855	0.88520
Multiclass Classif.	F1 Weighted	Logit Lasso	n	PCA	0.86755	0.85727	0.87805	0.89477
Multiclass Classif.	F1 Weighted	DecisionTreeClassifier	n	PCA	0.86671	0.85988	0.87366	0.86584
Multiclass Classif.	F1 Weighted	LinearSVC	n	PCA	0.86310	0.85366	0.87258	0.87905
Multiclass Classif.	F1 Weighted	DecisionTreeClassifier	n	KernelPCA-cosine	0.85439	0.84599	0.86296	0.85431
Multiclass Classif.	F1 Weighted	QDA	n	PCA	0.85356	0.84232	0.86520	0.87266
Multiclass Classif.	F1 Weighted	DecisionTreeClassifier	n	None	0.83706	0.83023	0.84389	0.83723
Multiclass Classif.	F1 Weighted	Logit Lasso	n	GaussRandProj	0.83559	0.82446	0.84696	0.87551
Multiclass Classif.	F1 Weighted	QDA	n	GaussRandProj	0.83491	0.82259	0.84769	0.87716
Multiclass Classif.	F1 Weighted	LinearSVC	n	GaussRandProj	0.83273	0.82127	0.84405	0.88491
Multiclass Classif.	F1 Weighted	Logit Ridge	n	GaussRandProj	0.83141	0.82070	0.84287	0.86336
Multiclass Classif.	F1 Weighted	Logit	n	GaussRandProj	0.83034	0.81895	0.84197	0.86336
Multiclass Classif.	F1 Weighted	QDA	n	Isomap	0.82628	0.81375	0.83916	0.87201
Multiclass Classif.	F1 Weighted	LDA	n	GaussRandProj	0.82042	0.80898	0.83172	0.86257
Multiclass Classif.	F1 Weighted	LDA	n	Isomap	0.81843	0.80759	0.82954	0.86209
Multiclass Classif.	F1 Weighted	QDA	n	KernelPCA-rbf	0.81718	0.80766	0.82686	0.84206
Multiclass Classif.	F1 Weighted	Logit	n	Isomap	0.81650	0.80596	0.82756	0.86257
Multiclass Classif.	F1 Weighted	Logit Ridge	n	Isomap	0.81640	0.80542	0.82752	0.86257
Multiclass Classif.	F1 Weighted	Logit Lasso	n	Isomap	0.81551	0.80484	0.82654	0.86267
Multiclass Classif.	F1 Weighted	LinearSVC	n	Isomap	0.81008	0.79922	0.82111	0.85039
Multiclass Classif.	F1 Weighted	Logit	n	k-Means	0.80836	0.79591	0.82107	0.83399
Multiclass Classif.	F1 Weighted	DecisionTreeClassifier	n	KernelPCA-poly3	0.80802	0.79816	0.81825	0.79286
Multiclass Classif.	F1 Weighted	DecisionTreeClassifier	n	k-Means	0.80726	0.79752	0.81703	0.79456
Multiclass Classif.	F1 Weighted	Logit Lasso	n	LLE-standard	0.80288	0.79049	0.81510	0.84799
Multiclass Classif.	F1 Weighted	Logit	n	LLE-standard	0.80135	0.78950	0.81338	0.84720
Multiclass Classif.	F1 Weighted	Logit Ridge	n	LLE-standard	0.80107	0.78907	0.81326	0.84708
Multiclass Classif.	F1 Weighted	Logit Lasso	n	k-Means	0.79967	0.78700	0.81287	0.81518
Multiclass Classif.	F1 Weighted	DecisionTreeClassifier	n	Isomap	0.79672	0.78615	0.80717	0.81619
Multiclass Classif.	F1 Weighted	Logit Ridge	n	k-Means	0.79450	0.78190	0.80775	0.80037
Multiclass Classif.	F1 Weighted	LDA	n	LLE-standard	0.79396	0.78129	0.80704	0.84124
Multiclass Classif.	F1 Weighted	NaiveBayesGaussian	n	None	0.79338	0.77981	0.80704	0.82509
Multiclass Classif.	F1 Weighted	LDA	n	k-Means	0.79325	0.78096	0.80564	0.78134
Multiclass Classif.	F1 Weighted	LinearSVC	n	LLE-standard	0.79177	0.77981	0.80376	0.83109
Multiclass Classif.	F1 Weighted	QDA	n	k-Means	0.79083	0.77636	0.80573	0.77201
Multiclass Classif.	F1 Weighted	DecisionTreeClassifier	n	LLE-standard	0.78623	0.77474	0.79769	0.82889
Multiclass Classif.	F1 Weighted	NaiveBayesGaussian	n	PCA	0.78580	0.77147	0.80015	0.84366
Multiclass Classif.	F1 Weighted	DecisionTreeClassifier	n	KernelPCA-rbf	0.77590	0.76564	0.78610	0.77047
Multiclass Classif.	F1 Weighted	DecisionTreeClassifier	n	GaussRandProj	0.76661	0.75653	0.77680	0.77224
Multiclass Classif.	F1 Weighted	DecisionTreeClassifier	n	KernelPCA-sigmoid	0.76097	0.75237	0.76954	0.76836
Multiclass Classif.	F1 Weighted	NaiveBayesGaussian	n	Isomap	0.75846	0.74461	0.77244	0.70719
Multiclass Classif.	F1 Weighted	QDA	n	LLE-standard	0.75533	0.73898	0.77219	0.70745
Multiclass Classif.	F1 Weighted	LinearSVC	n	k-Means	0.75100	0.73490	0.76710	0.76395
Multiclass Classif.	F1 Weighted	NaiveBayesGaussian	n	GaussRandProj	0.70637	0.69365	0.71916	0.66393
Multiclass Classif.	F1 Weighted	NaiveBayesGaussian	n	LLE-standard	0.70385	0.68770	0.71968	0.60175
Multiclass Classif.	F1 Weighted	QDA	n	KernelPCA-poly3	0.70021	0.68528	0.71497	0.68786
Multiclass Classif.	F1 Weighted	NaiveBayesGaussian	n	k-Means	0.64585	0.62479	0.66702	0.58769
Multiclass Classif.	F1 Weighted	QDA	n	KernelPCA-sigmoid	0.61452	0.60508	0.62406	0.58650
Multiclass Classif.	F1 Weighted	QDA	n	None	0.60079	0.56648	0.63558	0.72730
Multiclass Classif.	F1 Weighted	NaiveBayesGaussian	n	KernelPCA-poly3	0.43480	0.40515	0.46504	0.26402
Multiclass Classif.	F1 Weighted	NaiveBayesGaussian	n	KernelPCA-rbf	0.41649	0.39376	0.43884	0.46755
Multiclass Classif.	F1 Weighted	Logit Lasso	n	LLE-Modified	0.37581	0.36425	0.38721	0.33399
Multiclass Classif.	F1 Weighted	Logit	n	LLE-Modified	0.37576	0.36419	0.38734	0.33399
Multiclass Classif.	F1 Weighted	Logit Ridge	n	LLE-Modified	0.37516	0.36362	0.38668	0.33399
Multiclass Classif.	F1 Weighted	LDA	n	LLE-Modified	0.36759	0.35660	0.37930	0.31944
Multiclass Classif.	F1 Weighted	LDA	n	KernelPCA-rbf	0.35889	0.33873	0.37825	0.31129
Multiclass Classif.	F1 Weighted	DecisionTreeClassifier	n	LLE-Modified	0.35112	0.33963	0.36218	0.30046
Multiclass Classif.	F1 Weighted	LDA	n	KernelPCA-poly3	0.34128	0.32459	0.35742	0.29877
Multiclass Classif.	F1 Weighted	QDA	n	KernelPCA-cosine	0.33932	0.32149	0.35730	0.32619
Multiclass Classif.	F1 Weighted	LinearSVC	n	LLE-Modified	0.32116	0.30590	0.33668	0.29907
Multiclass Classif.	F1 Weighted	NaiveBayesGaussian	n	LLE-Modified	0.31157	0.29930	0.32416	0.30236
Multiclass Classif.	F1 Weighted	LDA	n	KernelPCA-sigmoid	0.23675	0.22059	0.25288	0.21899
Multiclass Classif.	F1 Weighted	LDA	n	KernelPCA-cosine	0.21659	0.20279	0.22976	0.20046
Multiclass Classif.	F1 Weighted	QDA	n	LLE-Modified	0.15255	0.14220	0.16277	0.16667
Multiclass Classif.	F1 Weighted	NaiveBayesGaussian	n	KernelPCA-sigmoid	0.14111	0.12842	0.15330	0.15158
Multiclass Classif.	Accuracy	XGBClassifier	y	None	0.98189	0.97712	0.98690	0.99244
Multiclass Classif.	Accuracy	NeuroNetClassifier	y	None	0.98011	0.97484	0.98555	1.00000
Multiclass Classif.	Accuracy	ExtraTreesClassifier	y	None	0.97756	0.97243	0.98264	0.99331
Multiclass Classif.	Accuracy	Logit Ridge	n	KernelPCA-poly3	0.97109	0.96515	0.97730	0.99997
Multiclass Classif.	Accuracy	Logit Lasso	n	KernelPCA-rbf	0.97085	0.96498	0.97702	0.99983
Multiclass Classif.	Accuracy	LinearSVC	n	KernelPCA-poly3	0.97013	0.96401	0.97657	0.99993
Multiclass Classif.	Accuracy	LinearSVC	n	KernelPCA-rbf	0.96834	0.96245	0.97428	0.99422
Multiclass Classif.	Accuracy	Logit Lasso	n	KernelPCA-poly3	0.96486	0.95847	0.97166	0.99415
Multiclass Classif.	Accuracy	Logit Ridge	n	KernelPCA-rbf	0.96476	0.95874	0.97091	0.99317
Multiclass Classif.	Accuracy	BaggedTrees	y	None	0.96309	0.95687	0.96934	0.98453
Multiclass Classif.	Accuracy	RandomForest	y	None	0.96261	0.95598	0.96953	0.98684
Multiclass Classif.	Accuracy	KernelSVC	y	None	0.96210	0.95538	0.96948	0.99787
Multiclass Classif.	Accuracy	Logit	n	KernelPCA-poly3	0.95561	0.94813	0.96349	0.99189
Multiclass Classif.	Accuracy	Logit Ridge	n	KernelPCA-sigmoid	0.94530	0.93869	0.95248	0.97019
Multiclass Classif.	Accuracy	LinearSVC	n	KernelPCA-sigmoid	0.94281	0.93628	0.94981	0.96255

Continued on next page

Problem Type	Metric	Model	Blackbox	Extractor	Mean	Mean CI 0.025	Mean CI 0.975	Median
Multiclass Classif.	Accuracy	Logit Lasso	n	KernelPCA-sigmoid	0.94147	0.93434	0.94894	0.96496
Multiclass Classif.	Accuracy	Logit	n	KernelPCA-rbf	0.93684	0.92914	0.94480	0.95774
Multiclass Classif.	Accuracy	Logit Lasso	n	KernelPCA-cosine	0.92326	0.91713	0.92971	0.92976
Multiclass Classif.	Accuracy	Logit Ridge	n	KernelPCA-cosine	0.91747	0.91106	0.92392	0.92391
Multiclass Classif.	Accuracy	KNeighborsClassifier	n	None	0.91590	0.90987	0.92219	0.93443
Multiclass Classif.	Accuracy	Logit	n	KernelPCA-cosine	0.91451	0.90776	0.92152	0.92391
Multiclass Classif.	Accuracy	Logit Ridge	n	None	0.91169	0.90494	0.91885	0.91807
Multiclass Classif.	Accuracy	Logit Lasso	n	None	0.91064	0.90371	0.91773	0.91908
Multiclass Classif.	Accuracy	LinearSVC	n	KernelPCA-cosine	0.90948	0.90263	0.91661	0.91803
Multiclass Classif.	Accuracy	Logit	n	None	0.90691	0.89973	0.91434	0.91805
Multiclass Classif.	Accuracy	Logit	n	PCA	0.89701	0.88869	0.90546	0.91755
Multiclass Classif.	Accuracy	LDA	n	None	0.89459	0.88720	0.90211	0.90164
Multiclass Classif.	Accuracy	Logit Ridge	n	PCA	0.89160	0.88251	0.90092	0.91329
Multiclass Classif.	Accuracy	Logit Lasso	n	PCA	0.88960	0.88079	0.89901	0.91226
Multiclass Classif.	Accuracy	LDA	n	PCA	0.88878	0.88112	0.89641	0.90140
Multiclass Classif.	Accuracy	LinearSVC	n	None	0.88171	0.87282	0.89100	0.90164
Multiclass Classif.	Accuracy	Logit	n	KernelPCA-sigmoid	0.88069	0.86905	0.89267	0.90164
Multiclass Classif.	Accuracy	NaiveBayesGaussian	n	KernelPCA-cosine	0.87838	0.87156	0.88506	0.87129
Multiclass Classif.	Accuracy	LinearSVC	n	PCA	0.87681	0.86786	0.88603	0.89734
Multiclass Classif.	Accuracy	DecisionTreeClassifier	n	PCA	0.86921	0.86242	0.87589	0.86885
Multiclass Classif.	Accuracy	Logit Lasso	n	GaussRandProj	0.86642	0.85728	0.87578	0.90164
Multiclass Classif.	Accuracy	Logit Ridge	n	GaussRandProj	0.86295	0.85364	0.87253	0.89380
Multiclass Classif.	Accuracy	LinearSVC	n	GaussRandProj	0.86218	0.85261	0.87190	0.90164
Multiclass Classif.	Accuracy	Logit	n	GaussRandProj	0.86073	0.85139	0.87032	0.89286
Multiclass Classif.	Accuracy	DecisionTreeClassifier	n	KernelPCA-cosine	0.85595	0.84785	0.86419	0.84764
Multiclass Classif.	Accuracy	QDA	n	PCA	0.85461	0.84446	0.86489	0.85659
Multiclass Classif.	Accuracy	Logit	n	Isomap	0.85454	0.84559	0.86375	0.89884
Multiclass Classif.	Accuracy	Logit Ridge	n	Isomap	0.85435	0.84518	0.86354	0.89884
Multiclass Classif.	Accuracy	Logit Lasso	n	Isomap	0.85387	0.84472	0.86279	0.89743
Multiclass Classif.	Accuracy	Logit	n	k-Means	0.85252	0.84278	0.86226	0.86206
Multiclass Classif.	Accuracy	Logit Lasso	n	LLE-standard	0.84969	0.84054	0.85908	0.88525
Multiclass Classif.	Accuracy	LDA	n	Isomap	0.84864	0.83967	0.85768	0.87500
Multiclass Classif.	Accuracy	Logit Lasso	n	k-Means	0.84811	0.83829	0.85787	0.83333
Multiclass Classif.	Accuracy	Logit Ridge	n	LLE-standard	0.84795	0.83864	0.85740	0.88525
Multiclass Classif.	Accuracy	Logit	n	LLE-standard	0.84770	0.83850	0.85705	0.88525
Multiclass Classif.	Accuracy	QDA	n	GaussRandProj	0.84749	0.83691	0.85799	0.86451
Multiclass Classif.	Accuracy	LinearSVC	n	Isomap	0.84643	0.83742	0.85540	0.87787
Multiclass Classif.	Accuracy	LDA	n	GaussRandProj	0.84638	0.83666	0.85617	0.87836
Multiclass Classif.	Accuracy	Logit Ridge	n	k-Means	0.84382	0.83360	0.85375	0.82485
Multiclass Classif.	Accuracy	LinearSVC	n	LLE-standard	0.83999	0.83092	0.84899	0.86885
Multiclass Classif.	Accuracy	QDA	n	Isomap	0.83752	0.82670	0.84832	0.84465
Multiclass Classif.	Accuracy	DecisionTreeClassifier	n	None	0.83403	0.82759	0.84064	0.83333
Multiclass Classif.	Accuracy	LDA	n	LLE-standard	0.83230	0.82269	0.84226	0.85445
Multiclass Classif.	Accuracy	LDA	n	k-Means	0.82495	0.81406	0.83591	0.79545
Multiclass Classif.	Accuracy	QDA	n	KernelPCA-rbf	0.82249	0.81314	0.83176	0.84393
Multiclass Classif.	Accuracy	DecisionTreeClassifier	n	Isomap	0.81890	0.80965	0.82841	0.83607
Multiclass Classif.	Accuracy	DecisionTreeClassifier	n	LLE-standard	0.81318	0.80483	0.82143	0.82846
Multiclass Classif.	Accuracy	DecisionTreeClassifier	n	k-Means	0.81198	0.80247	0.82134	0.79545
Multiclass Classif.	Accuracy	DecisionTreeClassifier	n	KernelPCA-poly3	0.81059	0.80047	0.82070	0.80052
Multiclass Classif.	Accuracy	NaiveBayesGaussian	n	PCA	0.80688	0.79404	0.81992	0.85549
Multiclass Classif.	Accuracy	LinearSVC	n	k-Means	0.80022	0.78622	0.81414	0.80241
Multiclass Classif.	Accuracy	DecisionTreeClassifier	n	KernelPCA-rbf	0.80015	0.79029	0.80973	0.80328
Multiclass Classif.	Accuracy	NaiveBayesGaussian	n	None	0.79680	0.78300	0.81117	0.80925
Multiclass Classif.	Accuracy	QDA	n	k-Means	0.79238	0.77770	0.80717	0.77588
Multiclass Classif.	Accuracy	NaiveBayesGaussian	n	Isomap	0.78966	0.77716	0.80220	0.74415
Multiclass Classif.	Accuracy	DecisionTreeClassifier	n	KernelPCA-sigmoid	0.78695	0.77914	0.79492	0.80328
Multiclass Classif.	Accuracy	QDA	n	LLE-standard	0.76601	0.75049	0.78145	0.70652
Multiclass Classif.	Accuracy	DecisionTreeClassifier	n	GaussRandProj	0.76568	0.75545	0.77595	0.77182
Multiclass Classif.	Accuracy	NaiveBayesGaussian	n	LLE-standard	0.74291	0.72808	0.75738	0.70586
Multiclass Classif.	Accuracy	NaiveBayesGaussian	n	GaussRandProj	0.73281	0.72086	0.74482	0.71013
Multiclass Classif.	Accuracy	QDA	n	KernelPCA-poly3	0.72323	0.71002	0.73622	0.70877
Multiclass Classif.	Accuracy	NaiveBayesGaussian	n	k-Means	0.68016	0.66002	0.70062	0.65886
Multiclass Classif.	Accuracy	QDA	n	KernelPCA-sigmoid	0.64782	0.63848	0.65712	0.67213
Multiclass Classif.	Accuracy	QDA	n	None	0.63063	0.59978	0.66094	0.70211
Multiclass Classif.	Accuracy	Logit	n	LLE-Modified	0.51711	0.50675	0.52708	0.49180
Multiclass Classif.	Accuracy	Logit Lasso	n	LLE-Modified	0.51705	0.50677	0.52756	0.49180
Multiclass Classif.	Accuracy	Logit Ridge	n	LLE-Modified	0.51683	0.50624	0.52733	0.49180
Multiclass Classif.	Accuracy	LDA	n	LLE-Modified	0.51076	0.50027	0.52073	0.48197
Multiclass Classif.	Accuracy	NaiveBayesGaussian	n	KernelPCA-poly3	0.50872	0.48366	0.53410	0.41667
Multiclass Classif.	Accuracy	DecisionTreeClassifier	n	LLE-Modified	0.47989	0.46699	0.49225	0.47541
Multiclass Classif.	Accuracy	NaiveBayesGaussian	n	KernelPCA-rbf	0.46980	0.44907	0.49012	0.48583
Multiclass Classif.	Accuracy	LinearSVC	n	LLE-Modified	0.45990	0.44445	0.47548	0.47541
Multiclass Classif.	Accuracy	NaiveBayesGaussian	n	LLE-Modified	0.41824	0.40370	0.43279	0.47541
Multiclass Classif.	Accuracy	LDA	n	KernelPCA-rbf	0.41204	0.39181	0.43209	0.37500
Multiclass Classif.	Accuracy	QDA	n	KernelPCA-cosine	0.39623	0.37755	0.41511	0.38889
Multiclass Classif.	Accuracy	LDA	n	KernelPCA-poly3	0.36693	0.35130	0.38247	0.35595
Multiclass Classif.	Accuracy	LDA	n	KernelPCA-sigmoid	0.26590	0.25081	0.28078	0.24590
Multiclass Classif.	Accuracy	QDA	n	LLE-Modified	0.25861	0.24757	0.26936	0.22543
Multiclass Classif.	Accuracy	NaiveBayesGaussian	n	KernelPCA-sigmoid	0.23124	0.21649	0.24597	0.21929
Multiclass Classif.	Accuracy	LDA	n	KernelPCA-cosine	0.21769	0.20464	0.23068	0.19297
Binary Classif.	Somers' D	XGBClassifier	y	None	0.94710	0.94165	0.95273	0.98701
Binary Classif.	Somers' D	NeuroNetClassifier	y	None	0.93387	0.92863	0.93927	0.97675
Binary Classif.	Somers' D	LinearSVC	n	KernelPCA-rbf	0.93387	0.92884	0.93898	0.97206
Binary Classif.	Somers' D	LinearSVC	n	KernelPCA-poly3	0.93365	0.92893	0.93872	0.96260
Binary Classif.	Somers' D	Logit Ridge	n	KernelPCA-rbf	0.93342	0.92823	0.93856	0.97550

Continued on next page

Problem Type	Metric	Model	Blackbox	Extractor	Mean	Mean CI 0.025	Mean CI 0.975	Median
Binary Classif.	Somers' D	Logit Ridge	n	KernelPCA-poly3	0.93193	0.92698	0.93696	0.96558
Binary Classif.	Somers' D	RandomForest	y	None	0.93134	0.92575	0.93699	0.97516
Binary Classif.	Somers' D	BaggedTrees	y	None	0.92903	0.92321	0.93489	0.97434
Binary Classif.	Somers' D	Logit Ridge	n	KernelPCA-sigmoid	0.92749	0.92239	0.93265	0.96422
Binary Classif.	Somers' D	LinearSVC	n	KernelPCA-sigmoid	0.92719	0.92213	0.93249	0.96165
Binary Classif.	Somers' D	Logit Ridge	n	None	0.92513	0.92009	0.93020	0.96471
Binary Classif.	Somers' D	LinearSVC	n	None	0.92444	0.91935	0.92971	0.96366
Binary Classif.	Somers' D	Logit Lasso	n	KernelPCA-sigmoid	0.92151	0.91614	0.92683	0.96315
Binary Classif.	Somers' D	Logit Ridge	n	KernelPCA-cosine	0.92113	0.91583	0.92651	0.96307
Binary Classif.	Somers' D	Logit Lasso	n	KernelPCA-poly3	0.92093	0.91566	0.92620	0.95253
Binary Classif.	Somers' D	Logit Lasso	n	None	0.91984	0.91465	0.92510	0.95871
Binary Classif.	Somers' D	Logit Lasso	n	KernelPCA-rbf	0.91951	0.91398	0.92513	0.96251
Binary Classif.	Somers' D	LinearSVC	n	KernelPCA-cosine	0.91747	0.91203	0.92309	0.96007
Binary Classif.	Somers' D	Logit Ridge	n	PCA	0.91741	0.91234	0.92278	0.95489
Binary Classif.	Somers' D	Logit Lasso	n	KernelPCA-cosine	0.91628	0.91068	0.92189	0.96210
Binary Classif.	Somers' D	Logit Lasso	n	PCA	0.91323	0.90806	0.91869	0.95240
Binary Classif.	Somers' D	LinearSVC	n	PCA	0.91065	0.90549	0.91603	0.94432
Binary Classif.	Somers' D	LDA	n	None	0.90770	0.90205	0.91343	0.94735
Binary Classif.	Somers' D	Logit	n	None	0.90505	0.89908	0.91118	0.95055
Binary Classif.	Somers' D	Logit	n	KernelPCA-cosine	0.90255	0.89626	0.90874	0.95294
Binary Classif.	Somers' D	Logit	n	PCA	0.90112	0.89525	0.90718	0.94375
Binary Classif.	Somers' D	LDA	n	PCA	0.90106	0.89566	0.90672	0.93614
Binary Classif.	Somers' D	KNeighborsClassifier	n	None	0.89951	0.89383	0.90516	0.93933
Binary Classif.	Somers' D	KernelSVC	y	None	0.89881	0.89202	0.90594	0.95790
Binary Classif.	Somers' D	ExtraTreesClassifier	y	None	0.88924	0.88237	0.89615	0.93204
Binary Classif.	Somers' D	DecisionTreeClassifier	n	None	0.82229	0.81309	0.83148	0.88235
Binary Classif.	Somers' D	QDA	n	PCA	0.80121	0.79340	0.80909	0.80984
Binary Classif.	Somers' D	Logit	n	GaussRandProj	0.77932	0.77048	0.78822	0.81376
Binary Classif.	Somers' D	Logit	n	KernelPCA-poly3	0.77911	0.76878	0.78957	0.83246
Binary Classif.	Somers' D	Logit	n	KernelPCA-rbf	0.77512	0.76575	0.78484	0.79859
Binary Classif.	Somers' D	NaiveBayesGaussian	n	PCA	0.77404	0.76603	0.78216	0.77718
Binary Classif.	Somers' D	Logit Lasso	n	GaussRandProj	0.77326	0.76414	0.78235	0.80059
Binary Classif.	Somers' D	LDA	n	GaussRandProj	0.77271	0.76398	0.78161	0.79281
Binary Classif.	Somers' D	DecisionTreeClassifier	n	KernelPCA-cosine	0.76992	0.76157	0.77788	0.79302
Binary Classif.	Somers' D	Logit	n	k-Means	0.76850	0.75918	0.77748	0.79729
Binary Classif.	Somers' D	LinearSVC	n	GaussRandProj	0.76627	0.75727	0.77561	0.78542
Binary Classif.	Somers' D	Logit Ridge	n	GaussRandProj	0.76528	0.75577	0.77468	0.79420
Binary Classif.	Somers' D	Logit Ridge	n	k-Means	0.76487	0.75548	0.77429	0.79519
Binary Classif.	Somers' D	DecisionTreeClassifier	n	PCA	0.76481	0.75667	0.77294	0.79137
Binary Classif.	Somers' D	Logit Lasso	n	k-Means	0.76452	0.75513	0.77393	0.79252
Binary Classif.	Somers' D	QDA	n	GaussRandProj	0.76380	0.75478	0.77289	0.76311
Binary Classif.	Somers' D	LDA	n	k-Means	0.76227	0.75325	0.77149	0.78872
Binary Classif.	Somers' D	NaiveBayesGaussian	n	KernelPCA-cosine	0.76209	0.75385	0.77028	0.78062
Binary Classif.	Somers' D	NaiveBayesGaussian	n	None	0.74901	0.73648	0.76127	0.82170
Binary Classif.	Somers' D	LinearSVC	n	k-Means	0.73229	0.72131	0.74312	0.76471
Binary Classif.	Somers' D	QDA	n	k-Means	0.71782	0.70777	0.72771	0.72646
Binary Classif.	Somers' D	QDA	n	KernelPCA-rbf	0.70449	0.69261	0.71633	0.72601
Binary Classif.	Somers' D	DecisionTreeClassifier	n	KernelPCA-poly3	0.70171	0.69137	0.71213	0.74484
Binary Classif.	Somers' D	QDA	n	Isomap	0.69949	0.68829	0.71091	0.71727
Binary Classif.	Somers' D	DecisionTreeClassifier	n	KernelPCA-sigmoid	0.69520	0.68613	0.70455	0.72172
Binary Classif.	Somers' D	NaiveBayesGaussian	n	GaussRandProj	0.69238	0.68321	0.70144	0.67517
Binary Classif.	Somers' D	DecisionTreeClassifier	n	k-Means	0.67348	0.66208	0.68482	0.71958
Binary Classif.	Somers' D	Logit Ridge	n	Isomap	0.65809	0.64578	0.67091	0.71463
Binary Classif.	Somers' D	Logit Lasso	n	Isomap	0.65764	0.64473	0.67027	0.71590
Binary Classif.	Somers' D	Logit	n	Isomap	0.65673	0.64386	0.66976	0.71006
Binary Classif.	Somers' D	NaiveBayesGaussian	n	Isomap	0.65510	0.64386	0.66650	0.67465
Binary Classif.	Somers' D	LDA	n	Isomap	0.64139	0.62800	0.65478	0.71147
Binary Classif.	Somers' D	LinearSVC	n	Isomap	0.64112	0.62783	0.65411	0.70529
Binary Classif.	Somers' D	QDA	n	None	0.64097	0.62671	0.65570	0.64691
Binary Classif.	Somers' D	DecisionTreeClassifier	n	KernelPCA-rbf	0.64022	0.62783	0.65277	0.71156
Binary Classif.	Somers' D	DecisionTreeClassifier	n	GaussRandProj	0.62981	0.61865	0.64104	0.65594
Binary Classif.	Somers' D	DecisionTreeClassifier	n	Isomap	0.62461	0.61384	0.63534	0.63871
Binary Classif.	Somers' D	DecisionTreeClassifier	n	LLE-standard	0.61851	0.60765	0.62963	0.61657
Binary Classif.	Somers' D	Logit	n	KernelPCA-sigmoid	0.59487	0.57821	0.61196	0.66565
Binary Classif.	Somers' D	QDA	n	LLE-standard	0.59259	0.58069	0.60495	0.55593
Binary Classif.	Somers' D	QDA	n	KernelPCA-poly3	0.59022	0.57725	0.60324	0.59347
Binary Classif.	Somers' D	NaiveBayesGaussian	n	k-Means	0.57050	0.55762	0.58350	0.57705
Binary Classif.	Somers' D	Logit Lasso	n	LLE-standard	0.55299	0.53962	0.56672	0.53044
Binary Classif.	Somers' D	Logit	n	LLE-standard	0.55197	0.53869	0.56553	0.52583
Binary Classif.	Somers' D	LDA	n	LLE-standard	0.54673	0.53324	0.56032	0.51877
Binary Classif.	Somers' D	Logit Ridge	n	LLE-standard	0.54596	0.53207	0.55978	0.51460
Binary Classif.	Somers' D	LinearSVC	n	LLE-standard	0.53968	0.52565	0.55373	0.51014
Binary Classif.	Somers' D	NaiveBayesGaussian	n	LLE-standard	0.49717	0.48454	0.50988	0.46410
Binary Classif.	Somers' D	NaiveBayesGaussian	n	KernelPCA-poly3	0.24096	0.22407	0.25780	0.10578
Binary Classif.	Somers' D	NaiveBayesGaussian	n	KernelPCA-rbf	0.22035	0.20667	0.23392	0.14850
Binary Classif.	Somers' D	NaiveBayesGaussian	n	LLE-Modified	0.15472	0.14314	0.16591	0.09789
Binary Classif.	Somers' D	LDA	n	KernelPCA-rbf	0.13590	0.12181	0.14949	0.06480
Binary Classif.	Somers' D	DecisionTreeClassifier	n	LLE-Modified	0.13415	0.12199	0.14606	0.03249
Binary Classif.	Somers' D	LDA	n	KernelPCA-poly3	0.07965	0.06805	0.09103	0.04125
Binary Classif.	Somers' D	QDA	n	KernelPCA-sigmoid	0.06030	0.04366	0.07656	-0.00694
Binary Classif.	Somers' D	Logit Lasso	n	LLE-Modified	0.05722	0.05000	0.06436	0.00000
Binary Classif.	Somers' D	Logit	n	LLE-Modified	0.05698	0.04908	0.06454	0.03392
Binary Classif.	Somers' D	LDA	n	LLE-Modified	0.05691	0.04927	0.06436	0.03180
Binary Classif.	Somers' D	Logit Ridge	n	LLE-Modified	0.05668	0.04893	0.06409	0.03342
Binary Classif.	Somers' D	LinearSVC	n	LLE-Modified	0.05590	0.04812	0.06350	0.03377
Binary Classif.	Somers' D	NaiveBayesGaussian	n	KernelPCA-sigmoid	0.04657	0.03623	0.05666	0.00000

Continued on next page

Problem Type	Metric	Model	Blackbox	Extractor	Mean	Mean CI 0.025	Mean CI 0.975	Median
Binary Classif.	Somers' D	QDA	n	KernelPCA-cosine	0.03877	0.02328	0.05397	0.00925
Binary Classif.	Somers' D	LDA	n	KernelPCA-cosine	0.03377	0.01706	0.05098	0.04425
Binary Classif.	Somers' D	LDA	n	KernelPCA-sigmoid	0.01777	0.00936	0.02635	0.01551
Binary Classif.	F1 Score	RandomForest	y	None	0.92256	0.91560	0.92962	0.96277
Binary Classif.	F1 Score	BaggedTrees	y	None	0.92165	0.91504	0.92848	0.96016
Binary Classif.	F1 Score	ExtraTreesClassifier	y	None	0.91513	0.90911	0.92148	0.95165
Binary Classif.	F1 Score	Logit	n	KernelPCA-poly3	0.89578	0.89022	0.90147	0.91327
Binary Classif.	F1 Score	XGBClassifier	y	None	0.88581	0.87407	0.89809	0.95952
Binary Classif.	F1 Score	LDA	n	None	0.88075	0.87346	0.88842	0.92074
Binary Classif.	F1 Score	Logit	n	KernelPCA-rbf	0.88068	0.87525	0.88637	0.88864
Binary Classif.	F1 Score	Logit Lasso	n	KernelPCA-rbf	0.88034	0.86972	0.89152	0.94055
Binary Classif.	F1 Score	Logit Ridge	n	KernelPCA-rbf	0.87812	0.86685	0.88977	0.94231
Binary Classif.	F1 Score	Logit Ridge	n	KernelPCA-poly3	0.87782	0.86720	0.88915	0.94030
Binary Classif.	F1 Score	KernelSVC	y	None	0.87365	0.86169	0.88617	0.95070
Binary Classif.	F1 Score	Logit Ridge	n	None	0.87339	0.86332	0.88394	0.93651
Binary Classif.	F1 Score	Logit Lasso	n	KernelPCA-sigmoid	0.87255	0.86232	0.88326	0.93506
Binary Classif.	F1 Score	Logit Lasso	n	KernelPCA-poly3	0.87086	0.86044	0.88190	0.93462
Binary Classif.	F1 Score	Logit Lasso	n	None	0.87015	0.85999	0.88052	0.92978
Binary Classif.	F1 Score	Logit Ridge	n	PCA	0.86973	0.85904	0.88059	0.93425
Binary Classif.	F1 Score	Logit	n	None	0.86905	0.85915	0.87965	0.93103
Binary Classif.	F1 Score	LinearSVC	n	KernelPCA-rbf	0.86896	0.85719	0.88091	0.93750
Binary Classif.	F1 Score	LDA	n	PCA	0.86863	0.86084	0.87685	0.90873
Binary Classif.	F1 Score	LinearSVC	n	KernelPCA-poly3	0.86550	0.85455	0.87692	0.93462
Binary Classif.	F1 Score	Logit	n	PCA	0.86541	0.85531	0.87581	0.92757
Binary Classif.	F1 Score	NeuroNetClassifier	y	None	0.86400	0.85201	0.87618	0.94862
Binary Classif.	F1 Score	Logit Lasso	n	KernelPCA-cosine	0.86339	0.85232	0.87496	0.93872
Binary Classif.	F1 Score	Logit	n	KernelPCA-cosine	0.86231	0.85171	0.87350	0.93087
Binary Classif.	F1 Score	Logit Lasso	n	PCA	0.85887	0.84826	0.86979	0.91846
Binary Classif.	F1 Score	QDA	n	Isomap	0.85857	0.85158	0.86564	0.86629
Binary Classif.	F1 Score	Logit Ridge	n	KernelPCA-cosine	0.85353	0.84208	0.86551	0.93312
Binary Classif.	F1 Score	LinearSVC	n	None	0.85210	0.84052	0.86398	0.93151
Binary Classif.	F1 Score	LinearSVC	n	PCA	0.85165	0.84087	0.86271	0.92279
Binary Classif.	F1 Score	QDA	n	KernelPCA-rbf	0.84983	0.84352	0.85623	0.87293
Binary Classif.	F1 Score	DecisionTreeClassifier	n	None	0.84380	0.83321	0.85455	0.91780
Binary Classif.	F1 Score	Logit	n	KernelPCA-sigmoid	0.82360	0.81705	0.83029	0.84608
Binary Classif.	F1 Score	DecisionTreeClassifier	n	PCA	0.81334	0.80340	0.82350	0.87617
Binary Classif.	F1 Score	DecisionTreeClassifier	n	KernelPCA-poly3	0.81104	0.80345	0.81885	0.84641
Binary Classif.	F1 Score	LinearSVC	n	KernelPCA-cosine	0.81030	0.79743	0.82344	0.92213
Binary Classif.	F1 Score	DecisionTreeClassifier	n	KernelPCA-rbf	0.80719	0.79948	0.81511	0.84028
Binary Classif.	F1 Score	LinearSVC	n	KernelPCA-sigmoid	0.80356	0.79024	0.81710	0.92647
Binary Classif.	F1 Score	NaiveBayesGaussian	n	PCA	0.80337	0.79675	0.81009	0.80576
Binary Classif.	F1 Score	QDA	n	GaussRandProj	0.80187	0.79547	0.80857	0.81809
Binary Classif.	F1 Score	DecisionTreeClassifier	n	KernelPCA-cosine	0.80130	0.78898	0.81447	0.88514
Binary Classif.	F1 Score	Logit Ridge	n	KernelPCA-sigmoid	0.79236	0.77755	0.80785	0.92813
Binary Classif.	F1 Score	KNeighborsClassifier	n	None	0.78623	0.77351	0.79936	0.88000
Binary Classif.	F1 Score	QDA	n	PCA	0.78489	0.77677	0.79324	0.81953
Binary Classif.	F1 Score	QDA	n	KernelPCA-poly3	0.78322	0.77294	0.79360	0.82371
Binary Classif.	F1 Score	NaiveBayesGaussian	n	GaussRandProj	0.78226	0.77392	0.79064	0.78108
Binary Classif.	F1 Score	NaiveBayesGaussian	n	Isomap	0.77427	0.76620	0.78214	0.79487
Binary Classif.	F1 Score	DecisionTreeClassifier	n	k-Means	0.76718	0.75702	0.77765	0.83234
Binary Classif.	F1 Score	QDA	n	k-Means	0.76704	0.75807	0.77635	0.81818
Binary Classif.	F1 Score	NaiveBayesGaussian	n	None	0.76164	0.75287	0.77023	0.79412
Binary Classif.	F1 Score	DecisionTreeClassifier	n	Isomap	0.76019	0.74988	0.77059	0.82623
Binary Classif.	F1 Score	Logit Ridge	n	Isomap	0.74434	0.73161	0.75705	0.83294
Binary Classif.	F1 Score	DecisionTreeClassifier	n	KernelPCA-sigmoid	0.74174	0.73000	0.75352	0.80384
Binary Classif.	F1 Score	Logit Ridge	n	GaussRandProj	0.73958	0.72674	0.75259	0.84836
Binary Classif.	F1 Score	Logit	n	GaussRandProj	0.73784	0.72536	0.75018	0.83212
Binary Classif.	F1 Score	QDA	n	None	0.73410	0.72532	0.74297	0.74157
Binary Classif.	F1 Score	Logit Lasso	n	GaussRandProj	0.72994	0.71726	0.74308	0.82812
Binary Classif.	F1 Score	DecisionTreeClassifier	n	GaussRandProj	0.72828	0.71711	0.73979	0.79730
Binary Classif.	F1 Score	Logit	n	k-Means	0.72752	0.71380	0.74164	0.84383
Binary Classif.	F1 Score	LDA	n	GaussRandProj	0.72735	0.71539	0.73929	0.83036
Binary Classif.	F1 Score	LDA	n	k-Means	0.72724	0.71430	0.74041	0.83033
Binary Classif.	F1 Score	NaiveBayesGaussian	n	KernelPCA-cosine	0.72600	0.71810	0.73401	0.75231
Binary Classif.	F1 Score	Logit Lasso	n	k-Means	0.72474	0.71112	0.73846	0.84145
Binary Classif.	F1 Score	Logit Ridge	n	k-Means	0.72458	0.71107	0.73847	0.84375
Binary Classif.	F1 Score	LinearSVC	n	Isomap	0.72119	0.70707	0.73552	0.83505
Binary Classif.	F1 Score	DecisionTreeClassifier	n	LLE-standard	0.71823	0.70583	0.73059	0.78469
Binary Classif.	F1 Score	LinearSVC	n	GaussRandProj	0.70827	0.69409	0.72241	0.83847
Binary Classif.	F1 Score	QDA	n	LLE-standard	0.68798	0.67914	0.69698	0.71759
Binary Classif.	F1 Score	Logit Lasso	n	Isomap	0.66692	0.65204	0.68263	0.80320
Binary Classif.	F1 Score	Logit	n	Isomap	0.66387	0.64925	0.67904	0.79397
Binary Classif.	F1 Score	NaiveBayesGaussian	n	LLE-standard	0.64756	0.63779	0.65745	0.67982
Binary Classif.	F1 Score	LDA	n	Isomap	0.64040	0.62484	0.65574	0.79339
Binary Classif.	F1 Score	NaiveBayesGaussian	n	k-Means	0.60111	0.59000	0.61232	0.64911
Binary Classif.	F1 Score	LinearSVC	n	k-Means	0.57746	0.55887	0.59627	0.70525
Binary Classif.	F1 Score	NaiveBayesGaussian	n	LLE-Modified	0.54948	0.53599	0.56327	0.58333
Binary Classif.	F1 Score	LDA	n	LLE-standard	0.53819	0.52164	0.55473	0.67500
Binary Classif.	F1 Score	LDA	n	KernelPCA-cosine	0.53331	0.52263	0.54402	0.54641
Binary Classif.	F1 Score	LDA	n	KernelPCA-sigmoid	0.53176	0.52225	0.54108	0.53731
Binary Classif.	F1 Score	Logit	n	LLE-standard	0.53073	0.51294	0.54876	0.69421
Binary Classif.	F1 Score	Logit Lasso	n	LLE-standard	0.53010	0.51247	0.54800	0.68103
Binary Classif.	F1 Score	LDA	n	KernelPCA-rbf	0.51811	0.50585	0.53058	0.54588
Binary Classif.	F1 Score	LDA	n	KernelPCA-poly3	0.51439	0.50412	0.52516	0.52665
Binary Classif.	F1 Score	NaiveBayesGaussian	n	KernelPCA-poly3	0.46432	0.44762	0.48126	0.60393

Continued on next page

Problem Type	Metric	Model	Blackbox	Extractor	Mean	Mean CI 0.025	Mean CI 0.975	Median
Binary Classif.	F1 Score	NaiveBayesGaussian	n	KernelPCA-rbf	0.41701	0.40210	0.43202	0.50267
Binary Classif.	F1 Score	Logit Ridge	n	LLE-standard	0.39037	0.37074	0.41059	0.27705
Binary Classif.	F1 Score	LinearSVC	n	LLE-standard	0.33212	0.31297	0.35068	0.04510
Binary Classif.	F1 Score	DecisionTreeClassifier	n	LLE-Modified	0.33079	0.31542	0.34628	0.26663
Binary Classif.	F1 Score	NaiveBayesGaussian	n	KernelPCA-sigmoid	0.28574	0.27141	0.30056	0.20184
Binary Classif.	F1 Score	QDA	n	KernelPCA-cosine	0.23433	0.21602	0.25239	0.00000
Binary Classif.	F1 Score	LinearSVC	n	LLE-Modified	0.22337	0.20635	0.24023	0.00000
Binary Classif.	F1 Score	QDA	n	KernelPCA-sigmoid	0.19168	0.17547	0.20744	0.00000
Binary Classif.	F1 Score	Logit	n	LLE-Modified	0.17027	0.15638	0.18355	0.02211
Binary Classif.	F1 Score	LDA	n	LLE-Modified	0.15358	0.13974	0.16692	0.00000
Binary Classif.	F1 Score	Logit Lasso	n	LLE-Modified	0.14963	0.13583	0.16318	0.00000
Binary Classif.	F1 Score	Logit Ridge	n	LLE-Modified	0.08789	0.07887	0.09640	0.00000
Binary Classif.	Accuracy	XGBClassifier	y	None	0.97580	0.97349	0.97820	0.99501
Binary Classif.	Accuracy	LinearSVC	n	KernelPCA-poly3	0.97429	0.97211	0.97655	0.98991
Binary Classif.	Accuracy	Logit Ridge	n	KernelPCA-rbf	0.97397	0.97177	0.97616	0.99077
Binary Classif.	Accuracy	LinearSVC	n	KernelPCA-rbf	0.97388	0.97169	0.97613	0.99126
Binary Classif.	Accuracy	Logit Ridge	n	KernelPCA-poly3	0.97349	0.97129	0.97578	0.98987
Binary Classif.	Accuracy	RandomForest	y	None	0.97308	0.97080	0.97542	0.98956
Binary Classif.	Accuracy	Logit Ridge	n	None	0.97291	0.97073	0.97511	0.98618
Binary Classif.	Accuracy	LinearSVC	n	None	0.97278	0.97067	0.97487	0.98462
Binary Classif.	Accuracy	BaggedTrees	y	None	0.97193	0.96965	0.97434	0.98803
Binary Classif.	Accuracy	Logit Lasso	n	None	0.97160	0.96952	0.97378	0.98667
Binary Classif.	Accuracy	Logit Lasso	n	KernelPCA-poly3	0.97112	0.96887	0.97333	0.98676
Binary Classif.	Accuracy	NeuroNetClassifier	y	None	0.97023	0.96780	0.97271	0.98987
Binary Classif.	Accuracy	Logit Ridge	n	KernelPCA-cosine	0.97019	0.96798	0.97251	0.98579
Binary Classif.	Accuracy	Logit Lasso	n	KernelPCA-cosine	0.96857	0.96611	0.97103	0.98573
Binary Classif.	Accuracy	Logit Lasso	n	KernelPCA-rbf	0.96827	0.96578	0.97082	0.98684
Binary Classif.	Accuracy	KernelSVC	y	None	0.96825	0.96609	0.97043	0.98246
Binary Classif.	Accuracy	Logit Lasso	n	KernelPCA-sigmoid	0.96769	0.96523	0.97020	0.98492
Binary Classif.	Accuracy	Logit Ridge	n	PCA	0.96715	0.96484	0.96954	0.98280
Binary Classif.	Accuracy	Logit	n	PCA	0.96590	0.96348	0.96840	0.98246
Binary Classif.	Accuracy	LinearSVC	n	PCA	0.96581	0.96348	0.96817	0.98048
Binary Classif.	Accuracy	Logit Lasso	n	PCA	0.96579	0.96335	0.96827	0.97996
Binary Classif.	Accuracy	Logit	n	None	0.96576	0.96314	0.96835	0.98482
Binary Classif.	Accuracy	LinearSVC	n	KernelPCA-cosine	0.96506	0.96277	0.96749	0.97990
Binary Classif.	Accuracy	LDA	n	None	0.96447	0.96216	0.96682	0.98154
Binary Classif.	Accuracy	Logit	n	KernelPCA-cosine	0.96402	0.96149	0.96662	0.98261
Binary Classif.	Accuracy	ExtraTreesClassifier	y	None	0.96392	0.96153	0.96636	0.97929
Binary Classif.	Accuracy	LinearSVC	n	KernelPCA-sigmoid	0.96350	0.96088	0.96618	0.98329
Binary Classif.	Accuracy	LDA	n	PCA	0.96190	0.95956	0.96423	0.97671
Binary Classif.	Accuracy	Logit Ridge	n	KernelPCA-sigmoid	0.95935	0.95604	0.96273	0.98502
Binary Classif.	Accuracy	KNeighborsClassifier	n	None	0.95725	0.95498	0.95953	0.96733
Binary Classif.	Accuracy	DecisionTreeClassifier	n	None	0.94625	0.94291	0.94963	0.97219
Binary Classif.	Accuracy	DecisionTreeClassifier	n	KernelPCA-cosine	0.93836	0.93537	0.94147	0.96149
Binary Classif.	Accuracy	DecisionTreeClassifier	n	PCA	0.93745	0.93440	0.94050	0.95703
Binary Classif.	Accuracy	Logit	n	KernelPCA-poly3	0.93650	0.93323	0.93994	0.95729
Binary Classif.	Accuracy	Logit	n	KernelPCA-rbf	0.93262	0.92949	0.93572	0.94824
Binary Classif.	Accuracy	Logit	n	GaussRandProj	0.92757	0.92463	0.93062	0.93140
Binary Classif.	Accuracy	Logit Lasso	n	GaussRandProj	0.92558	0.92241	0.92884	0.93140
Binary Classif.	Accuracy	LinearSVC	n	GaussRandProj	0.92391	0.92059	0.92727	0.93287
Binary Classif.	Accuracy	LDA	n	GaussRandProj	0.92282	0.91976	0.92580	0.92462
Binary Classif.	Accuracy	Logit Ridge	n	GaussRandProj	0.92213	0.91892	0.92541	0.92649
Binary Classif.	Accuracy	Logit Lasso	n	k-Means	0.91924	0.91511	0.92339	0.93877
Binary Classif.	Accuracy	Logit	n	k-Means	0.91908	0.91503	0.92330	0.93938
Binary Classif.	Accuracy	Logit Ridge	n	k-Means	0.91885	0.91477	0.92306	0.93844
Binary Classif.	Accuracy	QDA	n	KernelPCA-rbf	0.91686	0.91329	0.92066	0.93798
Binary Classif.	Accuracy	LDA	n	k-Means	0.91568	0.91166	0.91985	0.93160
Binary Classif.	Accuracy	DecisionTreeClassifier	n	KernelPCA-poly3	0.91400	0.91016	0.91792	0.93514
Binary Classif.	Accuracy	DecisionTreeClassifier	n	KernelPCA-sigmoid	0.91272	0.90863	0.91692	0.93671
Binary Classif.	Accuracy	DecisionTreeClassifier	n	KernelPCA-rbf	0.90835	0.90464	0.91210	0.93065
Binary Classif.	Accuracy	DecisionTreeClassifier	n	k-Means	0.90671	0.90232	0.91128	0.92904
Binary Classif.	Accuracy	DecisionTreeClassifier	n	GaussRandProj	0.90379	0.89960	0.90802	0.92391
Binary Classif.	Accuracy	DecisionTreeClassifier	n	Isomap	0.90378	0.90032	0.90734	0.91139
Binary Classif.	Accuracy	DecisionTreeClassifier	n	LLE-standard	0.90368	0.89927	0.90813	0.92857
Binary Classif.	Accuracy	Logit Lasso	n	Isomap	0.89188	0.88666	0.89716	0.91043
Binary Classif.	Accuracy	Logit	n	Isomap	0.89108	0.88594	0.89633	0.91030
Binary Classif.	Accuracy	Logit	n	KernelPCA-sigmoid	0.88502	0.87922	0.89098	0.94171
Binary Classif.	Accuracy	NaiveBayesGaussian	n	GaussRandProj	0.88259	0.87898	0.88621	0.89538
Binary Classif.	Accuracy	QDA	n	Isomap	0.88140	0.87793	0.88490	0.88206
Binary Classif.	Accuracy	Logit Lasso	n	LLE-standard	0.88121	0.87674	0.88573	0.88798
Binary Classif.	Accuracy	Logit	n	LLE-standard	0.88110	0.87654	0.88569	0.88709
Binary Classif.	Accuracy	LDA	n	Isomap	0.88001	0.87373	0.88639	0.90881
Binary Classif.	Accuracy	LinearSVC	n	Isomap	0.87888	0.87338	0.88449	0.90490
Binary Classif.	Accuracy	NaiveBayesGaussian	n	Isomap	0.87874	0.87479	0.88266	0.88940
Binary Classif.	Accuracy	Logit Ridge	n	Isomap	0.87815	0.87284	0.88361	0.90424
Binary Classif.	Accuracy	LDA	n	LLE-standard	0.87681	0.87220	0.88156	0.88608
Binary Classif.	Accuracy	QDA	n	k-Means	0.86160	0.85524	0.86817	0.89771
Binary Classif.	Accuracy	Logit Ridge	n	LLE-standard	0.85776	0.85235	0.86315	0.88513
Binary Classif.	Accuracy	QDA	n	GaussRandProj	0.85500	0.84802	0.86182	0.90017
Binary Classif.	Accuracy	NaiveBayesGaussian	n	PCA	0.85055	0.84323	0.85797	0.89873
Binary Classif.	Accuracy	QDA	n	KernelPCA-poly3	0.84064	0.83345	0.84803	0.86969
Binary Classif.	Accuracy	LinearSVC	n	k-Means	0.83259	0.82410	0.84145	0.89873
Binary Classif.	Accuracy	QDA	n	PCA	0.80471	0.79383	0.81567	0.89019
Binary Classif.	Accuracy	NaiveBayesGaussian	n	k-Means	0.79628	0.78831	0.80447	0.85154
Binary Classif.	Accuracy	QDA	n	KernelPCA-cosine	0.78526	0.77708	0.79353	0.84193

Continued on next page

Problem Type	Metric	Model	Blackbox	Extractor	Mean	Mean CI 0.025	Mean CI 0.975	Median
Binary Classif.	Accuracy	NaiveBayesGaussian	n	KernelPCA-rbf	0.78164	0.77402	0.78930	0.85905
Binary Classif.	Accuracy	LDA	n	LLE-Modified	0.78161	0.77424	0.78904	0.81697
Binary Classif.	Accuracy	Logit Lasso	n	LLE-Modified	0.78149	0.77404	0.78892	0.81697
Binary Classif.	Accuracy	QDA	n	KernelPCA-sigmoid	0.77980	0.77200	0.78780	0.80966
Binary Classif.	Accuracy	Logit	n	LLE-Modified	0.77884	0.77128	0.78633	0.81657
Binary Classif.	Accuracy	Logit Ridge	n	LLE-Modified	0.77759	0.77005	0.78504	0.80966
Binary Classif.	Accuracy	NaiveBayesGaussian	n	KernelPCA-cosine	0.77696	0.76662	0.78750	0.87701
Binary Classif.	Accuracy	NaiveBayesGaussian	n	None	0.77547	0.76342	0.78780	0.88375
Binary Classif.	Accuracy	NaiveBayesGaussian	n	LLE-standard	0.77495	0.76801	0.78197	0.82151
Binary Classif.	Accuracy	DecisionTreeClassifier	n	LLE-Modified	0.77475	0.76775	0.78173	0.80000
Binary Classif.	Accuracy	LinearSVC	n	LLE-standard	0.77095	0.76153	0.78065	0.83195
Binary Classif.	Accuracy	NaiveBayesGaussian	n	KernelPCA-sigmoid	0.76864	0.76141	0.77574	0.82695
Binary Classif.	Accuracy	QDA	n	LLE-standard	0.74505	0.73567	0.75442	0.80074
Binary Classif.	Accuracy	NaiveBayesGaussian	n	KernelPCA-poly3	0.73603	0.72637	0.74581	0.78504
Binary Classif.	Accuracy	QDA	n	None	0.71652	0.70330	0.72971	0.78530
Binary Classif.	Accuracy	LinearSVC	n	LLE-Modified	0.70639	0.69597	0.71725	0.73182
Binary Classif.	Accuracy	LDA	n	KernelPCA-rbf	0.67642	0.66897	0.68401	0.63264
Binary Classif.	Accuracy	LDA	n	KernelPCA-poly3	0.62790	0.62066	0.63553	0.62047
Binary Classif.	Accuracy	NaiveBayesGaussian	n	LLE-Modified	0.60908	0.59941	0.61889	0.58667
Binary Classif.	Accuracy	LDA	n	KernelPCA-sigmoid	0.60224	0.59696	0.60758	0.58460
Binary Classif.	Accuracy	LDA	n	KernelPCA-cosine	0.59160	0.58467	0.59847	0.58378

Table A.5: All results for all algorithms and aggregations of normalized metrics. The mean of this normalized score is given, together with a bootstrapped 95% confidence interval. The rightmost column contains the median as well.

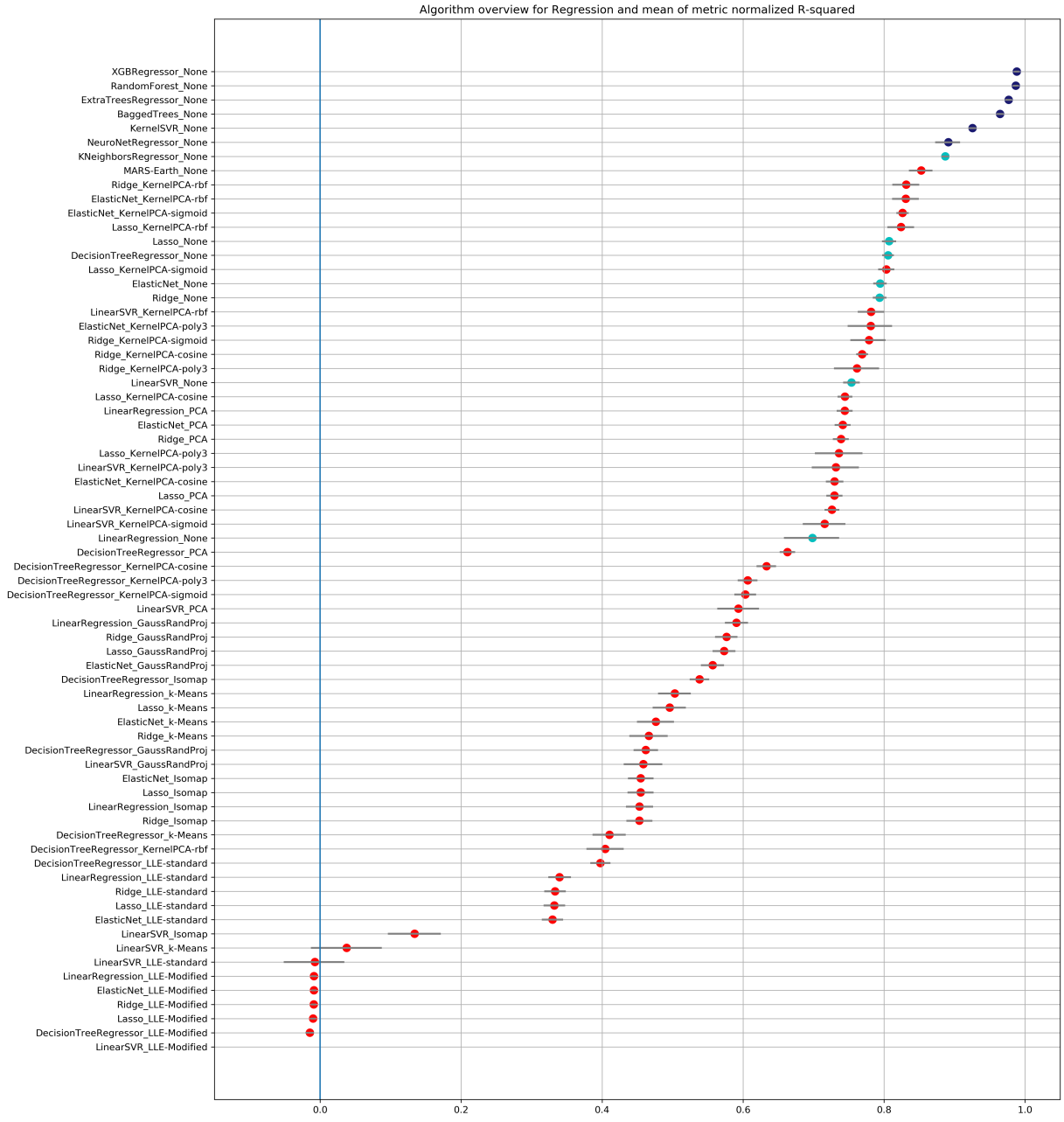

Figure A.1: Comparison of all algorithms for regression with regards to the mean normalized R-squared. Black box algorithms are displayed in dark blue. White box algorithms are shown in cyan if applied to the raw data, and in red if any extractor is used. Confidence intervals are shown with grey bars.

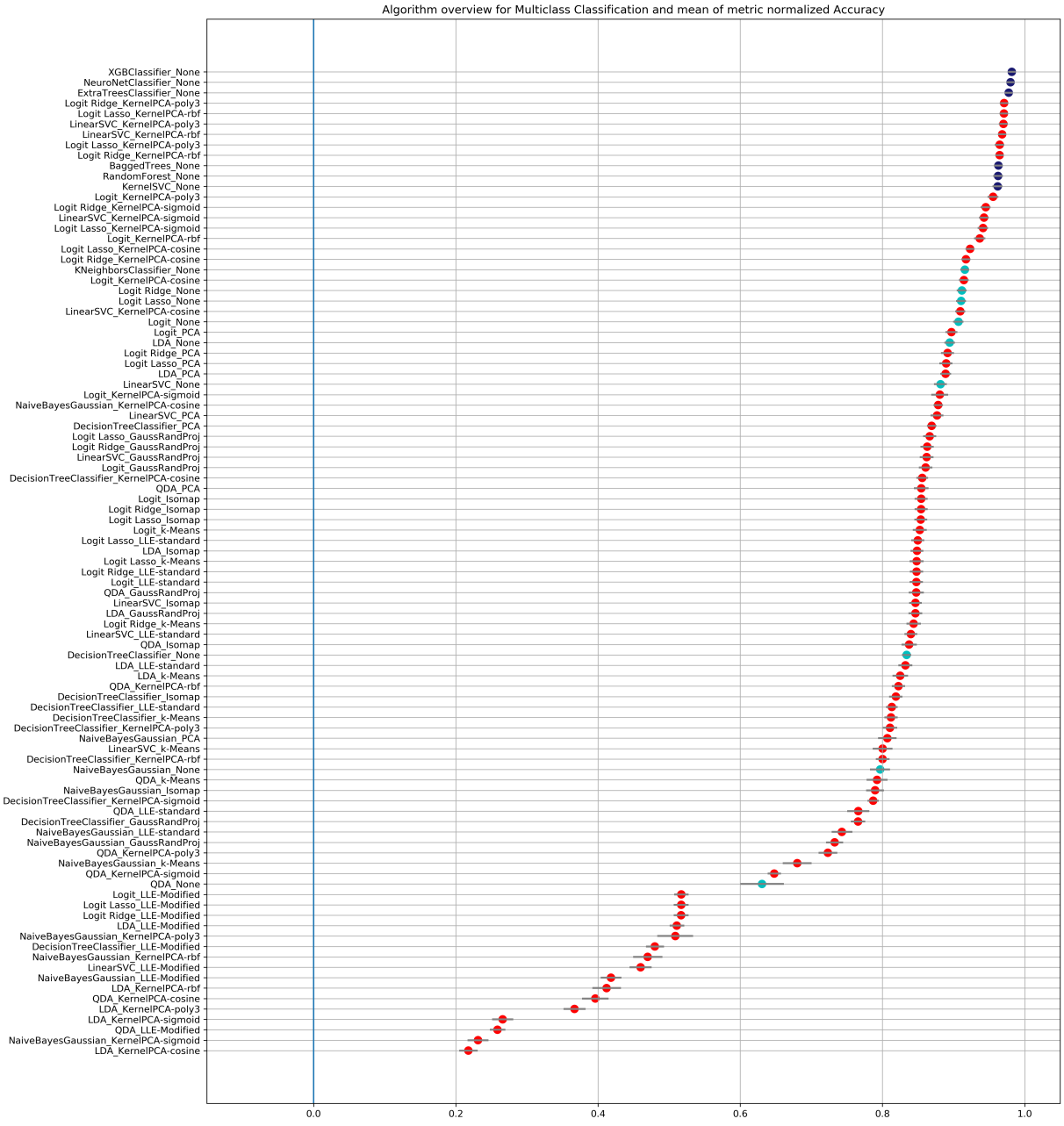

Figure A.2: Comparison of all algorithms for multiclass classification with regards to the mean normalized accuracy. Black box algorithms are displayed in dark blue. White box algorithms are shown in cyan if applied to the raw data, and in red if any extractor is used. Confidence intervals are shown with grey bars.

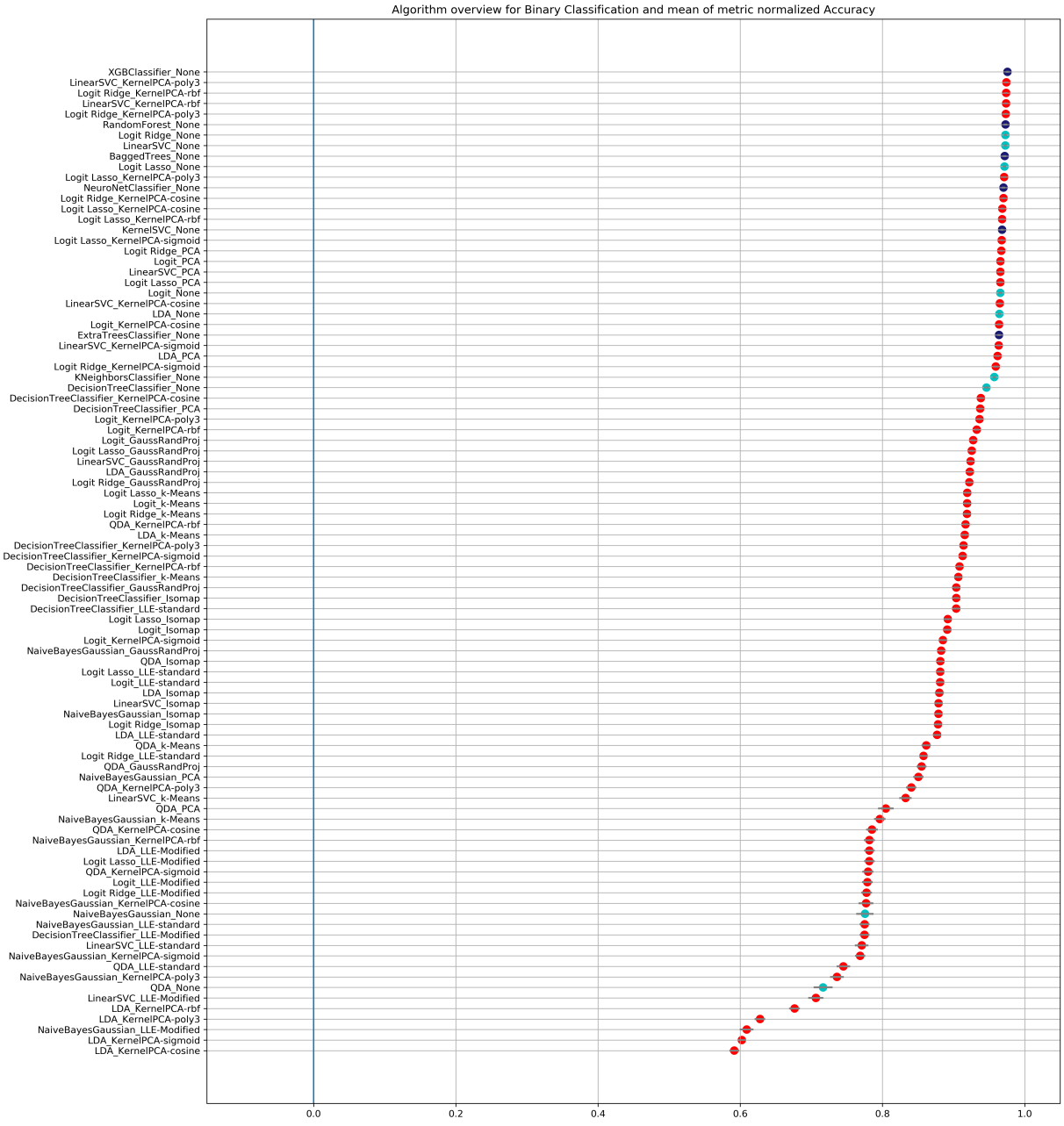

Figure A.3: Comparison of all algorithms for binary classification with regards to the mean normalized accuracy. Black box algorithms are displayed in dark blue. White box algorithms are shown in cyan if applied to the raw data, and in red if any extractor is used. Confidence intervals are shown with grey bars.

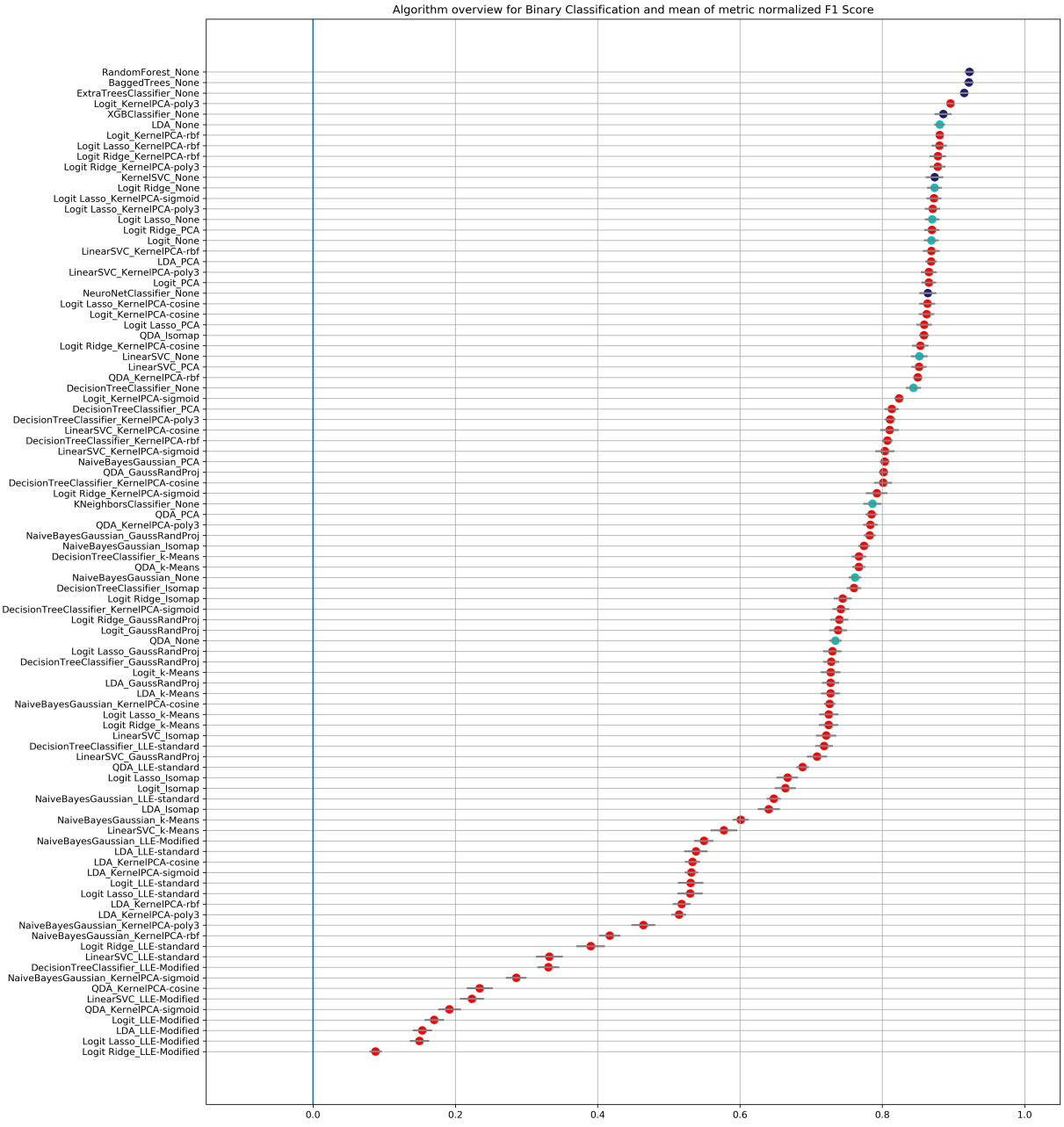

Figure A.4: Comparison of all algorithms for binary classification with regards to the mean normalized F1 score. Black box algorithms are displayed in dark blue. White box algorithms are shown in cyan if applied to the raw data, and in red if any extractor is used. Confidence intervals are shown with grey bars.

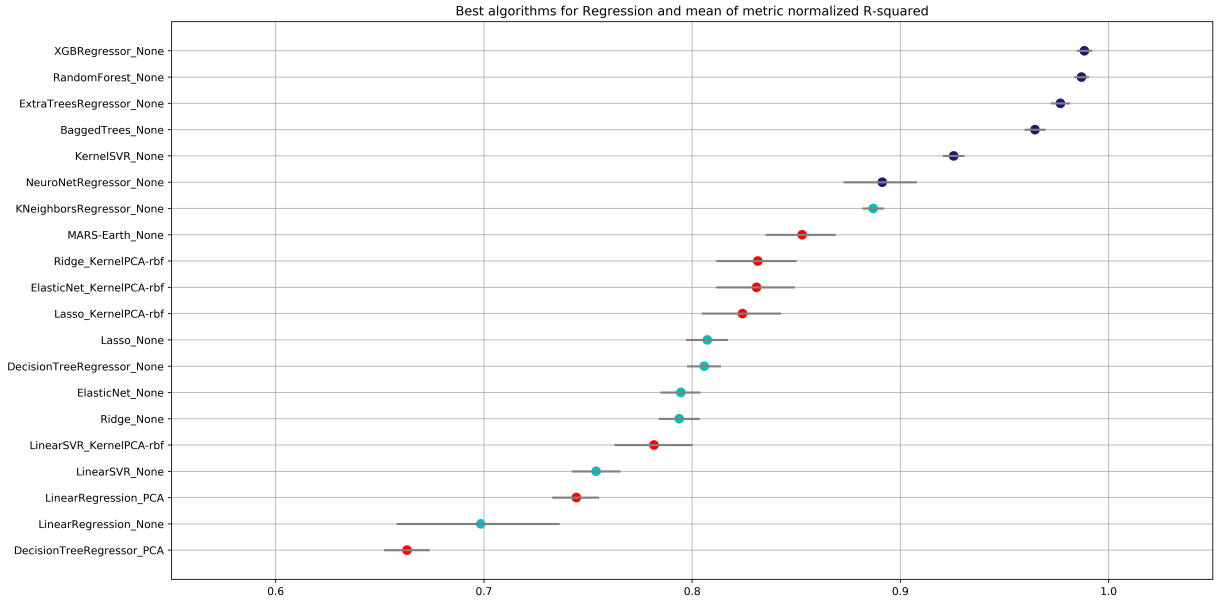

Figure A.5: Selection of the best regression algorithms with regards to the normalized R-squared. In this plot only the best white box extractor duo is shown together with the results without any extractors. Black box algorithms are displayed in dark blue. White box algorithms are shown in cyan if applied to the raw data, and in red if any extractor is used. Confidence intervals are shown with grey bars.

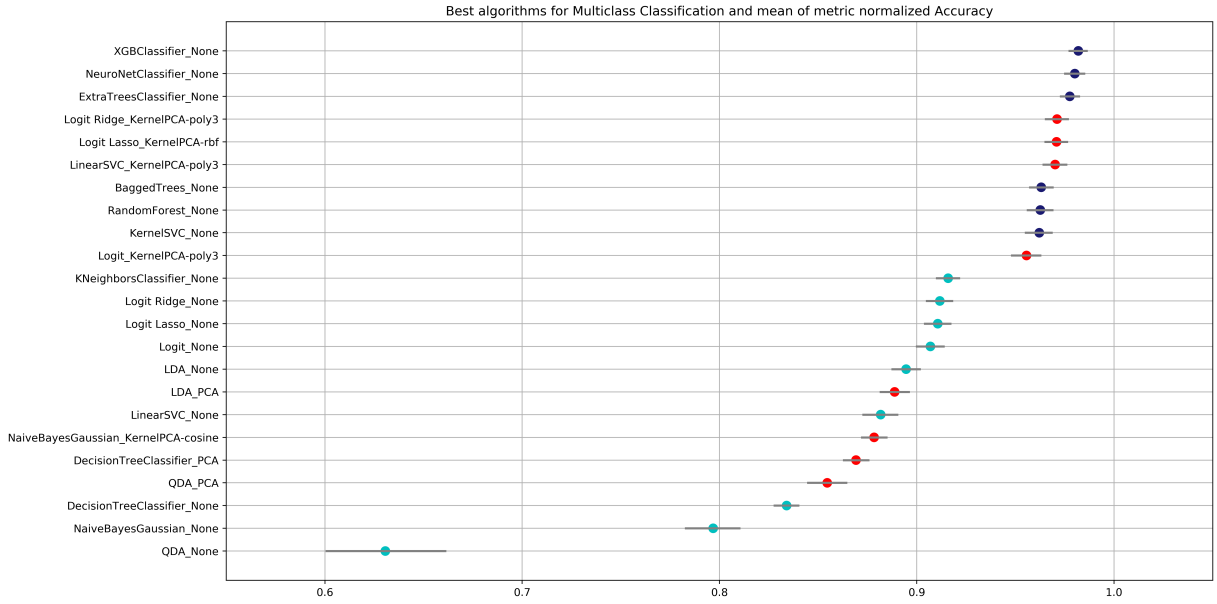

Figure A.6: Selection of the best multiclass classification algorithms with regards to the normalized accuracy. In this plot only the best white box extractor duo is shown together with the results without any extractors. Black box algorithms are displayed in dark blue. White box algorithms are shown in cyan if applied to the raw data, and in red if any extractor is used. Confidence intervals are shown with grey bars.

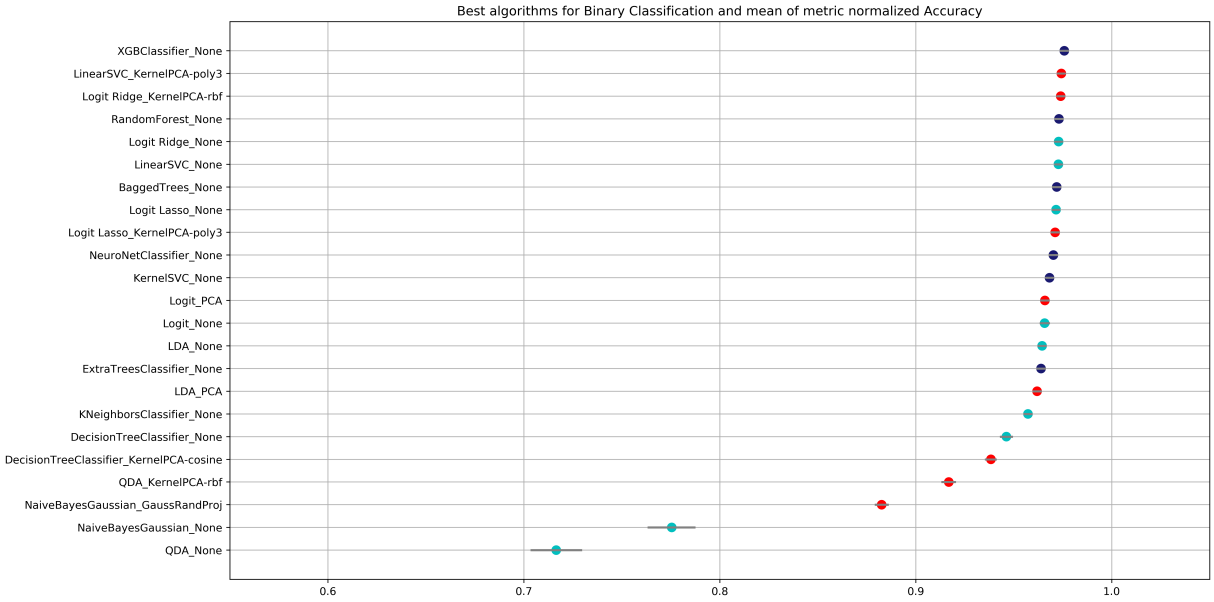

Figure A.7: Selection of the best binary classification algorithms with regards to the normalized accuracy. In this plot only the best white box extractor duo is shown together with the results without any extractors. Black box algorithms are displayed in dark blue. White box algorithms are shown in cyan if applied to the raw data, and in red if any extractor is used. Confidence intervals are shown with grey bars.

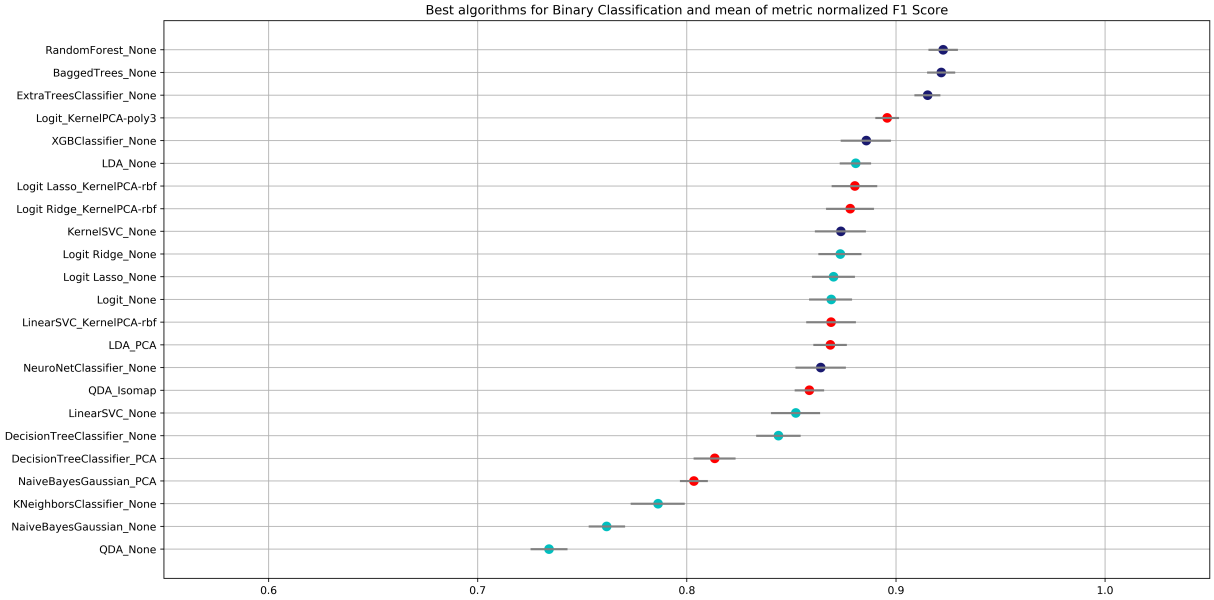

Figure A.8: Selection of the best binary classification algorithms with regards to the normalized F1 score. In this plot only the best white box extractor duo is shown together with the results without any extractors. Black box algorithms are displayed in dark blue. White box algorithms are shown in cyan if applied to the raw data, and in red if any extractor is used. Confidence intervals are shown with grey bars.

"Cheshire Puss," she began, rather timidly, as she did not at all know whether it would like the name: however, it only grinned a little wider. "Come, it's pleased so far," thought Alice, and she went on, "Would you tell me, please, which way I ought to go from here?"
"That depends a good deal on where you want to get to," said the Cat.
"I don't much care where——" said Alice.
"Then it doesn't matter which way you go," said the Cat.
"——so long as I get somewhere," Alice added as an explanation.
"Oh, you're sure to do that," said the Cat, "if you only walk long enough."

— Lewis Carroll – Alice's Adventures in Wonderland (1865)

B

Possible Extensions of this Thesis

Contents

B.1 Enhance the Extractors 75
B.2 Increase the Scope of the Analysis 75
B.3 Investigate the Monotony Hypothesis 76
B.4 Focus Studies 76

B.1 Enhance the Extractors

As we have seen the PCA methods were the most useful for feature extraction. For the kernel-PCA methods, instead of using default values for the kernel parameters, it might be worthwhile to include these parameters in the hyperparameter optimization. Also different extractors could be applied to numeric and categorical variables independently instead of applying a feature extraction to their concatenation. Besides this there might be more methods based on simple transformations which could be used as feature extractors. Potentially this could remedy the gap between white box and black box models for regression problems, as the exact linearity of the features is of higher importance than just extracting features for rank ordering, as is the case for the classification tasks.

B.2 Increase the Scope of the Analysis

An obvious extension would be to increase the number of datasets or increase the size of the datasets by not downsampling them by such a large amount. One reason for the downsampling was the high RAM requirement of the manifold methods.

Also for larger dataset the number of CV-repeats can likely be reduced as the results are more stable on larger datasets.

B.3 Investigate the Monotony Hypothesis

In our comparison of classification algorithms we only focused on the relative ordering of outcomes. In hindsight it might have been wise to include calibration for the classification problems as well and assess them on metrics which reward an appropriate assignment of probabilities (Niculescu-Mizil and Caruana 2005). We could then analyse if there is a structural difference of performance based on the type of metric, which would either confirm or falsify our monotony hypothesis from chapter 6.

Vice versa one could evaluate the regression predictions with ranking metrics and assess whether the performance gap between white and black box methods would shrink and behave similarly to the classification results.

B.4 Focus Studies

As we saw in this work the boosting algorithm xgboost outperformed most other methods on all three problem types. It might be interesting to perform a comparison between different boosting algorithms. Such analyses were previously done in the literature (Bauer and Kohavi 1999; McDonald, Hand, and Eckley 2003) but could be extended with new available boosting algorithms and variants. Additionally the feature extractions could be added to the boosting algorithms as well and one could analyse if and how their performance changes with the altered features. Such a benchmarking study might be of special interest for people competing in machine learning competitions.

The best thing about being a statistician is that you get to play in everyone's backyard.

— John Tukey

C Details on Datasets and Longlist

Contents

C.1 Notes on Specific Datasets 77
C.2 Longlist 79
C.3 Distributions for Hyperparameter Optimization 84

C.1 Notes on Specific Datasets

In this section we provide further notes on certain datasets. This chapter serves the purpose of making our results transparent and reproducible. The longlist, presented in table C.1, contains all variables per dataset and the type of each variable used by the algorithms.

Note that all datasets with more than 10,000 observations are downsampled by selecting 10,000 observations at random as described in the main text (see chapter 4). For some highly unbalanced datasets the downsampling was done by case sampling, as described in the dataset-specific sections below.

AirScaniaTrucks: As the dataset is highly unbalanced we selected the 1375 observations which had air pressure system failures and added 8625 randomly selected observations without failures, yielding 10,000 observations in our dataset.

AmesHousing: As the target variable was highly skewed we transformed it with the (natural) logarithm.

Automobile: We removed the 4 rows which had no price target from the dataset.

BankMarketingUCI: As the dataset is highly unbalanced we selected the 5289 observations with positive marketing outcome and added 19711 randomly selected observations with negative outcome, yielding a total of 25,000 observations in our initial dataset. From this dataset the 10,000 random observations were drawn.

CaliforniaHousing: As already noted by others, it appears that the house values were winsorized at half a million dollars. We hypothesise that this was done to reduce the skewness of the distribution to a more appropriate range, but could not find any information on this procedure in the documents attached to the data.

CaravanInsurance: Many columns just indicate the range of the decimal of the percentile of the original variable. We treat them like normal metric variables, see our longlist in table C.1.

CreditCardFrauds: This dataset from Kaggle[1] is highly unbalanced. We take all 492 entries with a default flag and add 9,000 randomly selected observations without default.

HorseColic: There was an additional test set available containing target labels for the horses. We concatenated it with the original training set to increase the number of observations. We removed the two horses with unknown target and mapped the targets "dead" and "euthanized" to the same category, as both indicate that a horse is problematic. In this way the dataset becomes a binary classification problem.

KingCountyHousing: As the target variable was highly skewed we transformed it with the (natural) logarithm. Note that we did not allow the algorithms to use the zip-code and timestamp column, as can be seen in the longlist table C.1.

MelbourneHousing: The data, which can be found on the Kaggle website[2], was originally scraped from the web. Fearing any erroneously scrapped values we removed any observations which had a missing value in one of the variables of the longlist or the target.

PortoSeguro: In order to balance the dataset we randomly selected 5,000 positive and 5,000 negative cases from the original data.

SberbankRussianHousingMarket: This Kaggle dataset's price was transformed with a natural logarithm to remove skewness. We want to note that in the original data there is extensive information available on the neighbourhoods of a house. In this study we only used the basic properties, as can be seen in the long list table C.1. Also a file containing corrections was posted by the competition hosts to fix some faulty entries. We applied these corrections also to our dataset.

[1] https://www.kaggle.com/isaikumar/creditcardfraud
[2] https://www.kaggle.com/anthonypino/melbourne-housing-market

WineQuality: Originally the information for white and red wines was in two separate csv-files. We concatenated the two and created an additional column which contains the wine colour "red" or "white".

C.2 Longlist

In the following section we present the exact longlist used for each dataset. While being rather elaborate we find it necessary to include this list to facilitate the exact reproduction of our results.

Problem Type	Dataset	Feature Type	Feature List
Binary Classification	AdultHighEarnerUCI	Categorical	{education, marital-status, native-country, occupation, race, relationship, sex, workclass}
Binary Classification	AdultHighEarnerUCI	Metric	{age, capital-gain, capital-loss, education-num, fnlwgt, hours-per-week}
Binary Classification	AirScaniaTrucks	Metric	{aa_000, ab_000, ac_000, ad_000, ae_000, af_000, ag_000, ag_001, ag_002, ag_003, ag_004, ag_005, ag_006, ag_007, ag_008, ag_009, ah_000, ai_000, aj_000, ak_000, al_000, am_0, an_000, ao_000, ap_000, aq_000, ar_000, as_000, at_000, au_000, av_000, ax_000, ay_000, ay_001, ay_002, ay_003, ay_004, ay_005, ay_006, ay_007, ay_008, ay_009, az_000, az_001, az_002, az_003, az_004, az_005, az_006, az_007, az_008, az_009, ba_000, ba_001, ba_002, ba_003, ba_004, ba_005, ba_006, ba_007, ba_008, ba_009, bb_000, bc_000, bd_000, be_000, bf_000, bg_000, bh_000, bi_000, bj_000, bk_000, bl_000, bm_000, bn_000, bo_000, bp_000, bq_000, br_000, bs_000, bt_000, bu_000, bv_000, bx_000, by_000, bz_000, ca_000, cb_000, cc_000, cd_000, ce_000, cf_000, cg_000, ch_000, ci_000, cj_000, ck_000, cl_000, cm_000, cn_000, cn_001, cn_002, cn_003, cn_004, cn_005, cn_006, cn_007, cn_008, cn_009, co_000, cp_000, cq_000, cr_000, cs_000, cs_001, cs_002, cs_003, cs_004, cs_005, cs_006, cs_007, cs_008, cs_009, ct_000, cu_000, cv_000, cx_000, cy_000, cz_000, da_000, db_000, dc_000, dd_000, de_000, df_000, dg_000, dh_000, di_000, dj_000, dk_000, dl_000, dm_000, dn_000, do_000, dp_000, dq_000, dr_000, ds_000, dt_000, du_000, dv_000, dx_000, dy_000, dz_000, ea_000, eb_000, ec_00, ed_000, ee_000, ee_001, ee_002, ee_003, ee_004, ee_005, ee_006, ee_007, ee_008, ee_009, ef_000, eg_000}
Binary Classification	AustralianCredit	Categorical	{A1, A11, A12, A4, A5, A6, A8, A9}
Binary Classification	AustralianCredit	Metric	{A10, A13, A14, A2, A3, A7}
Binary Classification	BankMarketingUCI	Categorical	{contact, default, education, housing, job, loan, marital, month, poutcome}
Binary Classification	BankMarketingUCI	Metric	{age, balance, campaign, day, duration, pdays, previous}
Binary Classification	BreastCancerWisconsinDiagnostic	Metric	{Area_mean, Area_std, Area_worst, Compactness_mean, Compactness_std, Compactness_worst, ConcavePoints_mean, ConcavePoints_std, ConcavePoints_worst, Concavity_mean, Concavity_std, Concavity_worst, FractalDim_mean, FractalDim_std, FractalDim_worst, Perimeter_mean, Perimeter_std, Perimeter_worst, Radius_mean, Radius_std, Radius_worst, Smoothness_mean, Smoothness_std, Smoothness_worst, Symmetry_mean, Symmetry_std, Symmetry_worst, Texture_mean, Texture_std, Texture_worst}
Binary Classification	CaravanInsurance	Categorical	{MOSTYPE}

Continued on next page

Problem Type	Dataset	Feature Type	Feature List
Binary Classification	CaravanInsurance	Metric	{AAANHANG, ABESAUT, ABRAND, ABROM, ABYSTAND, AFIETS, AGEZONG, AINBOED, ALEVEN, AMOTSCO, APERSAUT, APERSONG, APLEZIER, ATRACTOR, AVRAAUT, AWABEDR, AWALAND, AWAOREG, AWAPART, AWERKT, AZEILPL, MAANTHUI, MAUT0, MAUT1, MAUT2, MBERARBG, MBERARBO, MBERBOER, MBERHOOG, MBERMIDD, MBERZELF, MFALLEEN, MFGEKIND, MFWEKIND, MGEMLEEF, MGEMOMV, MGODGE, MGODOV, MGODPR, MGODRK, MHHUUR, MHKOOP, MINK123M, MINK3045, MINK4575, MINK7512, MINKGEM, MINKM30, MKOOPKLA, MOPLHOOG, MOPLLAAG, MOPLMIDD, MOSHOOFD, MRELGE, MRELOV, MRELSA, MSKA, MSKB1, MSKB2, MSKC, MSKD, MZFONDS, MZPART, PAANHANG, PBESAUT, PBRAND, PBROM, PBYSTAND, PFIETS, PGEZONG, PINBOED, PLEVEN, PMOTSCO, PPERSAUT, PPERSONG, PPLEZIER, PTRACTOR, PVRAAUT, PWABEDR, PWALAND, PWAOREG, PWAPART, PWERKT, PZEILPL}
Binary Classification	ChessKRKPA7	Categorical	{BP1, BP10, BP11, BP12, BP13, BP14, BP15, BP16, BP17, BP18, BP19, BP2, BP20, BP21, BP22, BP23, BP24, BP25, BP26, BP27, BP28, BP29, BP3, BP30, BP31, BP32, BP33, BP34, BP35, BP36, BP4, BP5, BP6, BP7, BP8, BP9}
Binary Classification	CreditCardFraud	Metric	{Amount, V1, V10, V11, V12, V13, V14, V15, V16, V17, V18, V19, V2, V20, V21, V22, V23, V24, V25, V26, V27, V28, V3, V4, V5, V6, V7, V8, V9}
Binary Classification	GermanCredit	Categorical	{CheckingAccountStatus, CreditHistory, EmploymentDuration, ForeignWorker, Housing, Job, NumberPeopleMaintenance, OtherDebtorsGuarantors, OtherInstallmentPlans, Personal_Sex, Property, Purpose, SavingsAccountBonds, Telephone}
Binary Classification	GermanCredit	Metric	{Age, Amount, Duration, InstallmentRatePercentage, NumberExistingCredits, ResidenceDuration}
Binary Classification	HeartStatlog	Categorical	{chest_pain, exercise_induced_angina, fasting_blood_sugar, rest_electrocar_res, sex, slope, thal}
Binary Classification	HeartStatlog	Metric	{age, max_heart_rate, num_major_vessels, oldpeak, rest_blood_pressure, serum_cholestoral}
Binary Classification	HorseColic	Categorical	{Age, abdomen, abdominal_distension, abdominocentesis_appearance, capillary_refill_time, mucous_membranes, nasogastric_reflux, nasogastric_tube, pain, peripheral_pulse, peristalsis, rectal_examination, surgery, temperature_extremities}
Binary Classification	HorseColic	Metric	{abdomcentesis_total_protein, nasogastric_reflux_PH, packed_cell_volume, pulse, rectal_temperature, respiratory_rate, total_protein}
Binary Classification	KobeBryantShots	Categorical	{action_type, combined_shot_type, period, playoffs, shot_type, shot_zone_area, team_id, team_name}
Binary Classification	KobeBryantShots	Metric	{game_date, lat, loc_x, loc_y, lon, minutes_remaining, seconds_remaining, shot_distance}
Binary Classification	Mushroom	Categorical	{bruises, cap_color , cap_shape , cap_surface , gill_attachment , gill_color , gill_size , gill_spacing , habitat , odor , population , ring_number , ring_type , spore_print_color , stalk_color_above_ring , stalk_color_below_ring , stalk_root , stalk_shape , stalk_surface_above_ring , stalk_surface_below_ring , veil_color , veil_type}

Continued on next page

Problem Type	Dataset	Feature Type	Feature List
Binary Classification	PortoSeguro	Categorical	{ps_calc_15_bin, ps_calc_16_bin, ps_calc_17_bin, ps_calc_18_bin, ps_calc_19_bin, ps_calc_20_bin, ps_car_01_cat, ps_car_02_cat, ps_car_03_cat, ps_car_04_cat, ps_car_05_cat, ps_car_06_cat, ps_car_07_cat, ps_car_08_cat, ps_car_09_cat, ps_car_10_cat, ps_car_11_cat, ps_ind_02_cat, ps_ind_04_cat, ps_ind_05_cat, ps_ind_06_bin, ps_ind_07_bin, ps_ind_08_bin, ps_ind_09_bin, ps_ind_10_bin, ps_ind_11_bin, ps_ind_12_bin, ps_ind_13_bin, ps_ind_16_bin, ps_ind_17_bin, ps_ind_18_bin}
Binary Classification	PortoSeguro	Metric	{ps_calc_01, ps_calc_02, ps_calc_03, ps_calc_04, ps_calc_05, ps_calc_06, ps_calc_07, ps_calc_08, ps_calc_09, ps_calc_10, ps_calc_11, ps_calc_12, ps_calc_13, ps_calc_14, ps_car_11, ps_car_12, ps_car_13, ps_car_14, ps_car_15, ps_ind_01, ps_ind_03, ps_ind_14, ps_ind_15, ps_reg_01, ps_reg_02, ps_reg_03}
Binary Classification	TaiwanCreditCardDefault	Categorical	{EDUCATION, MARRIAGE, SEX}
Binary Classification	TaiwanCreditCardDefault	Metric	{AGE, BILL_AMT1, BILL_AMT2, BILL_AMT3, BILL_AMT4, BILL_AMT5, BILL_AMT6, LIMIT_BAL, PAY_0, PAY_2, PAY_3, PAY_4, PAY_5, PAY_6, PAY_AMT1, PAY_AMT2, PAY_AMT3, PAY_AMT4, PAY_AMT5, PAY_AMT6}
Binary Classification	TitanicKaggle	Categorical	{Cabin, Embarked, Pclass, Sex}
Binary Classification	TitanicKaggle	Metric	{Age, Fare, Parch, SibSp, Ticket}
Multiclass Classification	BalanceScale	Metric	{Left_Distance, Left_Weight, Right_Distance, Right_Weight}
Multiclass Classification	CarEvaluation	Categorical	{buying, doors, lug_boot, maint, persons, safety}
Multiclass Classification	Glass	Metric	{Al, Ba, Ca, Fe, K, Mg, Na, RI, Si}
Multiclass Classification	OttoGroupProductClassChallenge	Metric	{feat_1, feat_10, feat_11, feat_12, feat_13, feat_14, feat_15, feat_16, feat_17, feat_18, feat_19, feat_2, feat_20, feat_21, feat_22, feat_23, feat_24, feat_25, feat_26, feat_27, feat_28, feat_29, feat_3, feat_30, feat_31, feat_32, feat_33, feat_34, feat_35, feat_36, feat_37, feat_38, feat_39, feat_4, feat_40, feat_41, feat_42, feat_43, feat_44, feat_45, feat_46, feat_47, feat_48, feat_49, feat_5, feat_50, feat_51, feat_52, feat_53, feat_54, feat_55, feat_56, feat_57, feat_58, feat_59, feat_6, feat_60, feat_61, feat_62, feat_63, feat_64, feat_65, feat_66, feat_67, feat_68, feat_69, feat_7, feat_70, feat_71, feat_72, feat_73, feat_74, feat_75, feat_76, feat_77, feat_78, feat_79, feat_8, feat_80, feat_81, feat_82, feat_83, feat_84, feat_85, feat_86, feat_87, feat_88, feat_89, feat_9, feat_90, feat_91, feat_92, feat_93}
Multiclass Classification	Wine	Metric	{Alcalinity_Ash, Alcohol, Ash, Color_intensity, Flavanoids, Hue, Magnesium, Malic_Acid, Nonflavanoid_phenols, OD_Quotient_dilution, Proanthocyanins, Proline, Total_phenols}
Regression	Abalone	Categorical	{Sex}
Regression	Abalone	Metric	{Diameter, Height, Length, Shell_weight, Shucked_weight, Viscera_weight, Whole_weight}

Continued on next page

Problem Type	Dataset	Feature Type	Feature List
Regression	AmesHousing	Categorical	{Alley, BedroomAbvGr, BldgType, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, BsmtFullBath, BsmtHalfBath, BsmtQual, CentralAir, Condition1, Condition2, Electrical, ExterCond, ExterQual, Exterior1st, Exterior2nd, Fence, FireplaceQu, Foundation, FullBath, Functional, GarageCond, GarageFinish, GarageQual, GarageType, HalfBath, Heating, HeatingQC, HouseStyle, KitchenAbvGr, KitchenQual, LandContour, LandSlope, LotConfig, LotShape, MSSubClass, MSZoning, MasVnrType, MiscFeature, Neighborhood, OverallCond, OverallQual, PavedDrive, PoolQC, RoofMatl, RoofStyle, SaleCondition, SaleType, Street, Utilities}
Regression	AmesHousing	Metric	{BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, EnclosedPorch, Fireplaces, GarageArea, GarageCars, GarageYrBlt, GrLivArea, LotArea, LotFrontage, LowQualFinSF, MasVnrArea, MiscVal, MoSold, OpenPorchSF, PoolArea, ScreenPorch, TotRmsAbvGrd, TotalBsmtSF, WoodDeckSF, X1stFlrSF, X2ndFlrSF, X3SsnPorch, YearBuilt, YearRemodAdd, YrSold}
Regression	AutoMPG	Categorical	{cylinders , origin }
Regression	AutoMPG	Metric	{acceleration, displacement, horsepower , model_year , weight }
Regression	Automobile	Categorical	{aspiration , body_style , drive_wheels , engine_location , engine_type , fuel_system , fuel_type , num_of_cylinders , num_of_doors }
Regression	Automobile	Metric	{bore , city_mpg , compression_ratio , curb_weight , engine_size , height , highway_mpg , horsepower , length , normalized_losses , peak_rpm , stroke , symboling , wheel_base , width }
Regression	BostonHousing	Categorical	{chas}
Regression	BostonHousing	Metric	{age , b , crim , dis , indus , lstat , nox , ptratio , rad , rm , tax , zn}
Regression	BostonHousing2	Categorical	{chas , town, tract}
Regression	BostonHousing2	Metric	{age , b , crim , dis , indus , lat, lon, lstat , nox , ptratio , rad , rm , tax , zn}
Regression	CaliforniaHousing	Metric	{households, housingMedianAge, latitude, longitude, medianIncome, population, totalBedrooms, totalRooms}
Regression	ConcreteCompressiveStrength	Metric	{Age, Blast_Furnance_Slag, Coarse_Aggregate, Comp_Cement, Fine_Aggregate, Fly_Ash, Superplasticizer, Water}
Regression	KingCountyHousing	Categorical	{waterfront}
Regression	KingCountyHousing	Metric	{bathrooms, bedrooms, condition, floors, grade, lat, long, sqft_above, sqft_basement, sqft_living, sqft_living15, sqft_lot, sqft_lot15, view, yr_built, yr_renovated}
Regression	MelbourneHousing	Categorical	{Method, Regionname, Type}
Regression	MelbourneHousing	Metric	{Bathroom, Bedroom2 , BuildingArea, Car, Distance, Landsize, Rooms, YearBuilt}
Regression	SberbankRussianHousingMarket	Categorical	{material, product_type, state, sub_area}
Regression	SberbankRussianHousingMarket	Metric	{build_year, floor, full_sq, kitch_sq, life_sq, max_floor, num_room}
Regression	WineQuality	Categorical	{wine_color}
Regression	WineQuality	Metric	{alcohol, chlorides, citric_acid, density, fixed_acidity, free_sulfur_dioxide, pH , residual_sugar, sulphates, total_sulfur_dioxide, volatile_acidity}

Table C.1: The Longlist with the variables for all datasets by feature type.

C.3 Distributions for Hyperparameter Optimization

As we use a random search we have to define distributions for each hyperparameter from which a random value is drawn, as described in chapter 4 of the main text. The distributions chosen by use are summarized in table C.2. A random draw from a given set is always used with replacement; thus in the next draw of hyperparameters the same value can be obtained.

The `exp10_uni` indicates the exponentiation of a univariate distribution. If U is uniformly distributed in [a,b) then it is given by $X = exp_{10}(U) = 10^U$. Note that we used 10 as a basis to facilitate the comparison with grid-searches in other publications.

The density of a random variable with this distribution can easily be calculated using the cumulative distribution function (CDF):

$$F_X(x) = \mathrm{P}(X \leq x) = \mathrm{P}(10^U \leq x) \tag{C.1}$$

We know that a logarithm with base $r > 1$ is a monotone function; thus if $x > y$ then $\log_r(x) > \log_r(y)$. Applying this to the CDF, noting that by its definition $x > 0$:

$$F_X(x) = \mathrm{P}(10^U \leq x) = \mathrm{P}(U \leq \log_{10}(x)) \tag{C.2}$$

Thus we can infer the shape of the probability density function:

$$f_X(x) = \frac{d}{dx} F_X(x) \propto \frac{d}{dx} \log_{10}(x) = \frac{1}{x \ln(10)} \tag{C.3}$$

Model	Parameter	Distribution	Comment
Ridge Regression	Penalty	exp10_uni(minExp = -7.0, maxExp=+5.0)	minExp and maxExp refer to the boundaries of the univariate distribution, thus indicating the minimum and maximum exponent of 10.
Lasso	Penalty	exp10_uni(minExp = -7.0, maxExp=+5.0)	
Logistic Ridge	Penalty	Random draw from np.logspace(-5, 9, 16)	
Logistic Lasso	Penalty	Random draw from np.logspace(-5, 9, 16)	
Elastic Net	Total Penalty	exp10_uni(minExp = -7.0, maxExp=+5.0)	
Elastic Net	Fraction of l1 penalty	Uniform (floating point) in [1%; 99%]	
MARS	Max terms	Uniform integer from [1; 500)	
MARS	Max degree	Random draw from {1, 2, 3}	
MARS	Penalty	exp10_uni(min_exp=-4.0, max_exp=3.0)	
Linear SVR C	C parameter	exp10_uni(min_exp=-9.0, max_exp=5.0)	
Kernel SVR C	C parameter	exp10_uni(min_exp=-9.0, max_exp=5.0)	
Kernel SVR C	Kernel	Random draw from {'radial basis function', 'sigmoid'}	Standard values are used for each kernel
KNN	Nr Neighbours	Uniform integer from [1, 500*)	* For small datasets we reduce this maximum to half the number of all observations (.5 rounded upwards)
KNN	Weight Method	Random draw from {'uniform', 'distance'}	For uniform all neighbours are treated equally while for the distance methods the neighbours are weighted by their distance to the point to be evaluated.
Decision Tree	Max Depth	Random integer in [1, 100]	
Random Forest	Fraction of variables for each split	Uniform (floating point) in [0%; 100%]	
Extra Trees	Fraction of variables for each split	Uniform (floating point) in [0%; 100%]	
XGBoost	Learning rate	Draw from {0.5, 0.3, 0.1, 0.05, 0.01, 0.005, 0.001}	
	Max Depth	Draw from {1, 2, 3, 4, 5, 6}	
	Nr Trees	Uniform integer in [50, 1000]	
Neural Network	Structure	Random draw from [(2,), (4,), (8,), (16,), (32,), (64,), (128,), (256,), (512,), (2,2), (4,4), (8,8), (16,16), (32,32), (64,64), (128,128), (256,256), (512,512)]	Where (x, y) corresponds to the number of neurons in the first and second hidden layer of the network. (x,) denotes a network with just one hidden layer.
	Alpha penalty	exp10_uni(min_exp=-5.0, max_exp=2.0)	

Table C.2: Distributions used for the hyperparameter optimization

References

Bauer, Eric and Ron Kohavi (1999). "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants". In: *Machine learning* 36.1-2, pp. 105–139.

Bengio, Yoshua and Yves Grandvalet (2004). "No unbiased estimator of the variance of k-fold cross-validation". In: *Journal of machine learning research* 5.Sep, pp. 1089–1105.

Bergstra, James and Yoshua Bengio (2012). "Random search for hyper-parameter optimization". In: *Journal of Machine Learning Research* 13.Feb, pp. 281–305.

Bingham, Ella and Heikki Mannila (2001). "Random projection in dimensionality reduction: applications to image and text data". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 245–250.

Bischl, Bernd, Julia Schiffner, and Claus Weihs (2014). "Benchmarking classification algorithms on high-performance computing clusters". In: *Data Analysis, Machine Learning and Knowledge Discovery*. Springer, pp. 23–31.

Boser, Bernhard E, Isabelle M Guyon, and Vladimir N Vapnik (1992). "A training algorithm for optimal margin classifiers". In: *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, pp. 144–152.

Box, George EP (1976). "Science and statistics". In: *Journal of the American Statistical Association* 71.356, pp. 791–799.

Brand, Matthew (2003). "Charting a manifold". In: *Advances in neural information processing systems*, pp. 985–992.

Breiman, L. et al. (1984). *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis.

Breiman, Leo (1996a). "Bagging predictors". In: *Machine learning* 24.2, pp. 123–140.

Breiman, Leo (1996b). *Out-of-bag estimation*.

Breiman, Leo (2001). "Random forests". In: *Machine learning* 45.1, pp. 5–32.

Breiman, Leo et al. (2001). "Statistical modeling: The two cultures (with comments and a rejoinder by the author)". In: *Statistical science* 16.3, pp. 199–231.

Breiman, Leo and Philip Spector (1992). "Submodel selection and evaluation in regression. The X-random case". In: *International statistical review/revue internationale de Statistique*, pp. 291–319.

Caruana, Rich, Nikos Karampatziakis, and Ainur Yessenalina (2008). "An empirical evaluation of supervised learning in high dimensions". In: *Proceedings of the 25th international conference on Machine learning*. ACM, pp. 96–103.

Caruana, Rich and Alexandru Niculescu-Mizil (2006). "An empirical comparison of supervised learning algorithms". In: *Proceedings of the 23rd international conference on Machine learning*. ACM, pp. 161–168.

Cawley, Gavin C and Nicola LC Talbot (2010). "On over-fitting in model selection and subsequent selection bias in performance evaluation". In: *Journal of Machine Learning Research* 11.Jul, pp. 2079–2107.

Chen, Tianqi and Carlos Guestrin (2016). "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, pp. 785–794.

Chen, Tianqi and Tong He (2015). "Higgs boson discovery with boosted trees". In: *NIPS 2014 Workshop on High-energy Physics and Machine Learning*, pp. 69–80.

Claesen, Marc and Bart De Moor (2015). "Hyperparameter search in machine learning". In: *arXiv preprint arXiv:1502.02127*.

Coates, Adam and Andrew Y Ng (2012). "Learning feature representations with k-means". In: *Neural networks: Tricks of the trade*. Springer, pp. 561–580.

Dasgupta, Sanjoy (2013). "Experiments with random projection". In: *arXiv preprint arXiv:1301.3849*.

De Cock, Dean (2011). "Ames, Iowa: Alternative to the Boston housing data as an end of semester regression project". In: *Journal of Statistics Education* 19.3.

Dheeru, Dua and Efi Karra Taniskidou (2017). *UCI Machine Learning Repository*.

Domingos, Pedro (2000). "A unified bias-variance decomposition". In: *Proceedings of 17th International Conference on Machine Learning*, pp. 231–238.

Drucker, Harris et al. (1997). "Support vector regression machines". In: *Advances in neural information processing systems*, pp. 155–161.

Fisher, Ronald A (1936). "The use of multiple measurements in taxonomic problems". In: *Annals of eugenics* 7.2, pp. 179–188.

Freund, Yoav and Robert E Schapire (1997). "A decision-theoretic generalization of on-line learning and an application to boosting". In: *Journal of computer and system sciences* 55.1, pp. 119–139.

Friedman, Jerome H (1991). “Multivariate adaptive regression splines”. In: *The annals of statistics*, pp. 1–67.

Friedman, Jerome H (2001). “Greedy function approximation: a gradient boosting machine”. In: *Annals of statistics*, pp. 1189–1232.

Friedman, Jerome H (2002). “Stochastic gradient boosting”. In: *Computational Statistics & Data Analysis* 38.4, pp. 367–378.

Géron, Aurélien (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O’Reilly Media, Inc."

Geurts, Pierre, Damien Ernst, and Louis Wehenkel (2006). “Extremely randomized trees”. In: *Machine learning* 63.1, pp. 3–42.

Goodfellow, Ian et al. (2016). *Deep learning*. Vol. 1. MIT press Cambridge.

Hastie, T., R. Tibshirani, and J. Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics. Springer New York.

Henking, Andreas, Christian Bluhm, and Ludwig Fahrmeir (2006). *Kreditrisikomessung: Statistische Grundlagen, Methoden und Modellierung*. Springer-Verlag.

James, Gareth et al. (2013). *An introduction to statistical learning*. Vol. 112. Springer.

Kim, Ji-Hyun (2009). “Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap”. In: *Computational statistics & data analysis* 53.11, pp. 3735–3745.

King, Ross D., Cao Feng, and Alistair Sutherland (1995). “Statlog: comparison of classification algorithms on large real-world problems”. In: *Applied Artificial Intelligence an International Journal* 9.3, pp. 289–333.

Kohavi, Ron et al. (1995). “A study of cross-validation and bootstrap for accuracy estimation and model selection”. In: *Ijcai*. Vol. 14. 2. Montreal, Canada, pp. 1137–1145.

Krstajic, Damjan et al. (2014). “Cross-validation pitfalls when selecting and assessing regression and classification models”. In: *Journal of cheminformatics* 6.1, p. 10.

Kuhn, Max and Kjell Johnson (2013). *Applied predictive modeling*. Vol. 26. Springer.

Laurent, Hyafil and Ronald L Rivest (1976). “Constructing optimal binary decision trees is NP-complete”. In: *Information processing letters* 5.1, pp. 15–17.

Lim, Tjen-Sien, Wei-Yin Loh, and Yu-Shan Shih (2000). “A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms”. In: *Machine learning* 40.3, pp. 203–228.

Mantovani, Rafael G et al. (2015). "Effectiveness of random search in SVM hyper-parameter tuning". In: *Neural Networks (IJCNN), 2015 International Joint Conference on*. IEEE, pp. 1–8.

McDonald, Ross A, David J Hand, and Idris A Eckley (2003). "An empirical comparison of three boosting algorithms on real data sets with artificial class noise". In: *International Workshop on Multiple Classifier Systems*. Springer, pp. 35–44.

Molinaro, Annette M, Richard Simon, and Ruth M Pfeiffer (2005). "Prediction error estimation: a comparison of resampling methods". In: *Bioinformatics* 21.15, pp. 3301–3307.

Niculescu-Mizil, Alexandru and Rich Caruana (2005). "Predicting good probabilities with supervised learning". In: *Proceedings of the 22nd international conference on Machine learning*. ACM, pp. 625–632.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Platt, John et al. (1999). "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods". In: *Advances in large margin classifiers* 10.3, pp. 61–74.

Quinlan, J. Ross (1987). "Simplifying decision trees". In: *International journal of man-machine studies* 27.3, pp. 221–234.

Roweis, Sam T and Lawrence K Saul (2000). "Nonlinear dimensionality reduction by locally linear embedding". In: *science* 290.5500, pp. 2323–2326.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). "Learning representations by back-propagating errors". In: *nature* 323.6088, p. 533.

Sasaki, Yutaka et al. (2007). "The truth of the F-measure". In: *Teach Tutor mater* 1.5, pp. 1–5.

Schölkopf, Bernhard, Alexander Smola, and Klaus-Robert Müller (1998). "Nonlinear component analysis as a kernel eigenvalue problem". In: *Neural computation* 10.5, pp. 1299–1319.

Shlens, Jonathon (2014). "A tutorial on principal component analysis". In: *arXiv preprint arXiv:1404.1100*.

Tenenbaum, Joshua B, Vin De Silva, and John C Langford (2000). "A global geometric framework for nonlinear dimensionality reduction". In: *science* 290.5500, pp. 2319–2323.

Tibshirani, Robert (1997). "The lasso method for variable selection in the Cox model". In: *Statistics in medicine* 16.4, pp. 385–395.

Tukey, John W (1962). "The future of data analysis". In: *The annals of mathematical statistics* 33.1, pp. 1–67.

Valentini, Giorgio and Thomas G Dietterich (2004). "Bias-variance analysis of support vector machines for the development of SVM-based ensemble methods". In: *Journal of Machine Learning Research* 5.Jul, pp. 725–775.

Vapnik, Vladimir N and Alexey J Chervonenkis (1974). "Theory of pattern recognition". In:

Varma, Sudhir and Richard Simon (2006). "Bias in error estimation when using cross-validation for model selection". In: *BMC bioinformatics* 7.1, p. 91.

Wasserman, Larry (2013). *All of statistics: a concise course in statistical inference*. Springer Science & Business Media.

Zhang, Yiyan et al. (2017). "Empirical study of seven data mining algorithms on different characteristics of datasets for biomedical classification applications". In: *Biomedical engineering online* 16.1, p. 125.

Zhang, Zhenyue and Jing Wang (2007). "MLLE: Modified locally linear embedding using multiple weights". In: *Advances in neural information processing systems*, pp. 1593–1600.

Zou, Hui and Trevor Hastie (2005). "Regularization and variable selection via the elastic net". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67.2, pp. 301–320.