Performance and Interpretability of Machine Learning Algorithms for Credit Risk Modelling

Leonhard Kampfer

Munich 2018

Performance and Interpretability of Machine Learning Algorithms for Credit Risk Modelling

Leonhard Kampfer

Master Thesis
at the Department of Statistics
of the Faculty for Mathematics, Informatics and Statistics
at the Ludwig-Maximilians-University Munich

Author:
Tassilo Leonhard Kampfer

Supervisors:
Prof. Dr. Stefan Mittnik
Christoph Berninger

Munich, 7th November 2018

Erklärung

Hiermit versichere ich, dass ich meine Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Munich, 7th November 2018 ....
Leonhard Kampfer

Abstract

Machine learning algorithms became increasingly important and available in the last years. They learn automatically from past experiences to do better in the future.

The paper at hand applies seven classification models to credit card data in order to predict defaults. The performance of the machine learning algorithms outperforms the benchmark logistic regression model by far, thus, provides promising results for application in credit risk modelling.

One major issue of machine learning is the lack of interpretability: the decision-making process is often considered as a black box. We present some model-agnostic methods which make models more interpretable and help to build trust in their prediction, although they cannot break up the black box completely.

Contents

Chapter 1

Introduction

According to Ben Bernake, the then Chairman of the Federal Reserve System (USA), the reasons for the "worst financial crisis in global history, including the Great Depression" are to be found in failures in the lending business (The Financial Crisis Inquiry Commission 2011). The possible impact of bad risk management not only for the financial industry but also for the economy as a whole has been drastically revealed. Additionally, in the aftermath, the crucial role of consumer behaviour at every stage of the recent financial crisis has been exhibited (Khandani, Kim & Lo 2010). In 2016, consumer spending contributed 53.5% or €5,592 bn to the GDP of the euro area[1], while the outstanding debts of housholds added up to €6,264 bn[2]. This clearly shows the major role of consumer behaviour and the vital importance of consumer credit risk evaluation.

Therefore, banks and insurance companies invest significant resources and develop sophisticated programs to manage their risk. Financial risk assessment is an area of great interest for academics, politics, regulators, and financial intermediaries. The measure of their customers' credit risk as accurately as possible and establishing a comprehensive and reliable risk management are of major significance for institutes and crucial for their financial success. Even small improvements in the prediction quality of the borrowers' repay ability can have a major impact on stability and profitability of lenders.

Common approaches like CreditMetrics or CreditRisk$^+$ (JP Morgan 1997, Credit Suisse 1997) use classical statistics to forecast defaults. Current credit-bureau analytics such as credit scores are based on slowly varying consumer characteristics, thus they are not feasible for tactical risk management decisions by chief risk officers and policymakers.

[1]http://ec.europa.eu/eurostat/statistics-explained/index.php/National_accounts_and_GDP (20.05.2018)

[2]https://www.ecb.europa.eu/press/pdf/ffi/eaefd_4q2016_early.pdf?aba27c2713b960657b8582dc992ea581 (20.05.2018)

The rapid increase in data availability and computational power makes a new group of methods available: Machine learning techniques become more and more important in finance in general and in risk management in particular. They are considerably more adaptive to dynamics of changing credit cycles and are able to capture complex non-linear, non-monotonic relationships (Khandani et al. 2010).

In this thesis, we want to have a look at different machine learning approaches for predicting consumers' defaults and examine interpretability and transparency of the methods since regulators often have requirements as to that. The aim is to produce an extensive review of possible machine learning approaches for consumer credit risk evaluation, to show and explain advantages and disadvantages of these techniques, to compare their accuracy and prediction quality, and to assess and improve their interpretability.

Chapter 2

Credit Risk Modelling

We want to assess the performance of several machine learning algorithms for credit risk modelling. In this chapter, we give a overview of the topic, provide some definitions, and explain basic concepts. Interpretability of machine learning will be discussed in chapter 4 and following.

2.1 Credit Risk

According to the Basel Committee on Banking Supervision and Bank for International Settlements (2000), credit risk, also known as counterparty or default risk, is “the potential that a bank borrower or counterparty will fail to meet its obligations in accordance with agreed terms”. It describes the risk that a customer fails willingly or unwillingly to make a required payment – such as a mortgage, overdraft, or credit card debt – duly, and defaults. A default is defined as a delinquency by a determined timeframe, for example, by 90 or more days.

In this paper, we will assess our customers’ credit risk by predicting the probability that a customer will pay his credit card debts duly next month (see section 3.1.1 for details on the data).

Credit risk management tries to retain the lender’s credit risk exposure within acceptable boundaries and thereby to maximise the profit rate (Basel Committee on Banking Supervision and Bank for International Settlements 2000). It is an essential component of a comprehensive enterprise risk management and critical for long-term success. The business perspective would go beyond the scope of this paper, the focus lies on risk modelling. Based on the financial risk assessment, the lender may introduce appropriate strategies to control its risk exposure. It can hedge some of its risks by purchasing credit insurance or credit derivatives, or reduce its exposure by selling credit portfolios to investors. Another approach is called tightening, which describes the risk reduction by lowering the risk exposure by, for example, cutting the line of credit for a portfolio of customers or truncating the number of newly sold credits (Khandani et al. 2010).

Most major companies deploy significant resources and develop sophisticated programs to analyse and manage their risk. One common approach is credit scoring which tries to attribute a number (i.e. the score) to a customer based on its predicted default probability. The score is usually calculated using a range of data sources such as application forms, credit agencies, or existing products of the customer with the lender (Fahrmeir, Kneib, Lang & Marx 2013). Based on the score, the bank may apply different strategies as adjust the line of credit, ask for more collateral, or charge borrowers with bad scores higher interest rates, i.e. price risk-based.

2.2 Binary Classification

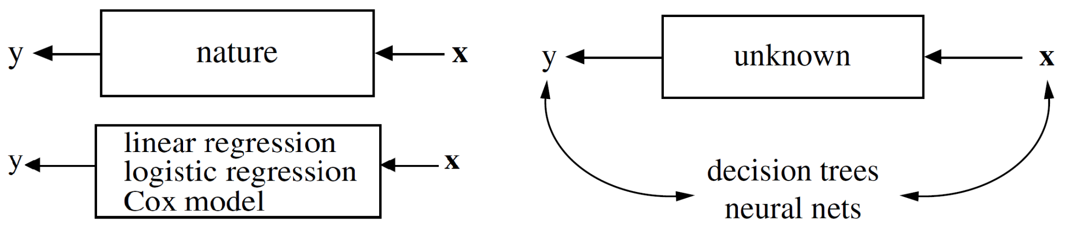
Customer credit risk evaluation can be described as a supervised binary classification.

The prediction of default is a binary classification problem since we want to identify to which category a new customer belongs: to the class with "good" customers to whom the bank wants to give a loan (since he has a high probability of paying back his credit duly), or to the class of "bad" customers to whom the bank does not want to lend money.

According to James, Witten, Hastie & Tibshirani (2013), there are mainly two categories of learning problems: supervised and unsupervised. While in the former category the algorithm is provided with a response associated with each observation of the predictor, in the latter the associated response is absent. We formulate a supervised learning problem with customers' default as the associated response. We try to forecast if a specific customer is likely to default or not given the input data. One possible approach seems to be to "memorise" the input-output pairs (Khandani et al. 2010). This would perfectly map the pairs in the dataset used for training the algorithm but it is unlikely to be successful in predicting outputs for new data. So, we need to find a mapping function that fits both the dataset used for training and any new customer data[3].

Therefore, we create a classifier which learns underlying relationships between input and output (Tsai & Chen 2010). Then, it assigns a certain default probability to each instance. If the predicted probability is higher than a defined threshold, we predict default, otherwise non-default. Since a bank will certainly refuse customers which are flagged as default, classification is in effect a decision based on calculated probabilities.

[3]This challenge is often referred to as the "bias-variance tradeoff" (cf. James et al. 2013).



(a) Data modelling culture

(b) Algorithm modelling culture

Figure 2.1: The two cultures of statistical modelling (Breiman 2001b).

2.3 Machine Learning

According to Breiman (2001b), there are two cultures in the use of statistical modelling to reach conclusions from data and, hence, to solve a binary classification problem.

Nature forms from input x the output y. The complex and unknown transformation is seen as a black box (see top figure in figure 2.1a). Statisticians aim to get information about the underlying data mechanisms and to predict, for example, defaults of customers. There are two main approaches to achieve this.

One assumes that the the output y is generated by a given stochastic data model. This approach tries to emulate the transformation of the input data by finding the model which is most similar to nature (see bottom figure in figure 2.1a).

The other treats the data transformation mechanism as unknown and uses algorithmic models to map input and output data (see figure 2.1b). This concept focuses on the data and the problem instead of asking what model creates the same output as nature does. These data-driven methods have arisen rapidly by increasing availability (and decreasing prices) of large storage capacity, high computational power, and big data sets over the last decades, and become more and more important in many areas (cf. Chen & Guestrin 2016). This master thesis will focus on the latter approach, which is known as machine learning.

Machine learning is about, simply said, learning automatically from past experiences to do better in the future. It is a hypernym for several methods which teach computers to analyse data, learn from it, and make predictions regarding new data (cf. Simon 1983). The aim is to develop and apply learning algorithms that do the learning and predicting automatically, without human intervention or assistance. We do not programme the computer to solve the task directly. In fact, we implement techniques to allow the computer to come up with its own programme based solely on provided examples in

Statistics	**Machine Learning**
Covariate	Feature/ Attribute
Maximising likelihood	Minimising loss
Fitting/Estimation	Learning
Parameter/Coefficient	Weights
Intercept	Bias term
Model	Hypothesis
Observation	Example/Instance
Regression/Classification	Supervised learning
Response	Label
Log. regr. as regression	Log. regr. as classification

Table 2.1: Different notations in statistics and machine learning.

order to capture complex data dependencies (Chen & Guestrin 2016). Machine learning, a subarea of artificial intelligence, intersects broadly with other scientific fields like statistics, mathematics, and theoretical computer science. It is used in Apple's Siri, in Netflix' movie recommendations, translation programmes, or image analyses. All these applications use massive input data and have the ability to learn from the datasets.

Classical statistics and machine learning often use different technical terms for the same matter. Table 2.1 gives an overview of how to link different notations used in statistics and machine learning. Furthermore, we use the terms model and algorithm synonymously for machine learning (and classical statistical) methods in the paper at hand. In combination with a threshold in order to assign the instances to certain classes based on their predicted probabilities, we also call it a classifier. A learner is an algorithm with a certain parametrisation, so there can be several learners of the same algorithm.

Chapter 3

Methodology

The aim of the paper is to conduct a comparative study of machine learning algorithms in credit risk modelling and to assess their interpretability. In this chapter, we will define and exhibit our general setting for the study.

The computational part of the project is done in the programming language `R` (R Core Team 2018). It offers a great number of packages and functions for modelling and machine learning. We use the package `mlr` (Bischl, Lang, Kotthoff, Schiffner, Richter, Studerus, Casalicchio & Jones 2016) as a framework for data preprocessing, tuning, resampling, application of algorithms, and evaluation. It provides a consistent interface to a great number of prediction algorithms via additional packages and several analysis tools. The applied algorithms and the therefore used packages will be described in chapter 5. Furthermore, we deploy package `iml` (Molnar, Bischl & Casalicchio 2018) to visualise our results and make the algorithms more interpretable.

3.1 Data Set

3.1.1 Data Description

The models and algorithms are applied to a data set of a major bank in Taiwan (cf. Yeh & Lien 2009). It contains the data of 30,000 credit card customers, 24 features with no missing values, and one response label. There are features with personal information like age and sex, and financial information like credit card billing or usage in the previous six months. The binary response `DEFAULT` indicates if a customer paid back all her credit card debt in October 2005 and its characteristic is tried to be predicted by the applied models and algorithms.

There are about 78% non-default and 22% default customers in the data. This is

Feature name	**Scale**	**Description**
`ID`	metric	Customer ID
`LIMIT_BAL`	metric	Credit line in thousand TWD granted to customer
`SEX`	nominal	Gender of customer
`EDUCATION`	nominal	Education (graduate school, university, high school, etc.)
`MARRIAGE`	nominal	Martial status (married, singel, divorced, other)
`AGE`	metric	Age in years
`PAY_`*x*	nominal	Repayment status (no consumption, paid in full, revolving credit, payment delay for 1, 2, ..., 8 months) in month x
`BILL_AMT`*x*	metric	Amount of bill statement in month x
`PAY_AMT`*x*	metric	Amount of payment in month x
`DEFAULT`	nominal	Default in October 2005, response

Table 3.1: Overview of all features and the response in the data set about Taiwanese credit card customers. $x = 1, \dots, 6$ denotes months April to September.

remarkable since the portion of defaults is usually much smaller in such data sets, i.e. they are often more imbalanced. We refrain from introducing and applying imbalanced data correction at this point for two reasons: First, it would go beyond the scope of the paper, and second, at this extent of imbalance a correction is not critical (cf. Bischl et al. 2016).

As seen in table 3.1, there are 14 metric features[4] in the data set. Table 3.2 describes their minimum and maximum values, as well as their medians and means. For `BILL_AMT`*x*, the table summarises the six separate features for the months April to September and treats them as one – the same procedure is done for `PAY_AMT`*x*. Negative values for `BILL_AMT`*x* indicate that more money was transferred to the credit card account than used for payments (in the previous month) by the customer. In an ideal situation, the amount of the bill statement `BILL_AMT`*i* in one month is equal to the amount of payment `PAY_AMT`*(i+1)* of the next month, which means that the full amount was paid duly.

Table 3.3 gives an overview of all nominal features and their modes. Features `PAY_`*x* denote the status of the past payments with "no consumption", "paid in full", "revolving credit"[5], or the number of months a payment is delayed. There are in total 180,000 months recorded for `PAY_`*x* in the data (see also table 3.3):

[4]Without customer ID.

[5]A revolving credit is defined as a payment of the minimum due amount while the credit card account has still a positive balance (i.e. the customer is still in debt) at the end of the period due to recent transactions for which payment has not yet come due.

Feature name	Minimum	Median	Mean	Maximum
`LIMIT_BAL`	10	140	167	1,000
`AGE`	21	34	35.5	79
`BILL_AMT`x	-339,600	19,720	44,980	1,664,000
`PAY_AMT`x	0	1,900	5,275	1,684,000

Table 3.2: Overview of metric features, their minimum, median, mean, and maximum values. `BILL_AMT`x and `PAY_AMT`x are each summarised for all $x = 1, \dots, 6$ months and treated as one feature.

Feature name	Mode	Other classes and their probabilities
`SEX`	Women (60%)	Men (40%)
`EDUCATION`	University (47%)	Graduate School (35%), High school (16 %)[6]
`MARRIAGE`	Single (53%)	Married (46%), Divorced and Other (1%)
`PAY_`x	Revolving credit (53%)	Paid in full (19%), Delay (for one or more months) (14%), Inactive (14%)
`DEFAULT`	Non-default (78%)	Default (22%)

Table 3.3: Overview of all nominal features, their mode and the portions for their classes. `PAY_`x is summarised for all $x = 1, \dots, 6$ months and treated as one feature.

- Customers were inactive in 24,415 months,
- The full amount was paid in 34,640,
- A revolving credit was used in 95,919, and
- In 25,026 months, the payment was delayed for one or more months.

Thus, features `PAY_`x indicate two things: first, if the customer is delinquent with her payments or paid in due time. And second, if she delayed a payment, for how long the payment is delayed.

One has to keep in mind that our data set only contains customers whose application have been accepted. So, there is no payment or default information about rejected customers. This problem of censorship in the data is often referred to as reject inference (Kruppa, Schwarz, Arminger & Ziegler 2013). Nevertheless, this issue cannot be corrected or resolved ex ante, and the comparison of different machine learning algorithms for credit risk modelling and its results are still reasonable.

[6]There are four more classes, indicated as "Others" or "Unknown", with together about 2%.

3.1.2 Data Preprocessing

Data preprocessing refers to any transformation of the data done before applying a learning algorithm. This comprises, for example, finding and resolving inconsistencies, imputation of missing values, identifying, removing or replacing outliers, discretising numerical data, or generating numerical dummy features for categorical data.

No feature engineering is applied to the data set for this study. Feature engineering is about generating or creating new features based on existing information in the data set, for example by introducing certain indicators, combining several features into one or splitting up one into multiple. Feature engineering can be very time consuming but also rewarding in terms of improvements of prediction quality. Since it needs to be done individual for different algorithms, it would reduce the comparability between them. Furthermore, it is not the aim of the paper and would exceed the scope, so it is omitted at his point.

One exception is the class level merge of `PAY_`$\boldsymbol{x}$, which is applied to some learners of each algorithm. We join the classes with delayed payment ("delay for one month", "delay for two months", ...) into one class "delay". This will be mentioned explicitly if applicable.

For algorithms like k-nearest neighbour (see section 5.2), the data needs to be normalised, i.e. scaled to mean$= 0$ and standard deviation$= 1$, in order to produce meaningful results. This is done by using the standard score z_n:

$$z_n = \frac{X - \mu}{\sigma}, \tag{3.1}$$

where μ denotes the mean and σ the standard deviation.

Furthermore, we remove any constant features and the customer IDs before applying any algorithms. Constant features can lead to errors in some algorithms like generalised linear models (Bischl et al. 2016), and IDs could lead to spurious results if the data is, for example, ordered by a specific logic.

3.2 Hyperparameter and Threshold Tuning

Most machine learning algorithms have characteristic variables, called hyperparameters, which steer several aspects of the prediction process. Different settings for theses hyperparameters may lead to different results, and a diligent search for the optimal setting is essential in order to obtain best possible prediction results. The problem of selecting the optimal set of characteristic parameters for a learning problem is called hyperparameter

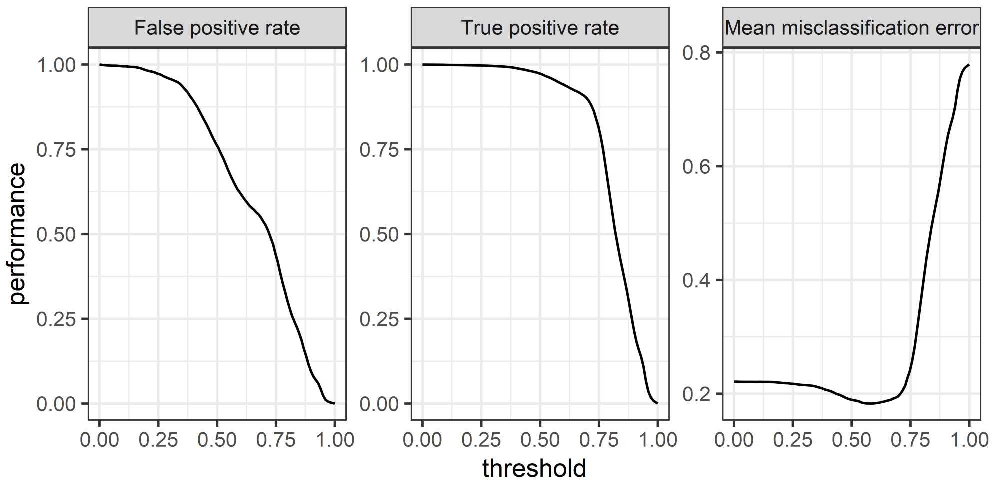


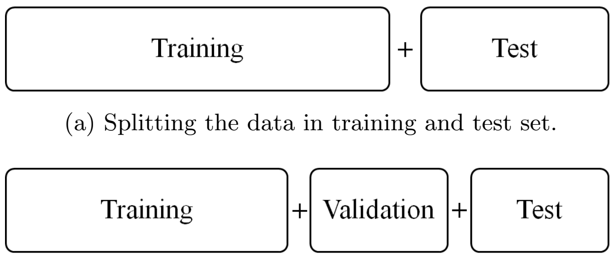
Figure 3.1: Performance measured as false positive rate, true positive rate and mean misclassification error subject to different thresholds.

tuning (Friedman, Hastie & Tibshirani 2001b).

We apply several hyperparameter sets and different search spaces (the range where the hyperparameters lie) to the algorithms, producing various learners. For reasons of clarity, we only present four learners for each algorithm. For example, we tune the number of neighbours k we look at for the kNN approach, or the number of hidden layers for artificial neural networks.

In addition to the hyperparameters, we also tune the threshold for all learners. We assign a number between 0 and 1 to each customer representing his probability of default, also called score. We search for the optimal cut-off point of this score to distinguish the classification of a customer as default or non-default (cf. Khandani et al. 2010).

Figure 3.1 shows on the right the prediction power of a algorithm, measured by the mean misclassification error (mmce, see section 3.4 for details), as a function of different threshold values. One can see clearly the tremendous effect of the chosen cut-off point on the performance and that, other than one might have expected, 0.5 is not the optimal value for the threshold. The plots on the left and in the middle display the performance measured by false positive rate (FPR) and true positive rate (TPR) (see section 3.4 for details), respectively. One can see that the trade-off between those two measures is not linear, i.e. a decrease in the FPR is not always accompanied by a decrease in TPR. This offers optimisation opportunities by finding an optimal threshold with a low FPR while the TPR is still high (Lobo, Jiménez-Valverde & Real 2008). This can also be applied in



(a) Splitting the data in training and test set.

(b) Splitting the data in training, validation and test set.

Figure 3.2: Different data set splits for resampling.

order to manage risk in combination with cost-benefit-analysis (Khandani et al. 2010).

3.3 Resampling

We split our data set into a training and a test subset in order to measure the performance of our learners (see figure 3.2a): First, we train our model on the training subset. Then, we predict the labels for the instances in the test subset and evaluate our prediction performance by comparing the actual labels with the predicted ones. The issue of this approach is that our performance estimation relies highly on the specific samples used for training and testing[7]. To avoid this downside, we repeat the procedure above several times with different splits each time and summarise the results. This approach is called resampling.

We apply 10-fold cross-validation as resampling strategy. Therefore, we split our data set randomly into 10 equal subsets. We use 9 subsets to train the model, predict the labels for the 10th subset, and assess the performance. We repeat this procedure 10 times, predicting at each iteration the label for another subset, and summarise all results as an estimate for the overall performance of the learner. Hereby, we reduce the dependence on the specific sample, and produce an unbiased estimator of the prediction power (Tsai & Chen 2010).

Like described in section 3.2, tuning is essential for good prediction results. In order to find the optimal hyperparamter sets and thresholds, we introduce a further split to our data set and a second resampling loop: We split into training, validation and test subset (see figure 3.2b) and resample in an inner and outer loop.

We tune our hyperparameters and threshold on the training subset in the inner

[7]See http://www.cs.uwyo.edu/ larsko/ml-fac/04-resampling-exercises.Rmd (23.07.2018) for an example on good, bad, and ugly splits.

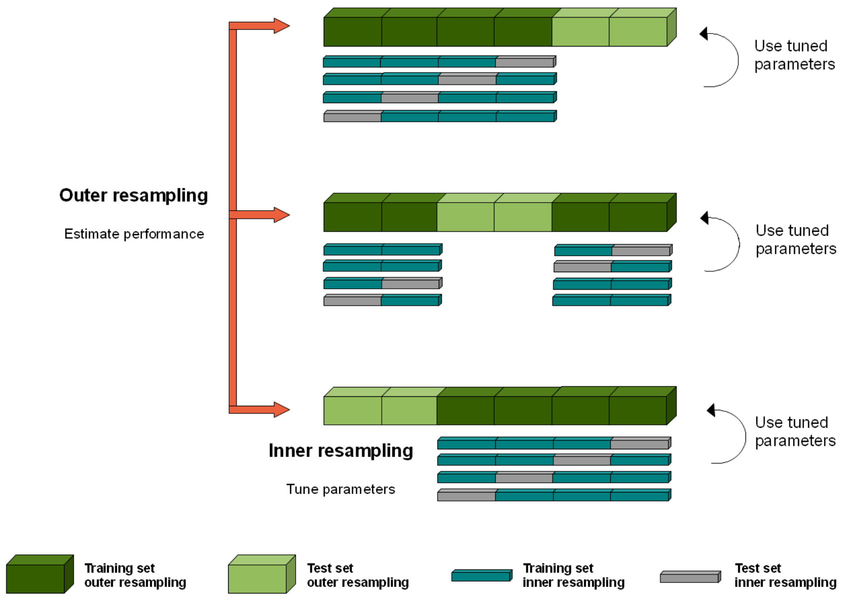


Figure 3.3: Concept of nested resampling with fourfold cross-validation in the inner and threefold cross-validation in the outer loop (Bischl et al. 2016).

loop. The tuning is done via 10-fold cross-validation, i.e. train our model with different hyperparameter settings on the training set, assess the results on the validation set in order to find the optimal set, and repeat this procedure for all 10 validation sets.

In the outer loop, we evaluate the prediction performance of the learner with the optimal hyperparameters for this subset learnt in the inner loop. The predictive power is then assessed via 10-fold cross-validation by several performance measurements introduced in section 3.4.

The split in three subsets is necessary to get honest and stable estimations of prediction power. The inner loop is necessary to find the optimal hyperparameter set, where resampling is needed to compare the results for different settings. The evaluation of the prediction power of a learner has to be done on unseen data to get reliable, unbiased results. Thus, resampling with two loops, called nested resampling, is necessary. Figure 3.3 illustrates nested resampling for parameter tuning with four-fold cross-validation in the inner and three-fold cross-validation in the outer loop.

Nested resampling is computationally very expensive. For example, if we tune one single hyperparamter, which can attain three values, using 10-fold cross-validation in both the inner and the outer loop, we have to train and test our model $3 \times 10 \times 10 = 300$ times. The huge computational burden reduces the number of hyperparameters to

		Actual outcome	
		0	**1**
Predicted outcome	**0**	True negative (TN)	False negative (FN)
	1	False positive (FP)	True positive (TP)
		True negative rate (TNR) $= \text{Specificity} = \frac{\lvert TN\rvert}{\lvert TN\rvert+\lvert FP\rvert}$	True positive rate (TPR) $= \text{Sensitivity} = \frac{\lvert TP\rvert}{\lvert FN\rvert+\lvert TP\rvert}$

Figure 3.4: Model of a confusion matrix. In this paper, 0 or negative indicate non-default, 1 or positive default.

be tuned and shrink their search spaces in order to obtain a feasible runtime of the nested resampling. The package `parallelMap` (Bischl & Lang 2015) is used to perform parallelisation of the resampling process and speed up the runtime.

There is some discussion about how to split the data set into training, validation and test set, and how big each proportion should be (cf. Guyon 1997). It is a trade-off between having more data to train the model – which lead to a better trained model – and having more data to validate the model – which lead to a more precise evaluation of the model's performance. We choose to have 10% as the test set. From the remaining data set we sample 10% as the validation set. This enables us to perform a 10-fold cross-validation in both the outer and the inner loop to tune and validate our algorithms.

3.4 Performance Measurements

In order to evaluate if an algorithm or learner is better than another, we have to define what "better" means. Therefore, we introduce several performance measures to quantify the prediction power of our learners (cf. James et al. 2013).

Confusion Matrix

Confusion matrices are among the most common performance measurements for machine learning problems (cf. Khandani et al. 2010). Figure 3.4 displays a model of a confusion matrix and specifies the most important numbers. Confusion matrices reveal several important insights of a learner at a glance.

Each instance, for which a prediction is made, falls into one of the four cells. The left column gathers all instances with actual 0, while the right column contains instances with 1. The rows correspond to the predicted outcome: the first row contains instances who are classified as 0, instances with predicted 1 go into the second row.

True negative (TN) indicates the number of instances which are predicted as 0 and are actual 0 (in this paper non-default), while true positive (TP) indicates the amount of instances correctly predicted as 1 (i.e. default). False negative (FN) and false positive (FP) state the instances wrongly classified as negative or positive, respectively. Following the Neyman-Pearson hypothesis-testing framework, FP can be considered as Type-I error, and FN as Type II-error (Tsai & Chen 2010). The true positive rate (TPR) defines how many correct positive predictions occur among all positive instances in the test subset. The false positive rate (FPR), on the other hand, defines how many incorrect positive predictions occur among all negative instances in the test subset and can be calculated by

$$FPR = 1 - Specificity = 1 - \frac{|TN|}{|TN| + |FP|}, \tag{3.2}$$

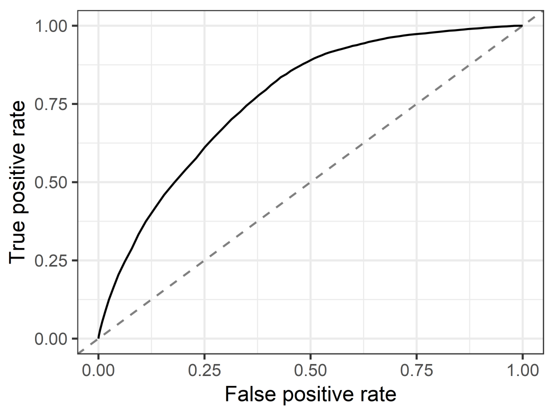
with $|\cdot|$ the cardinality.

Receiver Operating Characteristic (ROC) Curve

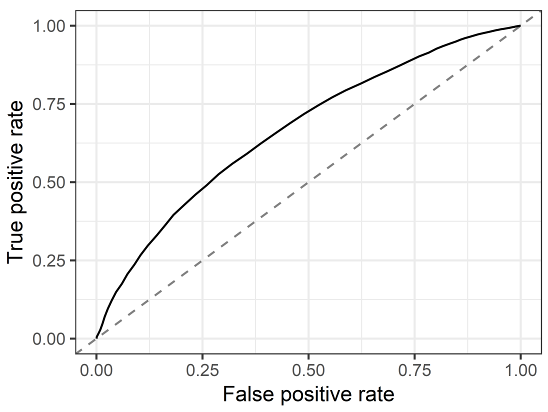
The receiver operating characteristic curve (see figure 3.5 for example ROCs) is a popular graphic for comparing classification algorithms and summarises TPR and FPR for all possible thresholds (cf. James et al. 2013).

The vertical axis displays the TPR, the horizontal the FPR. We calculate the two measures for all possible thresholds (i.e. $n - 1$ thresholds for n instances) and draw points on the ROC space. Alternatively, one can see the ROC curve as the result of plotting the cumulative distribution function (CDF) of TPR on the vertical axis against the CDF of FPR on the horizontal axis for all thresholds.

An ideal ROC curve would go through the point (0,1), i.e. through the top left corner, and would imply the existence of a threshold where all defaults have predicted probabilities above this value, and the non-defaults below, respectively. A random guess would follow the diagonal. The ROC describes only how well models rank defaults and non-defaults, for example, and does not evaluate the actual predicted probabilities



(a) ROC curve of Random Forest 3 with $AUC = 0.7671$



(b) ROC curve of kNN 1 with $AUC = 0.6671$

Figure 3.5: ROC curves for two algorithms with different AUCs.

(Cook 2007). An algorithm calculating a default probability of 0.20 for all defaults and 0.18 for all non-defaults would have perfect discrimination power, hence an optimal ROC curve, but the assigned probabilities are not reasonable.

Area Under The ROC Curve (AUC)

The area under the receiver operating characteristic curve (AUC) is a widely used measure to evaluate classification performance, particularly in retail banking (Hand & Anagnostopoulos 2013). The AUC tries to compile the information of the ROC into one number and states the integral of the ROC curve as mentioned above. It takes 1 for a perfect classifier and 0.5 for a random chance classifier. Values between 0 and 0.5 indicate classifiers that perform worse than chance and can be improved by inverting the predicted class (in the binary classification case).

Like the ROC curve, AUC does not evaluate the predicted probabilities (Kruppa et al. 2013).

Classification Accuracy (ACC)

The classification accuracy is the proportion of the correctly classified over all observations:

$$ACC = \frac{|TP| + |TN|}{n} = 1 - MCE, \tag{3.3}$$

with $MCE = \frac{|FP|+|FN|}{n}$ the misclassification error and n the total number of instances. A perfect prediction would result in an ACC of 1, the worst possible score is 0.

Balanced Accuracy (BAC)

The balanced accuracy is defined as the mean of true positive rate and true negative rate, hence

$$BAC = \frac{1}{2} \cdot (TPR + TNR) = \frac{1}{2} \cdot \left(\frac{|TP|}{|TP| + |FN|} + \frac{|TN|}{|TN| + |FP|} \right). \tag{3.4}$$

The measure is bounded between 0 and 1 with higher values indicating better performance.

Brier Score (Brier)

Introduced by Brier (1950), the Brier score measures the accuracy of the probabilistic prediction. In the original formulation, the score can take on values between two and zero. We apply an alternative formulation, which takes on values between zero and one, and is calculated as

$$Brier = \frac{1}{n} \sum_{i=1}^{n} (f_i - y_i)^2, \tag{3.5}$$

where $f_i \in [0, 1]$ is the predicted probability and $y_i \in \{0, 1\}$ the actual outcome. The lower the score, the more accurate the predicted probabilities.

Kolmogorov-Smirnov statistic (KS)

The Kolmogorov-Smirnov statistic (Kolmogorov 1933, Smirnov 1939) quantifies the maximum absolute distance between two empirical cumulative distribution functions (ECDFs) (see figure 3.6). The distance D between two ECDFs $F_{1,m}(x)$ and $F_{2,l}(x)$, with m and l instances each, can be calculated as

$$D_{1,2} = \sup_x |F_{1,m}(x) - F_{2,l}(x)|, \tag{3.6}$$

with sup the supremum function (cf. Anderson 2007). A high KS statistic indicates a good discriminatory power, hence a good predictive power of the classifier.

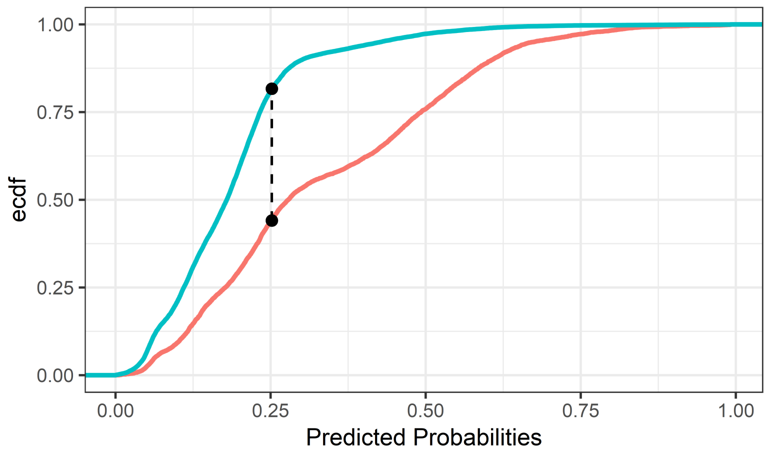


Figure 3.6: Kolmogorov-Smirnov statistic with marked maximum absolute distance between the two ECDFs for non-default in cyan and default in red.

AUC	**ACC**	**BAC**	**Brier**	**KS**
0.5	0.7788	0.5	0.2212	0

Table 3.4: Performance results of a trivial classifier which predicts a default probability of 0 for all instances.

Application

The introduced measures answer different questions and are calculated differently. Thus, we might have the case where one measure suggests that learner A is the best whereas another measure suggests learner B. We choose AUC to be our main measurement to compare the prediction performance of different learners and algorithms because of its popularity in credit risk (cf. Hand & Anagnostopoulos 2013).

The runtime will be measured in minutes. It not only depends on the method but also on the concrete implementation in the `R` package. Furthermore, a main driver in runtime is the number of hyperparameters to be tuned and their search space (cf. section 3.3). Hence, the displayed runtimes should be considered as rough indications and not as exact runtime predictions.

Table 3.4 shows the measures for a trivial classifier which predicts a default probability of 0 for all customers, i.e. classifies all instances as safe non-default customers. These values can be seen as an actual lower boundary of prediction performance.

Chapter 4

Black Box and Interpretability

In this chapter, we investigate interpretability of machine learning algorithms, see why their back boxes might cause problems, and try to find solutions to break them up.

4.1 The Black Box Problem

Most machine learning algorithms are considered as black boxes (cf. Ribeiro, Singh & Guestrin 2016). They create non-linear, non-monotonic models which are typically less transparent (Hall, Gill, Kurka & Phan 2017). The reasons, why an algorithm connects inputs to specific outputs or results in concrete predictions, remain often untold.

However, machine learning methods enter increasingly critical areas like the criminal justice system, medicine, and financial markets (Lipton 2016). If humans are not able to understand the decision-making processes and cannot assess when these are likely to fail, they might not trust in the predictions and stop deploying these algorithms.

Another important issue is that black boxes open the way to fraud and hostile attacks. Papernot, McDaniel, Goodfellow, Jha, Celik & Swami (2017) describe methods to corrupt an artificial neural network. They fuel models with malicious inputs which are imperceptible modified but appear legitimate to humans. These changes force the model to erroneous predictions and classifications, while humans would still classify correctly. The attacks can be used to plant malware or to control vehicle behaviour of autonomous cars.

Ribeiro et al. (2016) denote interpretability as the most important factor in order to create trust in algorithms which is essential if one takes actions based on model predictions. Trusting an individual prediction is as important as trusting the whole model to behave reasonable if deployed. Users need to understand decisions and be confident that there will be good performance on unseen data, before they apply complex models to the real world. Furthermore, debugging and auditing is only possible if there is an understanding of the models' prediction processes.

Especially in credit scoring, interpretability and explicability are important for the acceptance of models. Logistic regression models are often considered as more interpretable than, for example, neural networks. Furthermore, the customers right of explanation of algorithm-based decisions is seen as a blocker for further propagation of complex machine learning models (cf. European Parliament 2016). This might be a reason why machine learning methods are not widespread in credit risk for retail banks – despite their promising performance results.

Arguably, there are scenarios where interpretability is not needed or wanted, for example if the algorithm has no significant impact or if interpretability would enable gambling the system. Nevertheless, developing interpretable models still allows the model owner to cover the decision-making process if necessary.

4.2 Properties of Interpretable Algorithms

A classical statistician[8] might answer the question about interpretability with "the possibility to write down the model equation". For most machine learning algorithms, it is not possible to set up an equation like for linear models. But is a model equation the ideal in respect of making algorithm-based decisions more interpretable? It may not be feasible to present a model equation if there are hundreds of significant features in genomic studies, for example. What are general idea and desired properties of interpretability? How can we represent algorithms and their way of making a prediction more interpretable for users?

First, the bad news: There is no passe-partout, no one-fits-all method which can be applied to all models and algorithms and offers perfect and comprehensive interpretability for all purposes. Furthermore, we have to realise that accuracy and simplicity are in conflict in prediction (Breiman 2001b). And less simplicity means in most cases less interpretability.

The definition of interpretability is not uniform in literature. Sometimes, it is equated with transparency or understandability, i.e. one can see and understand how the model works (cf. Lipton 2016). The more interpretable a model is the easier it is to understand. Others try to explain the prediction by visual or textual figures or give qualitative

[8]However one might define classic statistics, probably as "the frequentistic statistics one learns in the basic studies in Statistics at LMU Munich".

understanding of the relationship between inputs and outputs or the model's prediction-making process (cf. Ribeiro et al. 2016).

Lipton (2016) summarises different aspects and situations when interpretability is desired:

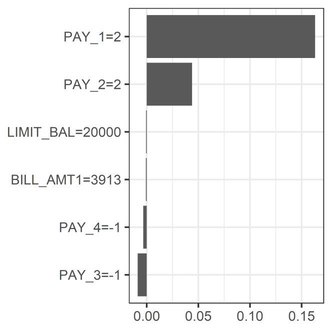
- Trust: Users want to trust predictions and models' behaviour in unexpected situations. Additionally, trust is essential for the willingness to deploy such models.
- Causality: Researchers often use models in order to generate hypotheses or make inference about the real world. Machine learning algorithms may not reflect those causal relationships accurately.
- Transferability: We want to transfer prior knowledge and additional information into the decision-making process of the algorithm.
- Informativeness: Models are also used to provide information to human decision makers. Interpretable models provide additional information about the diagnosis and delivers useful support to the user.
- Fair and Ethical Decision-Making: Decisions produced by algorithms must be conform with legal and ethical standards – and this must be verifiable.

We want results to be explainable to and understandable for users. There are two main approaches to make models and their decisions interpretable to humans (Hall et al. 2017).

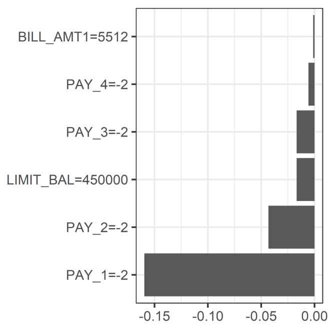
One refers to transparency, i.e. directly interpretable models. These are model-specific methods that are only applicable to a specific type of algorithm and try to elucidate how the model works by giving insights in the mechanisms of the algorithm.

The other is post-hoc or model-agnostic interpretability. These methods can be applied to various types of algorithms and give an understanding of the relationship between inputs and outputs without knowledge about the actual functionality of the model. They provide information about the model's decision by textual or visual representations or by explanations via example.

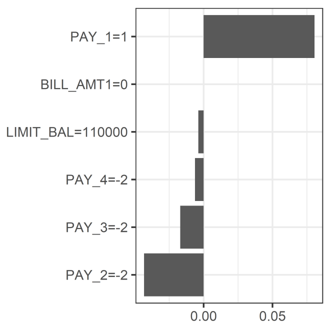
We distinguish between interpretability on a global, modular, and instance level. Global interpretability allows us to understand the entire relationship modelled and gives insights into each step in the decision-making process. Interpretability on a local or modular level provides understanding of parts of the model, of parameters, or of regions of the input data. If we can explain the predictions for concrete instances or in relation to other instances, interpretability on instance level is possible.



(a) Customer no. 1, default



(b) Customer no. 42, non-default



(c) Customer no. 123, default

Figure 4.1: Local surrogate model. Displays the effect of six features on the prediction of three selected customer.

4.3 Model-Agnostic Interpretability

We introduce three general approaches to make models and algorithms more interpretable. All three are model-agnostic approaches, hence, applicable to different machine learning algorithms.

Local Surrogate Model

Ribeiro et al. (2016) introduce Local Interpretable Model-agnostic Explanations (LIME), a local surrogate model. It explains a prediction for a single instance by creating a interpretable linear model that approximates the classifier locally. Molnar et al. (2018) slightly modify the LIME model in the package `iml` by using a different distance measurements.

We fit a weighted logistic regression model where the weights are calculated by Gower's similarity S_{ij} (Gower 1971)[9]. It measures the proximity of two data points i, j and is calculated as

$$S_{ij} = \frac{\sum_{k=1}^{n} s_{ijk}}{\sum_{k=1}^{n} \delta_{ijk}}, \tag{4.1}$$

where s_{ijk} denotes the contribution of the kth feature[10], and δ_{ijk} an indicator with $\delta_{ijk} = 1$ if i and j can be compared for k and $\delta_{ijk} = 0$ otherwise.

[9]LIME uses the Euclidean distance instead.
[10]See Gower (1971) for details on the calculation.

Figure 4.1 gives examples of local surrogate models for different customers. The figure 4.1a displays the six weighted features for customer with `ID` $= 1$ that explains the response the best according to the fitted linear model, and their effects on the prediction. One can see that `PAY_1` $= 2$, which denotes that the customer is in delay in September for credit card use two months ago, has a highly positive effect on `DEFAULT` probability. The same but to minor extent applies to `PAY_2` $= 2$, which denotes a two months delay in August[11]. `PAY_3` $= -1$ indicates that the customer paid all his due payments in July, which has a slightly negative effect on `DEFAULT`. The other variables have only a minor impact on the prediction according the local surrogate model. The interpretation of figures 4.1b and 4.1c goes accordingly.

Feature Importance

Feature or variable importance quantifies the impact of input features on the model's prediction (Friedman 2001a, Friedman et al. 2001b). The measure is calculated by shuffling each feature and measuring the performance drops in AUC. It assesses the decrease in impurity of the classification and measures the relative contribution to the calculated algorithm. However, the exact calculation varies for different algorithms, thus, for different methods, the feature importance numbers have different ranges and are not directly comparable (Liaw & Wiener 2002). Furthermore, importance does not imply significance.

Figure 4.2 shows the feature importance of a random forest[12]. One can see the huge importance of feature `PAY_1`, i.e. its huge impact on the prediction. The next important features are `LIMIT_BAL` and `BILL_AMT1`, while the other `PAY_`i features have only minor importance.

Partial Dependence Plots

Partial dependence plots (PDP) display the average effect of changing one feature on model prediction (Friedman 2001a, Friedman et al. 2001b). They can only show one feature at a time but take all instances into account.

Figure 4.3 reveals the PDP of three features. The PDP for `PAY_1` reveals how the probability of default would change for one (average) customer if the payment behaviour of the last month would change. One can see that for customers with `PAY_1`> 2, the

[11]Again, it denotes more precisely that the credit card was used two months ago, i.e. in June, and the bills have not been paid yet although they became due.

[12]See section 5.4 for details on random forest.

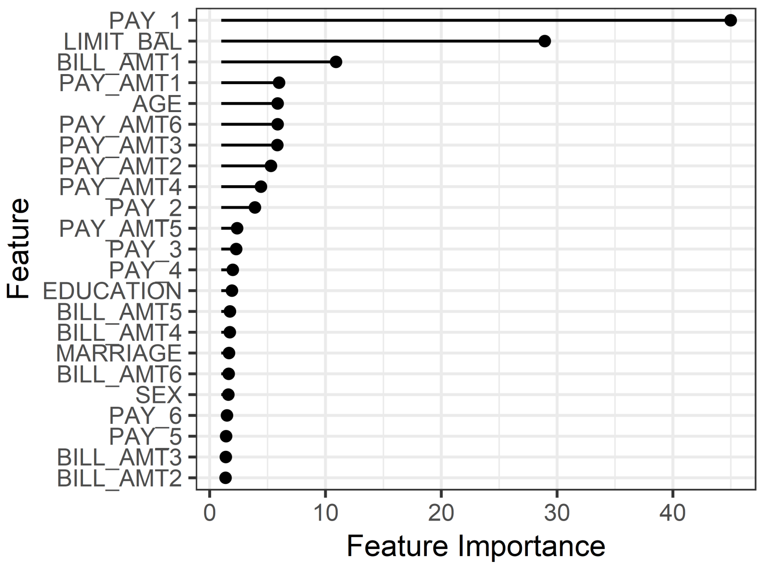


Figure 4.2: Feature importance of Random Forest 3.

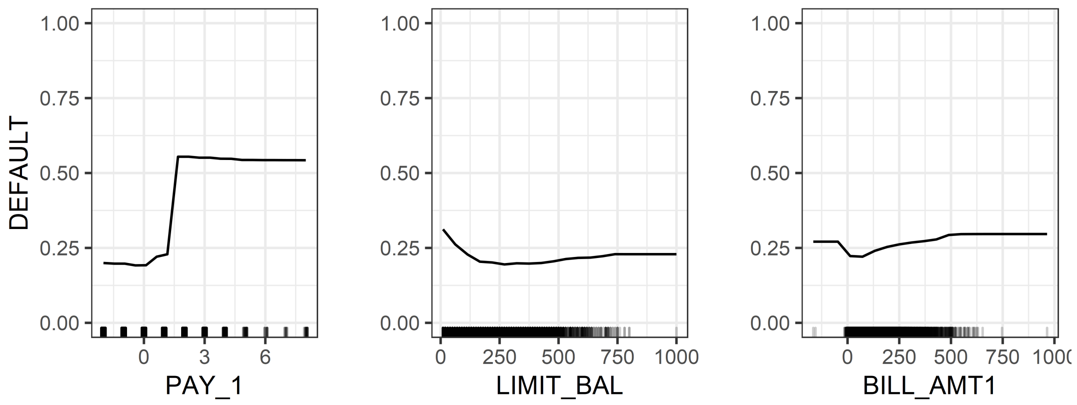


Figure 4.3: The partial dependence plots reveal the average impact of various values for `PAY_1`, `LIMIT_BAL` and `BILL_AMT1` on the default prediction.

predicted default probability soars. The graphs for both `LIMIT_BAL` and `BILL_AMT1` behave in a similar way with `LIMIT_BAL` being more smooth. Both decline rapidly to their minimum before their predicted probability slowly increases.

We focus on PDP and feature importance in the study if there are no other algorithm-specific interpretation methods available.

Chapter 5

Application

In this chapter, we apply seven different models and algorithms for credit risk modelling. For each algorithm, we first introduce the method, give a definition, and elaborate our tuning approach. The aim is to create several learners of each method with different hyperparameter, apply them to the data set and evaluate their performance locally[13]. Finally, we assess the interpretability of the algorithm and interpret a learner. In chapter 6, we compare the best learners of each algorithm to find the globally best classification method.

We only assume that the data is independent and identically distributed drawn from an unknown multivariate distribution (Breiman 2001b).

5.1 Generalized Linear Model (GLM)

We consider generalized linear models as benchmark models for our study since they are still most commonly used for consumer credit risk evaluation in retail banking (Bischl et al. 2016).

5.1.1 Definition

GLMs group various regression approaches which assume that the effect of weights can be modelled through a linear predictor η_i, $i = 1, \dots, n$, while the response does not necessarily have to follow a normal distribution (Fahrmeir et al. 2013). A classical linear model calculates the response directly by $y_i = x_i'\beta + \varepsilon_i$, whereas GLMs apply a link function $g(\cdot)$, thus $g(y_i) = x_i'\beta + \varepsilon_i$. For binary regression, the sigmoid link function

[13]We display the results of four learners. The selection of learners to be displayed is based on their performance, potential abnormalities, and comparability considerations.

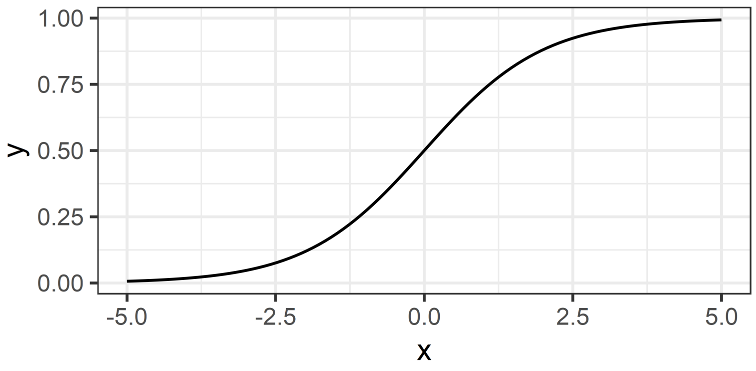


Figure 5.1: Logistic function, the response function $h(\cdot)$ of the logistic regression model.

relates the linear predictor $\eta_i = x_i'\beta$, with features $x_i = (1, x_{i1}, \dots, x_{ik})'$ and weights $\beta = (\beta_0, \beta_1, \dots, \beta_k)'$, to the label $y_i \in \{0, 1\}$ (McCullagh 1984).

We are interested in the probability of $Y = 1$, e.g. the default of a customer,

$$P(Y = 1|X = x_i) = E(y_i) =: \pi_i, \tag{5.1}$$

with $\pi_i \in [0, 1]$. To ensure this constraint without imposing restrictions on the parameters β, we introduce a cumulative distribution function (CDF)

$$\pi_i = h(\eta_i) = h(\beta_0 + \beta_1 x_{i1} + \dots + \beta_k x_{ik}), \tag{5.2}$$

where $h(\cdot)$ is called the response or inverse link function and $h = g^{-1}$. Hence

$$\eta_i = g(\pi_i), \tag{5.3}$$

with $g(\cdot)$ the link function.

Logistic and probit are the most widely used binary regression models and will be examined in the following sections. By introducing a threshold, we can use these regression models for binary classification problems (see section 3.2).

Logistic Regression

The logistic regression was introduced by Berkson (1944) and is widely used for credit scoring. It is easy to implement, stable against outliers and simple to explain (Fahrmeir et al. 2013).

The link function $g(\cdot)$ of the logistic regression is the logit function $\text{logit}(x) = \ln \frac{x}{1-x}$, the quantile function of the logistic function, hence

$$g(\pi_i) = \mathrm{logit}(\pi_i) = \ln \frac{\pi}{1-\pi} = \eta_i, \tag{5.4}$$

where $ln(\cdot)$ is the natural logarithm. The response function $h(\cdot)$ is the logistic function (see figure 5.1) and denoted by

$$h(\eta_i) = \frac{exp(\eta_i)}{1+exp(\eta_i)} = \frac{1}{1+exp(-\eta_i)} = \pi_i. \tag{5.5}$$

Probit

The probit model was originally proposed by Bliss (1934) and further developed by Finney & Tattersfield (1952). It is similar to logit models but preferred if data is normally distributed.

The link function $g(\cdot)$ is the inverse cumulative distribution function (CDF) of the standard normal distribution, the probit function $\Phi^{-1}(\cdot)$. Thus, we have

$$g(\pi_i) = \Phi^{-1}(\pi_i) = \eta_i \tag{5.6}$$

for the link function, and

$$h(\eta_i) = \Phi(\eta_i) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\eta_i} exp(-\frac{1}{2}t^2)\mathrm{d}t = \pi_i \tag{5.7}$$

for the response function $h(\cdot)$.

5.1.2 Tuning

Tuning as defined here is different for GLMs than for machine learning algorithms in one crucial point: GLMs have no hyperparameters. However, threshold tuning works like for machine learning and is conducted for each learner.

Additionally, we conduct feature selection in order to find the most important features for the model calculation. This is done by sequential backwards search with AUC as performance measure, i.e. starting from a model with all features we remove in each step the feature which reduces the AUC the least. We stop when the AUC decrease is less than $\alpha = 0.01$.

This results in four GLM learners to be compared to each other: logistic regression, logistic regression with feature selection based on AUC, probit, probit with feature selection based on AUC.

Learner	**AUC**	**ACC**	**BAC**	**Brier**	**KS**	**Runtime**	**Threshold**
LogReg	0.7233	0.8174	0.6566	0.1449	0.3759	1	0.4056
LogReg AUC	0.7218	0.8108	0.6055	0.1458	0.3850	132	0.4263
Probit	0.7224	0.8171	0.6553	0.1462	0.3725	1	0.4022
Probit AUC	0.7217	0.8017	0.5755	0.1473	0.3820	131	0.5935

Table 5.1: Overview of performance of generalized linear models. The first two models apply the logit function as link function, the last two the probit function. For each model, the results without and with feature selection via AUC are displayed.

5.1.3 Performance

Table 5.1 gives an overview of the performance of GLMs. GLM with logit as link function and no features selection (LogReg) produces the best results – for all performance measurements except KS. The best discriminative power according to KS has the logistic regression with feature selection.

Feature selection produces for both models inferior performance than the respective model without according to all measures but KS: The discriminative power can be improved by applying feature selection. The selection process comes with a rise in runtime: While the models without feature selection are calculated within one minute, the application and evaluation of feature selection takes more than two hours.

5.1.4 Interpretability

GLMs are commonly known as highly interpretable on a modular level (cf. Fahrmeir et al. 2013). The weights and the distribution of the features explain how parts of the model influence the prediction. One can write down a formula with the estimated weights which can be used to make predictions for new instances. This enables the interpretation of a sparse linear model on a global level, albeit the impact of a single feature can only be interpreted under the premise that all other features stay constant.

Table 5.2 presents weights[14] (β-coefficients), standard errors, z values, the Wald statistic[15], and the results of the significance test for the logistic regression model. We have to take all features into account to get an accurate and legitimate interpretation.

[14]The displayed weights are log odds (see equation (5.4)). In order to obtain odds, one needs to take the exponential. The predicted probability for a specific instance is calculated by equation (5.5). Therefore, all feature values and weights are taken into account.

[15]The Wald statistic tests the hypotheses $H_0 : \beta = 0 \; vs. \; H_1 : \beta \neq 0$ and is calculated by $2 \times \Phi\left(\frac{-|\hat{\beta}|}{se(\hat{\beta})}\right) \leq \alpha$, with $\Phi(\cdot)$ the CDF of the normal distribution, $\hat{\beta}$ the estimated weights (β-coefficients), $se(\cdot)$ the standard error, and α the confidence level.

Feature	**Weight**	**Std. Error**	z **Value**	**Pr**$(\| > z\|)$	**Significance**
Bias term	-2.150e+00	5.227e-01	-4.114	3.89e-05	***
AGE	6.284e-03	1.835e-03	3.424	0.000616	***
BILL_AMT1	-5.520e-06	1.137e-06	-4.857	1.19e-06	***
BILL_AMT2	2.389e-06	1.505e-06	1.587	0.112545	
PAY_1	5.770e-01	1.770e-02	32.605	$<$ 2e-16	***
$\vdots$					

Table 5.2: Weights, standard errors, z values, and Walt statistics of some features of the logistic regression model. The three asterisks (***) indicate a significant feature according to the Wald statistic with $\alpha = 0.001$

However, to get an idea of the approach and for greater clarity, we only present some selected features (see table A.1 in appendix A for are complete list of features). We see, for example, that the older the customer the higher the predicted probability of default (with all other features unchanged): for every year older the odds for default increase in average by factor $e^{0.0063} = 1.0063$. Furthermore, for each additional month the customer is in default (`PAY_1`), we expect the default odds to increase by factor 01.7807.

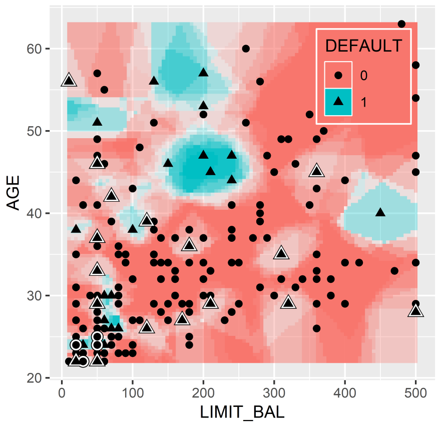
The interpretation of the formula is only possible if the number of features is small. For hundreds or thousands of features, the approach is not longer feasible, and other interpretation techniques might be preferred. Therefore, feature importance of the logistic regression model (figure A.1) and the partial dependence plots for `PAY_1`, `BILL_AMT1`, and `PAY_AMT1` (figure A.2) can be found in appendix A.

5.2 k-Nearest Neighbours (kNN)

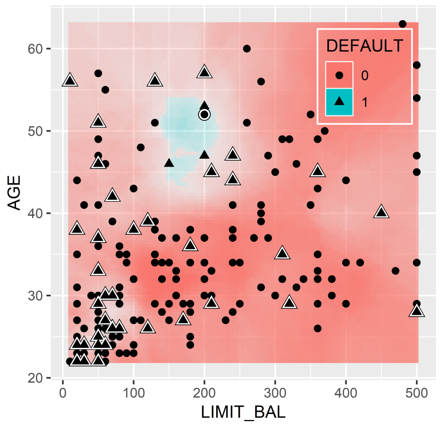
As a second algorithm, we introduce k-nearest neighbour, a pattern recognition approach for classification.

5.2.1 Definition

The k-nearest neighbours (kNN) algorithm uses the average response of the closest k observations in the training set to predict the outcome of new instances. Figure 5.2 shows graphically for the two-feature-case how the kNN splits the feature space into different regions according to the five or 20 closest instances. Friedman et al. (2001b)



(a) Plot for $k = 5$ with $mmce_{train} = 0.1100$ and $mmce_{test} = 0.2650$.



(b) Plot for $k = 20$ with $mmce_{train} = 0.2200$ and $mmce_{test} = 0.2450$.

Figure 5.2: k-nearest neighbours plots for two features (`AGE` and `LIMIT_BAL`) with different values for hyperparameter k.

calculate the predicted outcome $\hat{Y}$ by

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i \, , \tag{5.8}$$

where $N_k(x)$ denotes the neighbourhood of x defined by its k closest points x_i in the training sample. The input data needs to be normalised in order to achieve good prediction quality (cf. section 3.1.2). To define closeness, we need a metric to measure distance. We use the Minkowski distance with $p = 2$ (Schliep & Hechenbichler 2016), which is the Euclidean distance, thus

$$\mathrm{d}(x^{(l)}, x^{(m)}) = \sqrt{\sum_{j=1}^{k} \left(x_j^{(l)} - x_j^{(m)} \right)^2} \, . \tag{5.9}$$

If the metric determines several observations simultaneously as the ith nearest neighbours, the tie will be broken randomly.

We deploy the k-nearest neighbours implementation of the `kknn` package (Schliep & Hechenbichler 2016).

Learner	**AUC**	**ACC**	**BAC**	**Brier**	**KS**	**Runtime**	**Threshold**
kNN 1	0.6671	0.7794	0.5227	0.1619	0.2415	14	0.5071
kNN 2	0.7009	0.7958	0.5832	0.1695	0.2984	2	0.7353
kNN 3	0.7569	0.8119	0.6523	0.1399	0.3935	23	0.4465
kNN 4	0.7455	0.8033	0.6244	0.1440	0.3695	24	0.5134

Table 5.3: Overview of performance of k-nearest neighbours. kNN 1's data is not scaled, kNN 2 is without hyperparameter tuning, kNN 3 performs tuning on scaled data, some class levels are joint for kNN 4.

5.2.2 Tuning

k-nearest neighbours has one hyperparameter to be tuned: k. It determines the number of neighbours used for prediction (Schliep & Hechenbichler 2016). The optimal hyperparameter prevents the algorithm from over- or underfitting like illustrated in figure 5.2. Although, the training error $mmce_{train}$ of figure 5.2b is larger than of 5.2a, the test error $mmce_{test}$ – which is the error on unseen data – is smaller, hence, 5.2b generalises better and performs better on unseen data. The large difference between training and test error of 5.2a is an indicator overfitting. I.e. k too small (figure 5.2a) leads to overfitting and reduction in generalisation, hence, reduction in prediction ability. k too large leads to underfitting, i.e. too much generalisation, hence, to reduction in prediction ability as well. The search for the optimal k is essential for the prediction quality.

The vital search range was learnt by exhaustive search, the step range is a compromise on runtime. We will present the results for a learner item without normalisation and $k \in \{90, 93, \dots, 118\}$ (kNN 1), and a learner without tuning (kNN 2) which adopts the default settings $k = 7$ of the `kknn` package. For kNN 3, we tune k on $\{90, 93, \dots, 118\}$, and for kNN 4, we conduct tuning like for kNN 3 with joint class levels for `PAY_`x.

5.2.3 Performance

Table 5.3 reveals the tremendous performance enhancement achieved by scaling the data before applying algorithms: kNN 1 uses data without normalisation and produces the worst prediction results. Furthermore, the impact of the optimal hyperparameter k for the performance is clearly visible. kNN 2 does not allow hyperparameter tuning and looks always at the seven nearest neighbours for prediction, while kNN 3 searches for the optimal k for each training subset and uses it for prediction in the test subset. The tremendous improvement of kNN 3 clarifies the major impact of tuning in performance. kNN 4 shows that the loss of information by joining class levels of `PAY_`x results in less

ID	LIM.	SEX	EDU.	MAR.	AGE	PY1	PY2	PY3	PY4	...	DEF.
42	70000	1	1	3	25	0	0	0	0	...	0
29268	70000	0	2	2	26	0	0	0	0	...	0
12185	70000	1	2	2	32	0	0	0	0	...	0
21833	70000	1	1	3	27	0	0	0	0	...	0
14231	70000	1	3	2	47	0	0	0	0	...	1
27163	70000	0	2	3	31	0	0	0	0	...	0

Table 5.4: Features for the five nearest neighbours of customer number 42. For the distance calculation, the values have been normalised, the displayed values are the original ones.

accurate predictions and less discriminatory power for the given data set.

kNN 3 outperforms the other learners in all performance measures except runtime: The time needed for tuning and normalisation is clearly visible in table 5.3.

5.2.4 Interpretability

Since no global model or parameters are created, k-nearest neighbours is not interpretable on a global or modular level, respectively. It produces example-based predictions, thus, we can explain the prediction only for a particular instance.

It is possible to display the k nearest neighbours of any customer. These can help to understand, why the algorithm comes up with a certain prediction. For example, the five nearest neighbours of customer with $ID = 42$ are displayed for several features in table 5.4. These are the customers which have the smallest distance (in a $n = 23$ dimensional space), hence, are the most similar to customer number 42. Based on them, we would predict “non-default” (indicated by 0) as response which is also the true response of the customer.

This method makes the decisions of the algorithm transparent and easy to understand. Nevertheless, if there are many features it is not easy to see the proximity of two instances or to represent the neighbourhood. Furthermore, it is hardly possible to see at a glance what one needs to change in order to alter the predicted response, i.e. what a customer has to do to get a credit, for example.

There are feature importance (figure A.3) and partial dependence plots of `PAY_1`, `LIMIT_BAL`, and `SEX` (figure A.4) in appendix A.

5.3 Classification and Regression Tree (CART)

In this section, we introduce decision trees which can function as basis for more sophisticated approaches discussed in the subsequent sections. Simple tree-based methods are easy to interpret yet powerful prediction tools.

5.3.1 Definition

Classification and Regression Trees (CART) were introduced by Breiman, Friedman, Stone & Olshen (1984). They split the feature space into sets of rectangles and then fit constant models to each of them (Friedman et al. 2001b).

Decision trees like CART are usually drawn upside down (see figure 5.3 for examples): the leaves – also referred to as terminal nodes – are at the bottom of the tree while the (parent) node representing the full data set is at the top. At each node, instances satisfying the condition are assigned to the left branch, the others to the right one. The connection between two nodes is called branch. The terminal nodes of the tree correspond to the rectangle regions of the feature space (Kruppa et al. 2013).

With each split, we want to reduce the node impurity as much as possible. We use the `rpart` package (Therneau & Atkinson 2018), which measures the impurity of node m with K classes by the Gini index[16]:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{5.10}$$

with

$$\hat{p}_{mk} = \frac{1}{n_m} \sum_{x_i \in R_m} I\{y_i = k\}, \tag{5.11}$$

where R_m is the region represented by the node m with n_m instances, I is the indicator function for class k. $\hat{p}_{mk}$ can be interpreted as the proportion of observations from class k in node m, thus, small values for the Gini index indicate pure splits.

If new splits would not decrease the impurity of a node by a stopping criterion α, no further splits are attempted. The final prediction is then defined by the most frequent class in a terminal node. The resulting trees have a high variance, i.e. small changes in the training data lead to extremely different trees **??**.

So, CARTs tend to overfit. The best strategy to prevent overfitting is to grow large

[16]The AUC can be converted into the Gini index by: $Gini = 2 \cdot AUC - 1$.

Learner	AUC	ACC	BAC	Brier	KS	Runtime	Threshold
CART 1	0.6436	0.8196	0.6436	0.1461	0.2873	1	0.3819
CART 2	0.6936	0.8195	0.6504	0.1408	0.3728	64	0.4232
CART 3	0.6937	0.8201	0.6557	0.1406	0.3728	9	0.5095
CART 4	0.6999	0.8148	0.6559	0.1424	0.3726	9	0.4994

Table 5.5: Overview of performance of classification and regression trees (CART). CART 1 is without tuning, CART 2 and 3 differ in their search spaces, and CART 4 merge class levels.

trees at first, and then prune in order to find the optimal tree size **??**. Pruning removes branches that cause only minimal changes (cf. stopping criterion α) in the error function when removed.

5.3.2 Tuning

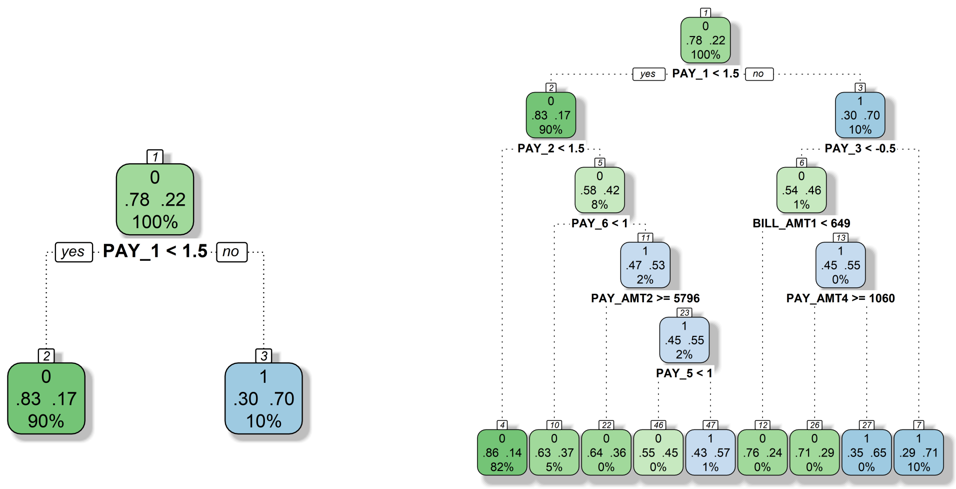
CART offers several hyperparameters to control the tree. Since it tends to overfit, finding the optimal hyperparameter set is essential for good prediction quality (Friedman et al. 2001b). According to Therneau & Atkinson (2018), the main tuning parameters are:

- `minsplit`: the minimal number of instances needed in a node to attempt a split
- `cp`: each split must decrease the error by cost complexity parameter α
- `maxdepth`: the maximum depth of any node of the final tree

All described hyperparameters are tuned (when applicable). We will present the results for the following four learners: CART 1 without any hyperparameter tuning, CART 2 with all hyperparameter tuned on large search spaces, while CART 3 is tuned like CART 2 but on tighter search spaces, and CART 4 joins class levels for PAY_*x* and is tuned like CART 3.

5.3.3 Performance

The impact of tuning on the performance of CART is clearly visible: CART 1, which is built without hyperparameter tuning, computes the results much faster but with considerable less prediction power than the other learners (see table 5.6) – except based on MMCE where it achieves the second best results. Overfitting is likely to be the reason why the more flexible CART 2 is outperformed by CART 3, which has smaller



(a) CART 1

(b) CART 2

Figure 5.3: Decision trees of CART algorithm with different hyperparameter settings for maximal depth and number of nodes.

search grids and less tuned hyperparameters. The different size of the search space is also reflected in the runtime: CART 3 is seven times faster than CART 2. The best learner is CART 4, which combines the hyperparameter search space of CART 3 with merging class levels of the features `PAY_`x (see section 3.1.2).

The results of CART learners are not very distinct. The question of the best leaner is answered according to which performance measurement is chosen: Based on AUC and BAC, CART 4 is doing best, but CART 3 exceeds in the other measures.

5.3.4 Interpretability

A sparse classification and regression tree is directly interpretable for humans on a global and on a local level. It is possible to visualise the entire model and its decision-making process by a simple two-dimensional graphic. The concept of CART is easy to understand, even to laymen, and one can see at a glance why the algorithm predicts a certain output. Furthermore, it is straightforward to see what needs to be changed in order to obtain a different classification.

Figure 5.3 shows the decision trees of CART 1 and 2 (see figure A.5 in appendix A for plots of CART 3 and 4). The sparse tree on the left (figure 5.3a) has only one node and a depth of zero, thus, it is very easy to interpret: For all customers with a value for PAY_1 smaller than 1.5 – i.e. customers which are one month due, inactive, use a revolving credit, or paid in full – we predict non-default, while we expect all others to default. The tree in figure 5.3b has eight nodes and a maximal depth of four, and is already more complicated to interpret, but still, interpretation is possible in a convenient way. It is obvious that the more nodes a decision tree has the harder it is to interpret it.

Figures A.6 and A.7 in appendix A display feature importance and partial dependence for the best performing learner CART 4.

5.4 Random Forest

The ensemble method random forest overcomes the high variance of CART by building many decision trees simultaneously.

5.4.1 Definition

Ho's (1995) idea of bootstrap aggregation – or bagging – creates several bootstrap[17] samples of the training set, fits a separate decision tree to each sample, and then aggregate the trees' prediction by majority vote to get a single predictive model (Breiman 1996). The idea is to reduce the variance of many noisy but nearly unbiased learners by combining independent trees.

Breiman's (2001a) random forest further lowers the variance of bagging by introducing a second layer of randomness: At every split, we randomly select m features and consider only these m features as splitting criteria (Breiman 2001a). This eventuates in independent and identically distributed trees and further reduces the correlation between them.

We use the `randomForest` package (Liaw & Wiener 2002), which provides an `R` interface to Breiman's (2001a) original `Fortran` programme code. The algorithm can be summarised as follows:

1. Draw n bootstrap samples from the data set
2. Grow an unpruned CART for each bootstrap sample where, chose at each node the best split among a random sample of m features
3. By aggregating the predictions of the n trees via majority vote, predict on new data

[17]Bootstrap denotes drawing randomly with replacement a sample with the same size as the original set.

Learner	AUC	ACC	BAC	Brier	KS	Runtime	Threshold
Random Forest 1	0.7237	0.8210	0.6544	0.1600	0.3964	66	0.4734
Random Forest 2	0.7230	0.8208	0.6551	0.1615	0.3991	288	0.4713
Random Forest 3	0.7671	0.8179	0.6513	0.1369	0.4066	284	0.5274
Random Forest 4	0.7256	0.8211	0.6545	0.1602	0.3959	1035	0.4794

Table 5.6: Overview of performance of random forest. Random Forest 1 and 2 are applied on different search spaces, for Random Forest 3 more trees are grown, and Random Forest 4 combines the tuning settings of Random Forest 1 and 3.

5.4.2 Tuning

According to Breiman (2001a), random forest is robust against overfitting when the number of trees is sufficiently large enough. There are a lot of theoretical considerations about "sufficiently large enough", which may depend on the concrete data situation. Furthermore, James et al. (2013) suggest for the optimal number of features drawn at each split $m \approx \sqrt{p}$, with p the total number of features.

We want to follow Breiman's (2001b) idea and "let the data speak", thus we try to find the optimal hyperparameters for our data set by tuning. We will examine the following hyperparameters (Liaw & Wiener 2002):

- `ntree`: the number of trees that are grown
- `mtry`: the number of randomly selected variables at each node
- `maxnodes`: the maximal number of terminal nodes of each tree

Several hyperparameter sets are trained and assessed as well as joint class levels for `PAY_x`. Since the joint class levels lead to inferior results, we omit their results and present the following learners: mtry and maxnodes are tuned for Random Forest 1, then the same are tuned on a larger search space, for Random Forest 3 ntree is tuned, Random Forest 4 tunes all three parameters.

5.4.3 Performance

The prediction performance of Random Forest 1, 2 and 4 are quite similar. The expansion of the search space of learner 2 compared to 1 are reflected in higher BAC and KS but lead to worse performance according to AUC, ACC, Brier and runtime. The increase of the number of trees grown (Random Forest 3) has a tremendous impact on the performance: while ACC and BAC get worse, AUC and Brier improve dramatically, thus Random

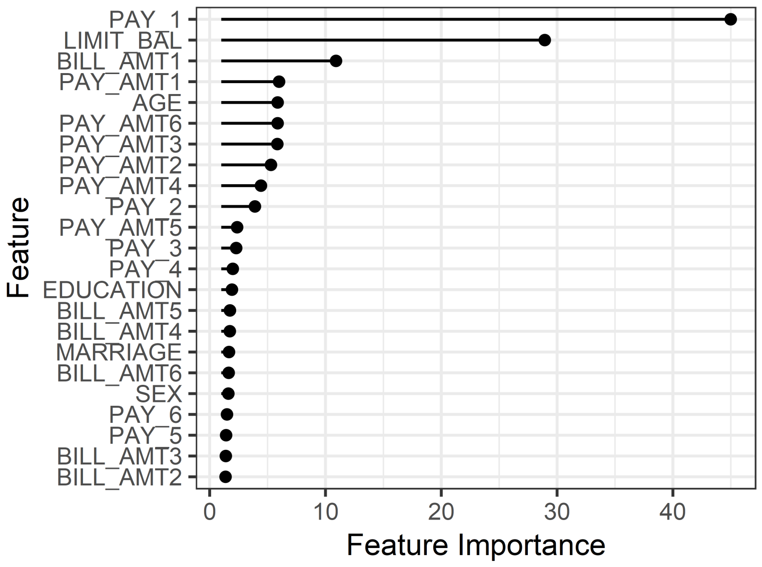


Figure 5.4: Feature importance of Random Forest 3.

Forest 3 is the best learner. Obviously, the combination of tuning `mtry` and `maxnodes` and increasing the number of tress `ntree` for learner 4 does not lead to better results.

With default settings, i.e. no hyperparameter but threshold tuning, and 10-fold cross-validation in both the inner and the outer resampling loop, 50,000 trees needs to be grown with randomly sampling instances and features. For example, Random Forest 4 grows $5,040,000$ trees and needs therefore more than 17 hours. The larger search space for learner 2 compared with 1 results in a 10 times longer runtime. This empahsises the huge computationally burden of random forest.

5.4.4 Interpretability

Random forest improves the prediction accuracy of CART at the expense of interpretability. The algorithm is more complex and consists of hundred or even thousands of trees. Although it might theoretically be possible to draw these trees on a piece of paper, the interpretability for humans suffers tremendously. Since the trees are not based on the same data set but on different bootstrapp samples, and each splits considers only a random selection of features as splitting criteria, the decision-making process is very complex. Thus, gaining a full understanding is not possible for humans and random forest is treated as a black box (James et al. 2013).

There are no feasible methods to make the decision making process directly transpar-

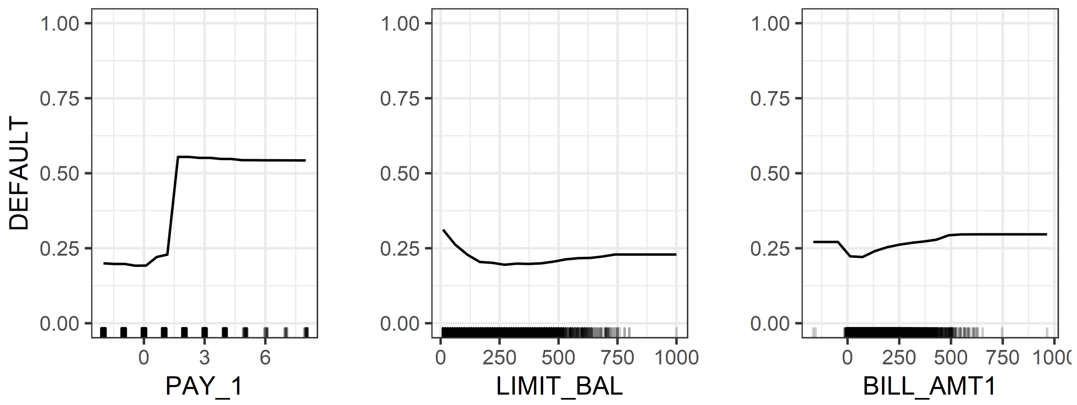


Figure 5.5: Partial dependence plots of the three most important features (according to figure 5.4) of Random Forest 3.

ent. Nevertheless, model-agnostic approaches can help to understand why the algorithm comes up with a certain classification, and give insights which features contribute to the prediction. Figure 5.4 displays the feature importance of Random Forest 3. Similarly to previously analysed algorithms, `PAY_1` impacts the prediction the most. Moreover, `LIMIT_BAL` and `BILL_AMT1` reveal a large importance.

Feature importance does not reveal the direction and extent of the impact. We further examine the three most important features by looking at their partial dependence plot in figure 5.5. The jump in predicted default probability for customers with `PAY_1` ≥ 2 is tremendous. The predicted probability increases from about 0.20 to more than 0.50. The change in probability when altering the values for `LIMIT_BAL` and `BILL_AMT1`, is not as distinct. Nevertheless, one can clearly see a decline at the beginning, then a rising score.

5.5 Gradient Boosting Machine (GBM)

While random forest grows many trees simultaneously, gradient boosting machines grow them successively.

5.5.1 Definition

Gradient boosting machines (GBM) were introduced by Breiman (1997) as powerful learning ideas for classification problems, and later extended by Friedman (2001a) to regression as well. A slightly different and quite famous approach of GBM is the adaptive boosting (AdaBoost) by Freund & Schapire (1997), which will not be considered in this paper. GBM is also called gradient tree boosting or gradient boosted regression tree.

The basic idea is to combine many "weak" classifiers to produce a powerful prediction algorithm (James et al. 2013). We start with a sparse decision tree fitted to all observations and analyse the misclassification. Then, we fit sequentially new trees to improve our prediction in areas where the previous tree does not perform well. Thus, the trees depend strongly on the trees grown in the iterations before. In boosting, successive trees give extra weight to points incorrectly predicted by earlier predictors.

In this paper, we deploy the gradient boosting algorithm implemented in Chen & Guestrin's (2016) `xgboost` package.

`Xgboost` fits sequentially K additive functions to predict the label:

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F}, \tag{5.12}$$

with $\mathcal{F}$ the space of CART defined as

$$\mathcal{F} = \{f(\mathbf{x}) = w_{q(\mathbf{x})}\}, \tag{5.13}$$

where $q : \mathbb{R}^m \to S$ the decision rule, S the number of leaves in the tree, and $w_s \in \mathbb{R}^S$ the weight of the sth leaf. The decision rule q describes the tree structure which maps an instance to the corresponding leaf. The algorithm uses the decision rule q_k of the kth tree to classify the instance and calculates the final prediction by summing up the S weights w_s.

The algorithm adds greedily the f_k which improves the model most, based the first $k-1$ trees built.

The optimal weight w_s^* of leaf s is calculated by

$$w^* = -\frac{\sum_{i \in I_s} g_i}{\sum_{i \in I_s} h_i + \lambda}, \tag{5.14}$$

where $g_i = \partial_{\hat{y}^(t-1)}\, l\left(y_i, \hat{y}_i^{(t-1)}\right)$ and $h_i = \partial^2_{\hat{y}^(t-1)}\, l\left(y_i, \hat{y}_i^{(t-1)}\right)$ the first and second order gradient statistics of a loss function l.

5.5.2 Tuning

Gradient boosting machines tend to overfit but `xgboost` has implemented several protective measures to prevent overfitting (Chen & Guestrin 2016). Due to computational

Learner	AUC	ACC	BAC	Brier	KS	Runtime	Threshold
GBM 1	0.6436	0.8195	0.6436	0.1976	0.2873	1	0.5567
GBM 2	0.7439	0.8196	0.6496	0.1934	0.4022	21	0.4969
GBM 3	0.7781	0.8212	0.6571	0.1347	0.4292	33	0.4954
GBM 4	0.7748	0.8139	0.6561	0.1371	0.4212	32	0.4972

Table 5.7: Overview of performance of gradient boosting machine. GBM 1 applies only trees with one single node (additive model). 2 and 3 tune different hyperparameters, while GBM 4 combines 3 with merged class levels for `PAY_x`.

considerations, we focus on the following hyperparameters:

- `max_depth`: the maximum depth of a single tree
- `min_child_weight`: the minimum sum of instance weight in each node
- `gamma`: minimum loss reduction required by splits
- `nrounds`: maximum number of iterations of the data set

We train several learners and present the performance results of four. The first learner (GBM 1) consists of trees with only one node each, which results in a generalised additive model. GBM 2 tunes maximum depth, minimum sum of weights, and minimum loss reduction, whereas `max_depth`, `gamma` and `nrounds` are tuned for GBM 4. The last displayed learner combines the hyperparameter set and search space of 3 with merged class levels form`PAY_x`.

5.5.3 Performance

Table 5.7 shows the performance of gradient boosting machine. The results reveal that the additive model performs poor according to AUC and KS, but astonishing well according to ACC and BAC, Brier – and it is very fast. The impact of hyperparameter tuning is clearly visible: GBM 2 outperforms the additive model in every measure, the improvements according to AUC and KS are tremendous. The enhancement by tuning `nrounds` boosts the performance of GBM 3 and 4 compared to 2: GBM 3 is the best gradient boosting machine learner. Due to the information loss by joining class levels, GBM 4 obtains the second place.

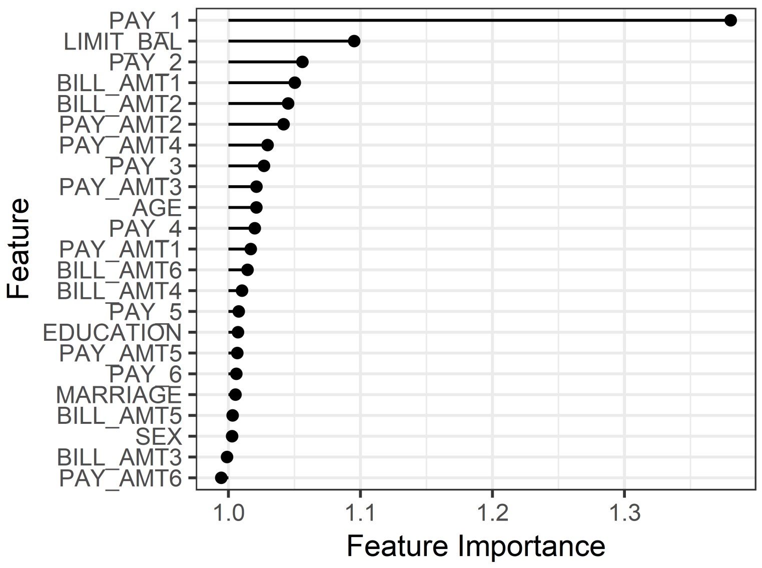


Figure 5.6: Feature importance of GBM 3.

5.5.4 Interpretability

GBM offers great prediction performance. This is achieved by boosting, hence, many decision trees are build depending on the previous ones. The knowledge learned by the algorithm is difficult to understand, since the final prediction is calculated by weighted classification based on previous trees. Due to this high dependence between the trees a direct interpretation is not possible.

As described in section 5.5.3, one can restrict the trees to have only one split. The resulting additive models are easier to interpret but their performance suffers from the restriction (see GBM 1 in table 5.7). Additive models can be explained by a model equation similarly to generalized linear models.

Model-agnostic methods, however, provide some interpretability. Figure 5.6 reveals the feature importance of GBM 3. The major importance of `PAY_1` is clearly visible, followed by `LIMIT_BAL` and `PAY_2`. The margin between `PAY_1` and the next important features is remarkable. Substantially, it might be obvious that the payment history of the last month is very important for the question if the customer pays his credit duly this month. Technically, it is interesting that the ensemble method gradient boosting machine has such a unilateral dependence on one feature.

The partial dependence plots in figure 5.7 reveal similar graphics for `PAY_1` and `LIMIT_BAL` as for random forest in figure 5.5. Customers who are delayed for two or

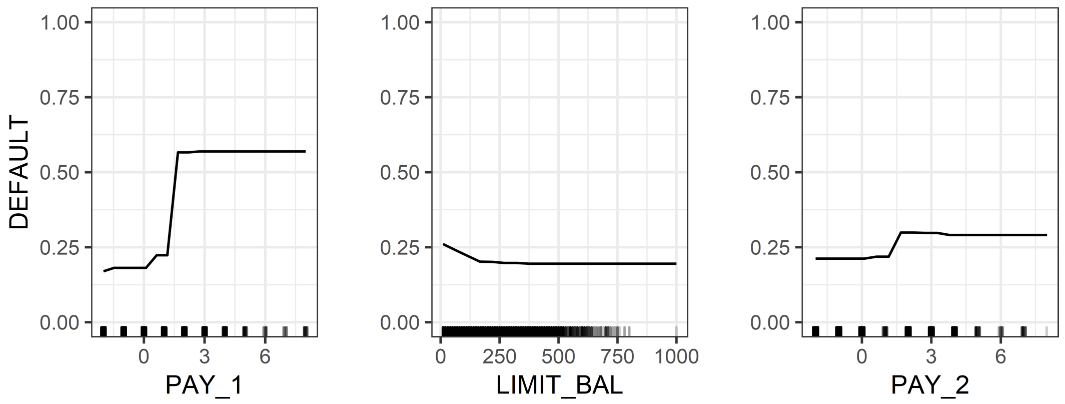


Figure 5.7: Partial dependence plots of the three most important features (according to figure 5.6) of GBM 3.

more months face a tremendously increasing default probability. The impact of `PAY_2` is related to `PAY_1` but to a lower extent.

5.6 Artificial Neural Networks (ANN)

Artificial neural networks, also called artificial neural nets, are probably the most famous machine learning method.

5.6.1 Definition

First introduced by McCulloch & Pitts (1943), an artificial neural network is a learning algorithm inspired by biological neuronal networks working in human brains. Nowadays there are several slightly different ideas of artificial neural networks available. Schmidhuber (2015) provides a comprehensive overview of different settings and approaches.

The units or neurons receive signals and process it via connections (like the biological synapses) to other neurons. The artificial neurons are typically arranged in layers (see figure 5.8). We use the `H2O` (LeDell, Gill, Aiello, Fu, Candel, Click, Kraljevic, Nykodym, Aboyoun, Kurka & Malohlava 2018) package since it offers a very fast and accurate implementation of neural networks in `R`. It is a multi-layer feedforward artificial neural network – also known as deep neural network – with stochastic gradient descent using back-propagation. The central idea is to extract linear combinations of the inputs as

[18]https://en.wikipedia.org/wiki/Artificial_neural_network (20.09.2018)

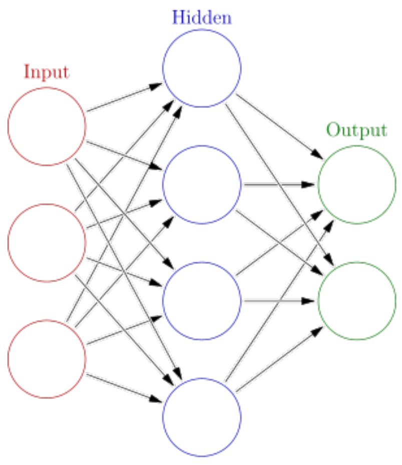


Figure 5.8: Concept of an artificial neural network for binary classification with three input neurons and one hidden layer with four neurons.[18]

derived features, and then model the target as a non-linear function of these features (Friedman et al. 2001b).

There are K units in the output layer on the right for K-classification with the kth unit modelling the probability of class k (Friedman et al. 2001b). The response probability is denoted by $Y_k,\ k = 1, \ldots, K$, which is calculated by a non-linear transformation (for example, by a sigmoid function $\sigma(v) = 1/(1 + e^v)$) form $T_k,\ k = 1, \ldots, K$. T_k is derived from the Z_m in the hidden layer with m units as follows:

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X_l), \tag{5.15}$$

$$T_k = \beta_{0k} + \beta_k^T Z, \tag{5.16}$$

$$f_k(X) = g_k(T), \tag{5.17}$$

where $X_l,\ l = 1, \ldots, L$ denotes the input layer for a training set with L features, and $g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_k}}$ the softmax output function which allows a final transformation of the output. The weights α_m and β_k are set randomly at the beginning and then updated in each iteration according to error minimisation.

5.6.2 Tuning

Artificial neural networks are highly flexible algorithms and are able to capture complex non-linear and non-monotonic structures. Nevertheless, they are prone to overfit (Srivastava, Hinton, Krizhevsky, Sutskever & Salakhutdinov 2014). We apply tuning to

Learner	AUC	ACC	BAC	Brier	KS	Runtime	Threshold
Net 1	0.7682	0.8167	0.6395	0.1374	0.4125	7	0.5766
Net 2	0.7610	0.8181	0.6422	0.1371	0.4121	116	0.5141
Net 3	0.7668	0.8173	0.6418	0.1376	0.4111	240	0.5362
Net 4	0.7635	0.8067	0.6360	0.1404	0.4031	23	0.5226

Table 5.8: Overview of performance of artificial neural networks. Net 1 uses two hidden layers, 2 and 3 allow up to five hidden layers. Net 4 applies two hidden layers to a data set with merged class levels for `PAY_x`.

find optimal hyperparameter sets and focus on the following two (LeDell et al. 2018):

- `hidden`: the number and size of the hidden layers
- `epochs`: iterations of the data set

There are additional hyperparameters that can be tuned but it would exceed the scope of the work and the computational resources to tune more. We display the performance results of four learners. They have two hidden layers (Net 1) or up to five (Net 2), with each up to 100 units. For Net 3, the hidden layers (up to five) and epochs are tuned, while Net 4 is like Net 1 with joint class levels for PAY_*x*.

5.6.3 Performance

Net 1, which uses two hidden layers, delivers the best performance: it has the highest AUC and the shortest runtime. Table 5.8 reveals that increasing the number of hidden layers for Net 2 and 3 does lead to better performance according to ACC and BAC, but not according to the main measure AUC. More hidden layers with more units make the algorithm more flexible but also increases the risk of overfitting. Furthermore, the runtime increases rapidly. Merging the class levels for `PAY_x` reduces the amount of information in the data set and deteriorates the prediction performance of the learner.

5.6.4 Interpretability

An artificial neural network with multiple hidden layers is very flexible but also very complex. It processes inputs in a non-linear fashion, and stores information in weights and connections that are not easily accessible (cf. Friedman et al. 2001b).

In order to make the relationship between inputs and outputs visible, Tzeng & Ma (2005) suggest visualising weights in a Neural Interpretation Diagram (NID). NIDs display a neural network similar to figure 5.8. The width of the connections in the NID

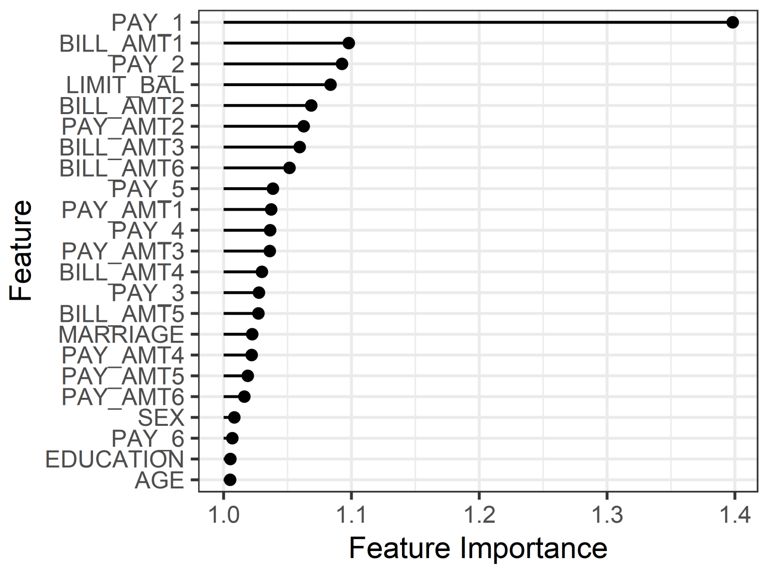


Figure 5.9: Feature importance of Net 1.

are defined by the relative magnitude of the connection weights, the line shading by the direction of the weight. It is obvious that this approach is not feasible for multiple hidden layers with 23 input features and up to 100 neurons per layer.

In the following, we focus on model-agnostic methods. Figure 5.9 gives the feature importance of Net 1. Also for this algorithm, `PAY_1` is the most important feature, followed by `BILL_AMT` and `PAY_2`.

The partial dependence plots (figure 5.10) reveal the direction of their impact. The predicted default probability decreases for `PAY_1` between -2 and 0.5, then soars to the maximum at `PAY_1`$= 3$, before it declines again. The reason for the decline between -2 and 0.5 could be explained by the fact that -2 indicates "no consumption", which obviously leads to a higher predicted default probability than "revolving credit" or "paid duly". Furthermore, the figure reveals the high predicted default probability for customers with `PAY_2` > 3. The other two features show monotonic behaviour: The higher `BILL_AMT` the smaller the default probability, whereas for `PAY_2`, the higher the value the higher the predicted probability.

5.7 Stacking

In this section, we introduce a ensemble method which uses the results achieved so far as inputs.

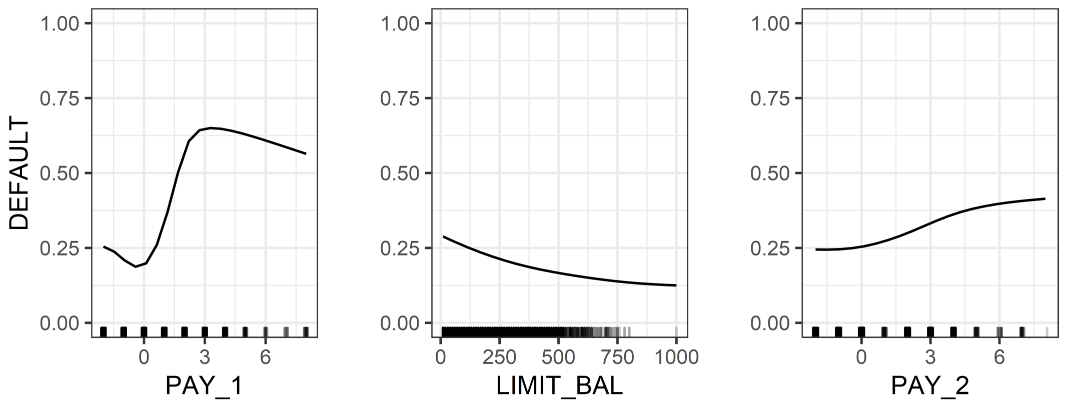


Figure 5.10: Partial dependence plots of the three most important features (according to figure 5.9) of Net 1.

5.7.1 Definition

In the previous sections, we applied several different learning algorithms to the data set and each algorithm calculated a score for each instance. Stacking takes these algorithms as “meta learners”, use their outputs as inputs and aggregates them using another learning algorithm which is called “super learner” (Van der Laan, Polley & Hubbard 2007).

Stacking, together with boosting and bagging, can be summarised as ensemble methods (Liaw & Wiener 2002). Ensemble methods aggregate the results of many classifiers. While boosting (see section 5.5) is deployed to reduce bias and bagging (see section 5.4) to reduce variance, the intention to use stacking is to directly improve prediction accuracy. We apply the first two techniques to many classifiers of the same algorithm (in the paper at hand, to CART). Stacking, however, based on classifiers of different algorithms.

Van der Laan et al. (2007) postulates a theorem to justify theoretically the application of super learners and prove their superior performance under certain conditions. The super learner itself is a prediction algorithm which deploys several learners to the data set and choose the optimal learner – or the optimal combination of learners. Since it is impossible to know a priori which type of classifier performs best for a given real-world problem, the advantages of being able to choose between a set learners are clear and can lead to better performance results (Breiman 2001b).

Learner	AUC	ACC	BAC	Brier	KS	Runtime	Threshold
Stacking 1	0.7820	0.8209	0.6515	0.1344	0.4309	358	0.5374
Stacking 2	0.7823	0.8208	0.6544	0.1344	0.4149	358	0.5903
Stacking 3	0.7821	0.8196	0.6513	0.1346	0.4255	634	0.5313
Stacking 4	0.7829	0.8214	0.6546	0.1340	0.4312	634	0.5368

Table 5.9: Overview of stacking results for LogReg (1 and 2) and Net 1 (3 and 4) as super learners

5.7.2 Tuning

We tune the threshold for stacking. Wang, Hao, Ma & Jiang (2011) suggests that ensemble methods need accuracy and diversity in order to produce good results. There are no hyperparameters in the strict sense, nevertheless, we apply every previously introduced algorithm as a super learner and present the results of the two best: Logit without feature selection (Stacking 1 and 2) and Net 1 (Stacking 3 and 4). Furthermore, we generate two input data sets: One is built of the output of the best learner of each algorithm (Stacking 1 and 3). The other is built of the outputs of the six globally best learners[19], which are *k*NN 3, Random Forest 3, GBM 3 and 4, and Nets 1 and 3 (Stacking 2 and 4).

5.7.3 Performance

Table 5.9 shows the performance of the super learners. The best result is achieved by stacking with an artificial neural network as super learner and the sic globally best learners as meta learners: Stacking 4 is the best learner according to all prediction measures.

The results are quite similar for all learners. The runtime displayed is the total runtime and takes also the runtimes of the meta learners into account. ANN as super learner is twice as time consuming as LogReg.

5.7.4 Interpretability

Stacking adds an extra layer of complexity to the classifier. We have not only the super learner – a neural network – to interpret but also six other learners which deliver the inputs for the super learner. For a comprehensive approach to make the learner more interpretable, we have to take all knowledge and insights about the base learners into

[19]With the constraint, that not more than two learners of the same algorithm are selected.

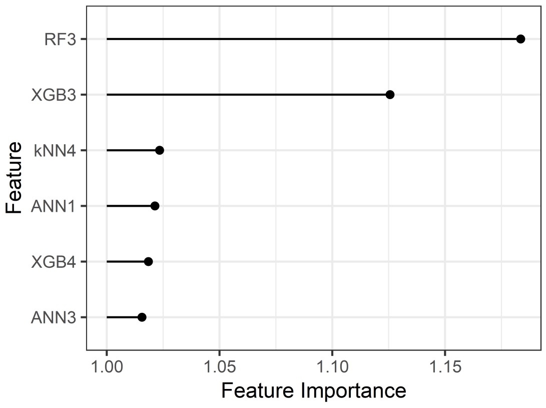


Figure 5.11: Feature importance of Stacking 4.

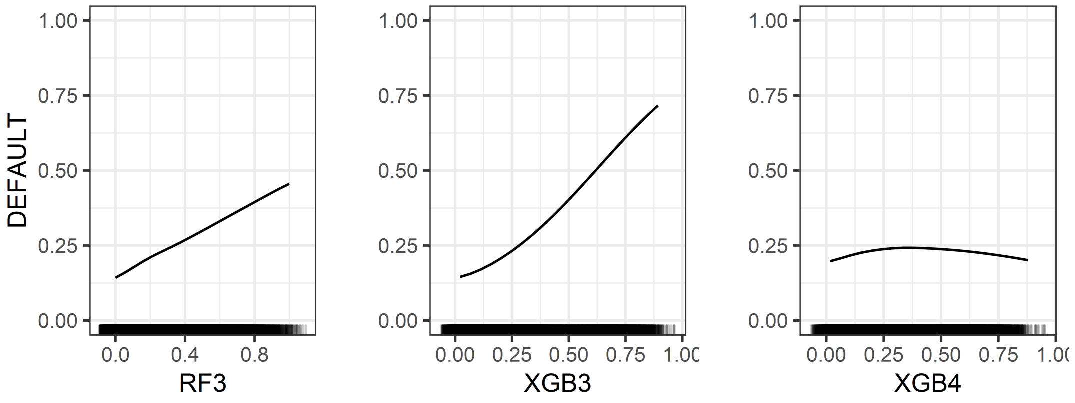


Figure 5.12: Partial dependence plots of the three most important features (according to figure 5.11) of Stacking 4.

account which we created in the previous sections. We focus on model-agnostic methods in order to analyse the impact of the base learners to the super learner's prediction.

The feature importance in figure 5.11 reveals the importance of the base learners. The major importance of random forest (RF3) might be surprising, since the best single base learner is GBM 3 (here denoted as XGB3) followed by Net 1 (see chapter 6). The reason could be the fact that for random forest, there is only one learner considered for this stacking algorithm, whereas there are two gradient boosting machine learners as base learners. Both GBM learners are similar in their decision-making process, so there is some information stored in both learners, and the single learner is no longer that important.

The partial dependence plots give as more information. The plot for Random Forest 3 looks like expected: the higher the predicted score of the base learner, the higher the predicted score of the super learner. For GBM 3, the graphic is similar, although the impact declines for higher input scores. The PDP for GBM 4 gives a different picture: for higher input probabilities the predicted default probabilities of the super learner decrease. The downwards movement of the graph may indicate a correction of (by GBM 4) systematically wrong classified instances. This error correction is a possible explanation for the superior performance of stacking.

Chapter 6

Performance

In this chapter, we present the best learners of each algorithm and compare their results.

Table 6.1 lists the performance results of the algorithms introduced in chapter 5. Stacking with ANN as super learner is the best classifier according to AUC, best non-stacked or single algorithm is GBM. The table reveals that all machine learning algorithms (except for CART) classify better than the benchmark model logistic regression. CART performs very poor compared to the other algorithms but the decision tree idea can be improved tremendously by bagging or boosting. These ensemble methods use the poor performing CART and create powerful prediction tools. Boosting with gradient boosting machine yields better results than bagging in order to create an random forest, while stacking, which combines not only decision trees but different methods, produces the best AUC.

Figure 6.1 shows the ROC curves of the algorithms stated in table 6.1. One can see that stacking is superior to all learners for almost all possible thresholds.

The results also reveal that a higher AUC does not automatically come with higher

Algorithm	**AUC**	**ACC**	**BAC**	**Brier**	**KS**	**Runtime**
Stacking	0.7829	0.8214	0.6546	0.1340	0.4312	372
GBM	0.7781	0.8212	0.6571	0.1347	0.4292	33
Neural Net	0.7682	0.8167	0.6395	0.1374	0.4125	7
Random Forest	0.7671	0.8179	0.6513	0.1369	0.4066	284
kNN	0.7569	0.8119	0.6523	0.1399	0.3935	23
GLM	0.7233	0.8174	0.6546	0.1449	0.3759	1
CART	0.6999	0.8148	0.6559	0.1424	0.3726	9

Table 6.1: Overview of performance results for each algorithm.

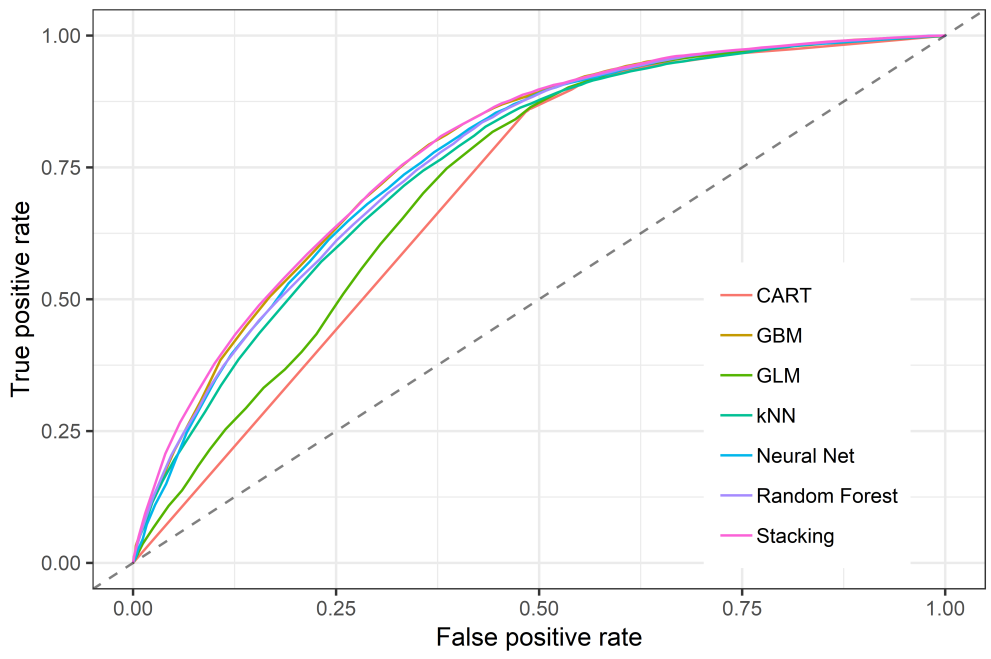


Figure 6.1: ROC curves of the best learner of each algorithm

ACC (see chapter 3.4 for details). If ACC is the main performance measurement, the outcome order would be different. For example, GLM would be ranked higher, whereas Neural Net would lose its third place.

Similar to ACC, BAC is based on the confusion matrix, so the results are similar to each other. The highest BAC achieves GBM, which is also the second best predictor according to AUC. The third best AUC achieves neural net, but the lowest BAC. Both BAC and ACC measure the classification results of the learner.

The Brier score assesses the sufficiency of the predicted probabilities. It gives a similar picture of the achieved performance as AUC. Kolmogorov-Smirnov statistic evaluates the discriminatory power like the AUC. The order of the results would be exactly the same as in table 6.1. Figure 6.2 shows the ECDFs for both Stacking and the benchmark model GLM. For each algorithm, the upper curve is the ECDF of the predicted probabilities of non-default, the lower of default. The largest distance between both curves is marked with a black dotted line. The Kolmogorov-Smirnov statistic measures this distance.

Performance measured as runtime eventuates in a totally different order of the results. Obviously, the runtime of stacked learners is by far the highest. It is calculated as the sum of all underlying meta learners, i.e. every learner except stacking displayed in

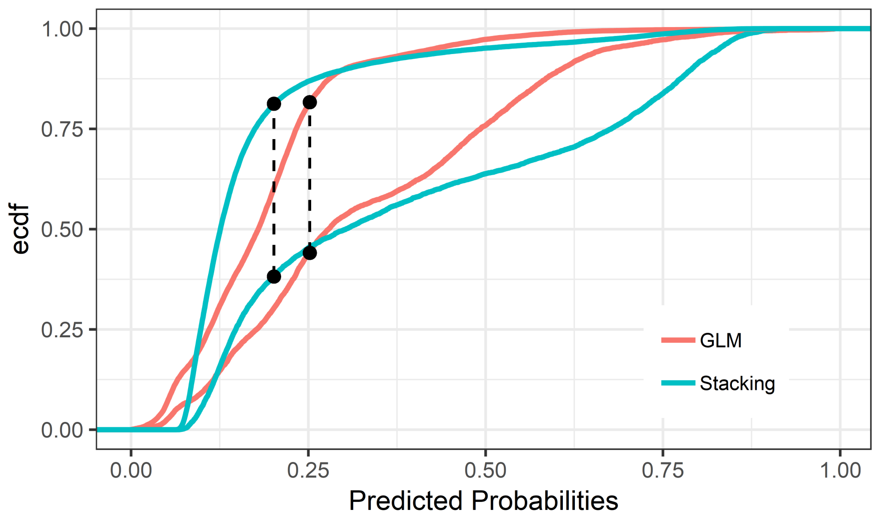


Figure 6.2: Kolmogorov-Sminrov statistic for Stacking and GLM. For both algorithms, the ECDFs for the probabilities of predicted non-defaults (superior) and defaults (inferior).

table 6.1, and additionally the time to aggregate their outputs via ANN. Furthermore, random forest grows many complex trees after newly resampling data and features for each tree, which is very time consuming. On the other hand, the neural net of `h2o` package delivers very fast – despite its complexity – and precise results. One have to keep in mind, that the runtime of the different algorithm are not regardlessly comparable. It heavily depends on search space and number of hyperparameters to be tuned, and the specific implementation of the method. The author did not compare the runtime of different algorithm implementations or different `R` packages in detail, this would go beyond the scope of the work.

Figure 6.3 shows the AUC of each algorithm subject to their runtimes. The figure reveals the two groups of learners: the fast performing ones with runtimes below 25 minutes, and the slow learners which run for more than four hours. The red labelled learners could be considered as efficient learners (cf. the efficient frontier in portfolio analysis (Sharpe 1963)): there are no learners with higher AUC for same or faster runtime or, respectively, there are no faster learners with at least as good predictive ability.

Taken together, stacking achieves the highest AUC, hence can be considered as the best learner, while GLM is the fastest algorithm. The application of machine learning algorithms highly improves the classification performance compared to the benchmark

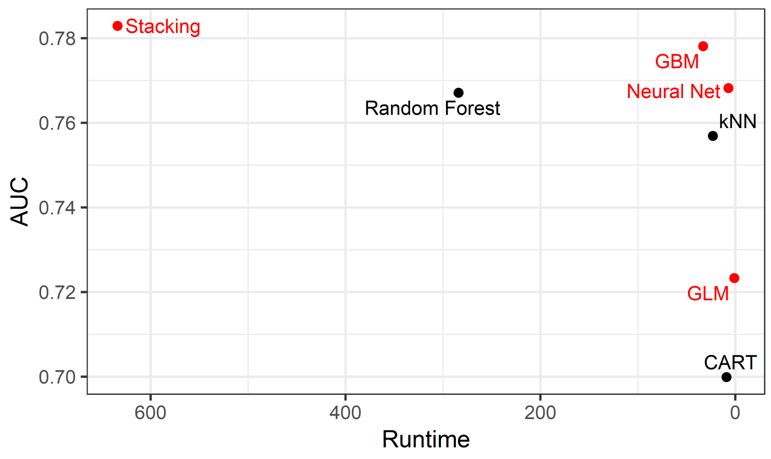


Figure 6.3: Learners' AUC subject to their runtimes. The efficient learners are red labelled.

model GLM. Different performance measures would lead to different "best" algorithms.

Chapter 7

Interpretability

As mentioned before, interpretability of prediction algorithms is a key driver for building trust in the models and for deploying them widely in credit risk. We have seen some methods like feature importance and partial dependence plots that try to make machine learning more interpretable. They bring some light into the black box by providing possible explanations and identifying important features and their influence, and can be applied to all learners described in the paper at hand. Nevertheless, these methods are model-agnostic, hence, they are not suited to give clear insights into the models and do not make the decision-making process traceable. For some algorithms, there are special techniques to get a deeper understanding of "how the model works". We want to summarise the lessons learned in the following.

Classification and regression trees can be considered as highly interpretable. One can draw the exact model with pen and paper as a two-dimensional graphic. The visualisation is fast and easy to understand, even to non-statisticians. Only trees with many branches suffer a loss of interpretability.

Generalized linear models offer weights in order to quantify the impact of single features on the prediction. The exact model can be described by an equation. This is suitable as long as there are not too many features.

For k-nearest neighbour, we can actually display the nearest neighbours of a customer of interest. This can be used as an explanation for a certain prediction. The idea of the algorithm is easy to understand and by printing the neighbours one easily understand the decision-making process for single customers.

Bagging with random forest improves the predictive performance of CART but creates more complex algorithms and reduces the interpretability. Theoretically, it is possible to draw the 500 or 1000 trees of random forest on a piece of paper – but the interpretability for humans might be questionable. Furthermore, the fact, that not every tree is based on the same sample, and not every split on the same feature subset, makes the decision-making process very complex. Thus, direct (feasible) interpretability of the model can be denied. Model-agnostic methods provide possible explanations and give insights into which features contribute to the prediction more than other.

Due to boosting, the trees of GBM highly depend on previously built trees. Since the prediction takes weighted residuals into account, it is not possible (or at least not feasible) to display the model by drawing decision-trees. Although direct interpretation of the model is not possible, model-agnostic models provide some kind of interpretability. A special case of gradient boosting machine is a learner with only one-split-trees. Then, we have an additive model, which can be represented, similarly to generalized linear models, by using a model equation.

Artificial neural networks can indeed be compared to a bunch of generalized linear models. Since they are arranged in several hidden layers and transformed in a non-linear manner, these are very flexible and highly complex, hence, interpretation on model level is not possible. The model-agnostic methods described previously help to build some understanding of the decisions and deliver possible explanations.

Since stacking combine all algorithms described above, interpretability suffers from every black box model which provides input for the stacked model. Model-agnostic models provide (in the setting as used in this paper) information only about which algorithm is more important for a certain prediction, but does not deliver information about the importance of features (of the base learners), for example.

To answer the question if a machine learning algorithm is interpretable, one have to specify the question. The model-agnostic methods, which can be applied to all algorithms, give some information about the data and some underlying relationships. One can give a possible explanation why one customer does not get a credit, but another does. Nevertheless, these methods do not provide a deep and presentable understanding of the algorithm's decision-making process, they do not make the model itself transparent. Whether the model-agnostic methods are sufficient to build trust in the predictions, one has to answer individually.

Chapter 8

Summary and Outlook

In the previous chapters, we have seen promising results of machine learning algorithms for credit risk modelling. The improvements in prediction performance are tremendous. All machine learning algorithms, except for CART, achieve higher AUC measures than the benchmark logistic regression model. The best single learner is a gradient boosting machine which results in an AUC of 0.7781, compared to 0.7233 of logistic regression. The performance can be enhanced further by combining the predicted probabilities of different algorithms via logistic regression as a super learner. This is called stacking and achieves an AUC of 0.7829.

The improvements mean a better prediction of default probabilities and default events. This can lead to less defaults in loan portfolios, thus, to more stable incomes for banks and lower interest rates for customers. Furthermore, better default prediction can reduce the number and extent of personal insolvency. The macroeconomic influences of a stronger and more liquid consumer credit sector are not to deny.

The paper at hand could not totally break up the black box which covers most machine learning algorithms. Some approaches and ideas are introduced but the challenge is not solved comprehensively. For extensive and area-wide adoption of machine learning algorithms, particularly in the financial services sector, the author thinks the black box prevents further expansion. Due to requirements of regulators and higher management the argument of better performance might fade away as long as no satisfying interpretability is established. Data protection laws and the customers' vital interest in transparent decision making processes makes it essential to put more effort in enhancing the interpretability of machine learning algorithms. In general, banks can be considered as careful when it comes to changes of systems, in particular to changes of sensitive and running systems like scorecards. Similarly, regulators and legislators need to be convinced that machine learning algorithms – despite it downsides in interpretability – can lead to a more stable financial system and brings improvements for the customers.

In the analyses at hand, we assess the customer's probability of default at one point in time. However, this might unrealistically oversimplify the situation. The probability of default of customers will vary over time. For example, reducing the line of credit could lower the probability of default. This can be modelled with time varying models or reinforcement learning. Although these are very interesting and promising areas, they would extend the scope of the work but are open for further research.

List of Figures

List of Tables

Bibliography

Anderson, R. (2007), *The credit scoring toolkit: theory and practice for retail credit risk management and decision automation*, Oxford University Press.

Basel Committee on Banking Supervision and Bank for International Settlements (2000), *Principles for the management of credit risk*, Bank for International Settlements.

Berkson, J. (1944), 'Application of the logistic function to bio-assay', *Journal of the American Statistical Association* **39**(227), 357–365.

Bischl, B. & Lang, M. (2015), *parallelMap: Unified Interface to Parallelization Back-Ends*. R package version 1.3.
URL: *https://CRAN.R-project.org/package=parallelMap*

Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G. & Jones, Z. M. (2016), 'mlr: Machine learning in r', *Journal of Machine Learning Research* **17**(170), 1–5.
URL: *http://jmlr.org/papers/v17/15-066.html*

Bliss, C. I. (1934), 'The method of probits', *Science* **79**(2037), 38–39.

Breiman, L. (1996), 'Bagging predictors', *Machine learning* **24**(2), 123–140.

Breiman, L. (1997), Arcing the edge, Technical report, Technical Report 486, Statistics Department, University of California at Berkeley.

Breiman, L. (2001a), 'Random forests', *Machine learning* **45**(1), 5–32.

Breiman, L. (2001b), 'Statistical modeling: The two cultures', *Statistical science* **16**(3), 199–231.

Breiman, L., Friedman, J., Stone, C. J. & Olshen, R. A. (1984), *Classification and regression trees*, CRC press.

Brier, G. W. (1950), 'Verification of forecasts expressed in terms of probability', *Monthly Weather Review* **78**(1), 1–3.

Chen, T. & Guestrin, C. (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.

Cook, N. R. (2007), ‘Use and misuse of the receiver operating characteristic curve in risk prediction’, *Circulation* **115**(7), 928–935.

Credit Suisse (1997), ‘Creditrisk+: A credit risk management framework’, *Credit Suisse Financial Products* pp. 18–53.

European Parliament (2016), ‘Regulation (eu) 2016/679 of the european parliament and of the coucil of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation)(2016)’, *Official Journal of the European Union L* **119**, 1–88.

Fahrmeir, L., Kneib, T., Lang, S. & Marx, B. (2013), *Regression: models, methods and applications*, Springer Science & Business Media.

Finney, D. J. & Tattersfield, F. (1952), *Probit analysis*, Cambridge University Press; Cambridge.

Freund, Y. & Schapire, R. E. (1997), ‘A decision-theoretic generalization of on-line learning and an application to boosting’, *Journal of computer and system sciences* **55**(1), 119–139.

Friedman, J. H. (2001a), ‘Greedy function approximation: a gradient boosting machine’, *Annals of statistics* pp. 1189–1232.

Friedman, J., Hastie, T. & Tibshirani, R. (2001b), *The elements of statistical learning*, Vol. 1, Springer series in statistics New York.

Gower, J. C. (1971), ‘A general coefficient of similarity and some of its properties’, *Biometrics* pp. 857–871.

Guyon, I. (1997), ‘A scaling law for the validation-set training-set size ratio’, *AT&T Bell Laboratories* pp. 1–11.

Hall, P., Gill, N., Kurka, M. & Phan, W. (2017), ‘Machine learning interpretability with h2o driverless ai’, *h20 documentation* .

Hand, D. J. & Anagnostopoulos, C. (2013), ‘When is the area under the receiver operating characteristic curve an appropriate measure of classifier performance?’, *Pattern Recognition Letters* **34**(5), 492–495.

Ho, T. K. (1995), Random decision forests, *in* ‘Document analysis and recognition, 1995., proceedings of the third international conference on’, Vol. 1, IEEE, pp. 278–282.

James, G., Witten, D., Hastie, T. & Tibshirani, R. (2013), *An introduction to statistical learning*, Vol. 112, Springer.

JP Morgan (1997), ‘Creditmetrics - technical document’, *JP Morgan, New York* .

Khandani, A. E., Kim, A. J. & Lo, A. W. (2010), 'Consumer credit-risk models via machine-learning algorithms', *Journal of Banking & Finance* **34**(11), 2767–2787.

Kolmogorov, A. (1933), 'Sulla determinazione empirica di una lgge di distribuzione', *Inst. Ital. Attuari, Giorn.* **4**, 83–91.

Kruppa, J., Schwarz, A., Arminger, G. & Ziegler, A. (2013), 'Consumer credit risk: Individual probability estimates using machine learning', *Expert Systems with Applications* **40**(13), 5125–5131.

LeDell, E., Gill, N., Aiello, S., Fu, A., Candel, A., Click, C., Kraljevic, T., Nykodym, T., Aboyoun, P., Kurka, M. & Malohlava, M. (2018), *h2o: R Interface for 'H2O'*. R package version 3.20.0.2.
URL: *https://CRAN.R-project.org/package=h2o*

Liaw, A. & Wiener, M. (2002), 'Classification and regression by randomforest', *R News* **2**(3), 18–22.
URL: *http://CRAN.R-project.org/doc/Rnews/*

Lipton, Z. C. (2016), 'The mythos of model interpretability', *arXiv preprint arXiv:1606.03490* .

Lobo, J. M., Jiménez-Valverde, A. & Real, R. (2008), 'Auc: a misleading measure of the performance of predictive distribution models', *Global ecology and Biogeography* **17**(2), 145–151.

McCullagh, P. (1984), 'Generalized linear models', *European Journal of Operational Research* **16**(3), 285–292.

McCulloch, W. S. & Pitts, W. (1943), 'A logical calculus of the ideas immanent in nervous activity', *The bulletin of mathematical biophysics* **5**(4), 115–133.

Molnar, C., Bischl, B. & Casalicchio, G. (2018), 'iml: An r package for interpretable machine learning', *JOSS* **3**(26), 786.
URL: *http://joss.theoj.org/papers/10.21105/joss.00786*

Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B. & Swami, A. (2017), Practical black-box attacks against machine learning, *in* 'Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security', ACM, pp. 506–519.

R Core Team (2018), *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria.
URL: *https://www.R-project.org/*

Ribeiro, M. T., Singh, S. & Guestrin, C. (2016), Why should i trust you?: Explaining the predictions of any classifier, *in* 'Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining', ACM, pp. 1135–1144.

Schliep, K. & Hechenbichler, K. (2016), *kknn: Weighted k-Nearest Neighbors*. R package version 1.3.1.
URL: *https://CRAN.R-project.org/package=kknn*

Schmidhuber, J. (2015), 'Deep learning in neural networks: An overview', *Neural networks* **61**, 85–117.

Sharpe, W. F. (1963), 'A simplified model for portfolio analysis', *Management science* **9**(2), 277–293.

Simon, H. A. (1983), Why should machines learn?, *in* 'Machine learning', Springer, pp. 25–37.

Smirnov, N. V. (1939), 'On the estimation of the discrepancy between empirical curves of distribution for two independent samples', *Bull. Math. Univ. Moscou* **2**(2), 3–14.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014), 'Dropout: a simple way to prevent neural networks from overfitting', *The Journal of Machine Learning Research* **15**(1), 1929–1958.

The Financial Crisis Inquiry Commission (2011), *The financial crisis: inquiry report*, Superintendent of Documents, U.S. Government Printing Office, Washington.

Therneau, T. & Atkinson, B. (2018), *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-13.
URL: *https://CRAN.R-project.org/package=rpart*

Tsai, C.-F. & Chen, M.-L. (2010), 'Credit rating by hybrid machine learning techniques', *Applied soft computing* **10**(2), 374–380.

Tzeng, F.-Y. & Ma, K.-L. (2005), Opening the black box-data driven visualization of neural networks, *in* 'Visualization, 2005. VIS 05. IEEE', IEEE, pp. 383–390.

Van der Laan, M. J., Polley, E. C. & Hubbard, A. E. (2007), 'Super learner', *Statistical applications in genetics and molecular biology* **6**(1).

Wang, G., Hao, J., Ma, J. & Jiang, H. (2011), 'A comparative assessment of ensemble learning for credit scoring', *Expert systems with applications* **38**(1), 223–230.

Yeh, I.-C. & Lien, C.-h. (2009), 'The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients', *Expert Systems with Applications* **36**(2), 2473–2480.

Appendix A

Feature Importance and Partial Dependence Plots

GLM

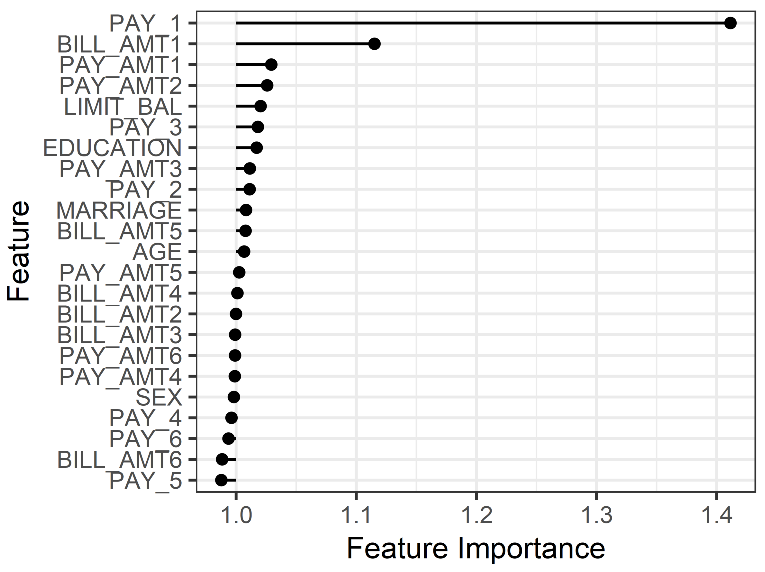


Figure A.1: Feature importance GLM 1.

kNN

CART

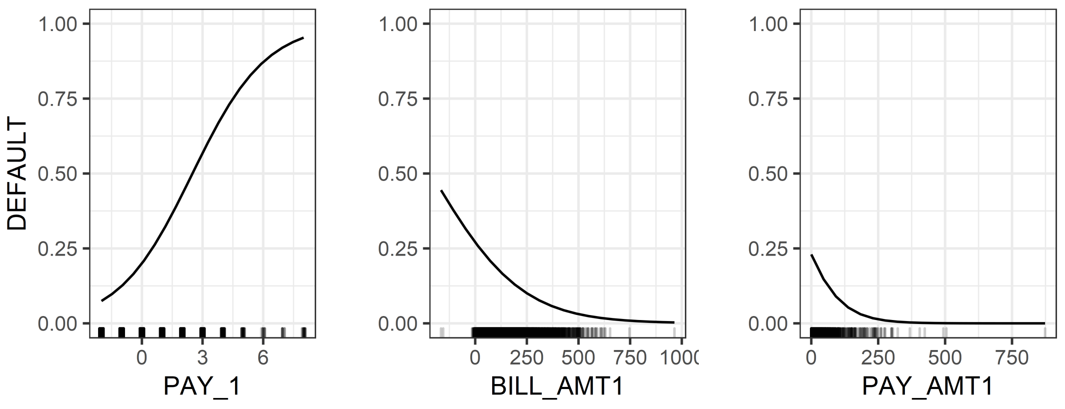


Figure A.2: Partial dependence plots of GLM 1.

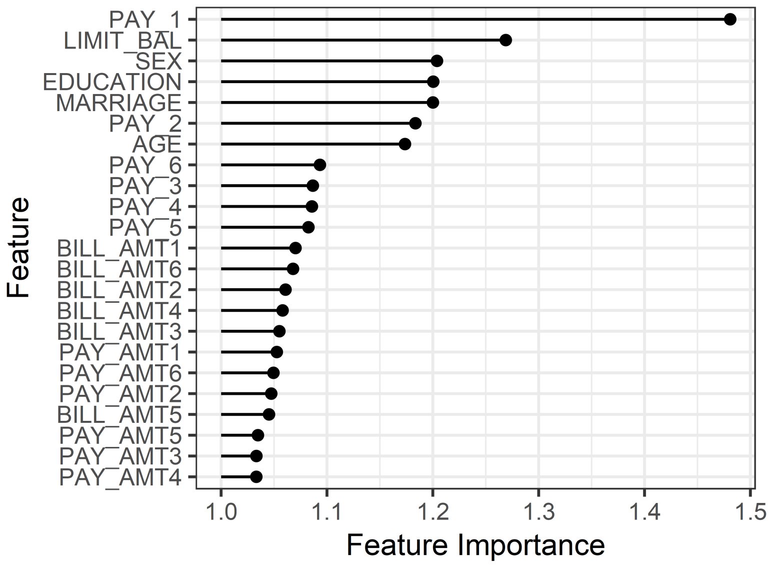


Figure A.3: Feature importance kNN 4

Feature	**Weight**	**Std. Error**	z **Value**	**Pr**$(\| > z\|)$	**Significance**
Bias term	-2.150e+00	5.227e-01	-4.114	3.89e-05	***
LIMIT_BAL	-7.582e-07	1.569e-07	-4.831	1.36e-06	***
SEX2	-1.111e-01	3.072e-02	-3.618	0.000297	***
EDUCATION	-1.018e-01	2.102e-02	-4.845	1.26e-06	***
MARRIAGE1	1.262e+00	5.154e-01	2.449	0.014332	*
MARRIAGE2	1.070e+00	5.157e-01	2.074	0.038066	*
MARRIAGE3	1.182e+00	5.324e-01	2.221	0.026361	*
AGE	6.284e-03	1.835e-03	3.424	0.000616	***
PAY_1	5.770e-01	1.770e-02	32.605	$<$ 2e-16	***
PAY_2	8.330e-02	2.019e-02	4.126	3.68e-05	***
PAY_3	7.254e-02	2.261e-02	3.208	0.001336	**
PAY_4	2.350e-02	2.501e-02	0.940	0.347410	
PAY_5	3.418e-02	2.688e-02	1.271	0.203615	
PAY_6	7.774e-03	2.214e-02	0.351	0.725464	
BILL_AMT1	-5.520e-06	1.137e-06	-4.857	1.19e-06	***
BILL_AMT2	2.389e-06	1.505e-06	1.587	0.112545	
BILL_AMT3	1.330e-06	1.323e-06	1.005	0.314963	
BILL_AMT4	-1.728e-07	1.349e-06	-0.128	0.898079	
BILL_AMT5	6.320e-07	1.519e-06	0.416	0.677408	
BILL_AMT6	3.951e-07	1.195e-06	0.331	0.741011	
PAY_AMT1	-1.360e-05	2.305e-06	-5.899	3.65e-09	***
PAY_AMT2	-9.600e-06	2.095e-06	-4.582	4.60e-06	***
PAY_AMT3	-2.750e-06	1.722e-06	-1.597	0.110264	
PAY_AMT4	-4.038e-06	1.785e-06	-2.262	0.023677	*
PAY_AMT5	-3.321e-06	1.776e-06	-1.870	0.061452	.
PAY_AMT6	-2.060e-06	1.295e-06	-1.590	0.111808	

Table A.1: All weights of logistic regression model.

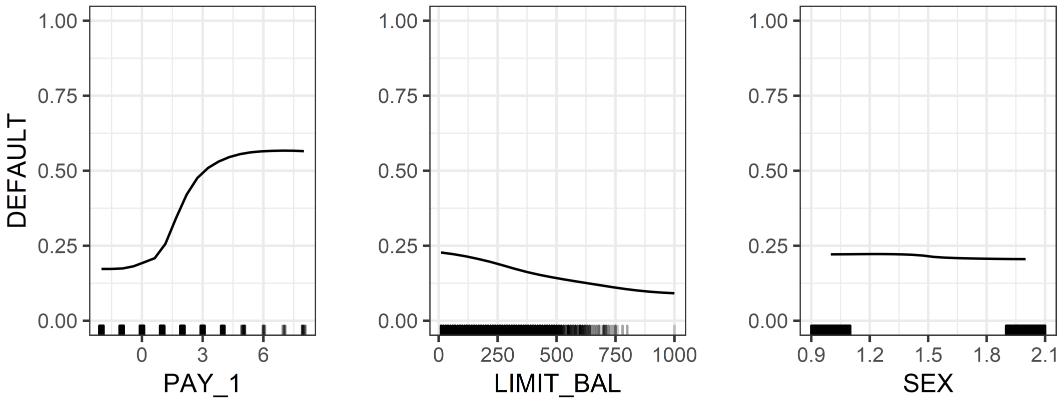
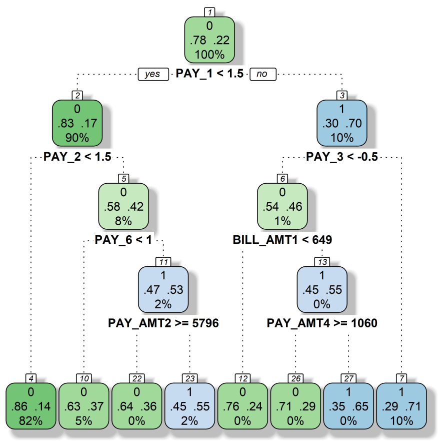
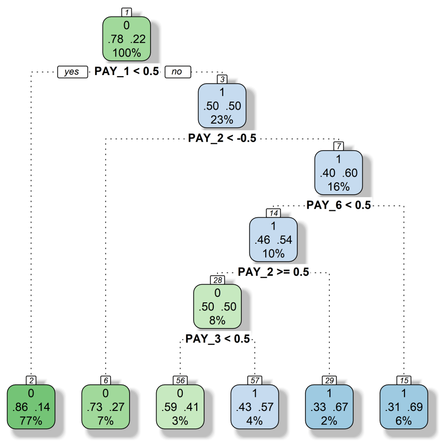


Figure A.4: Partial dependence plots of kNN 4.



(a) plotCart3



(b) plotCart4

Figure A.5: CART plots

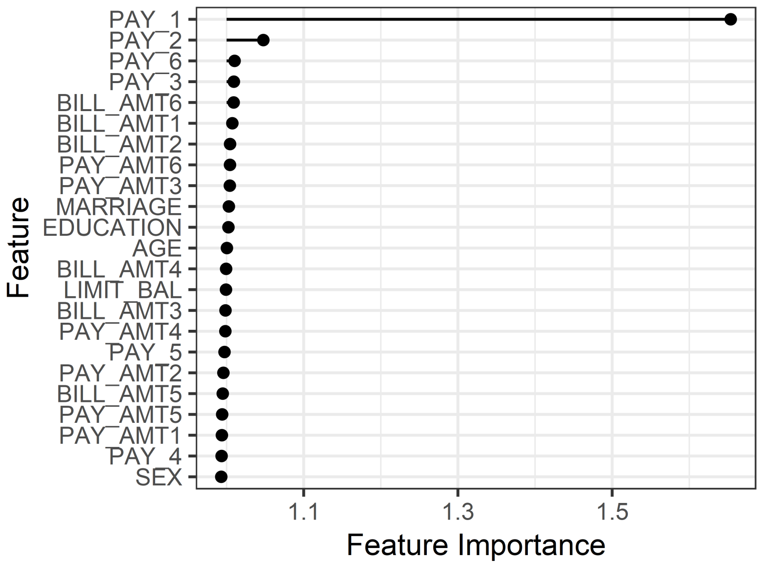


Figure A.6: Feature importance CART 4

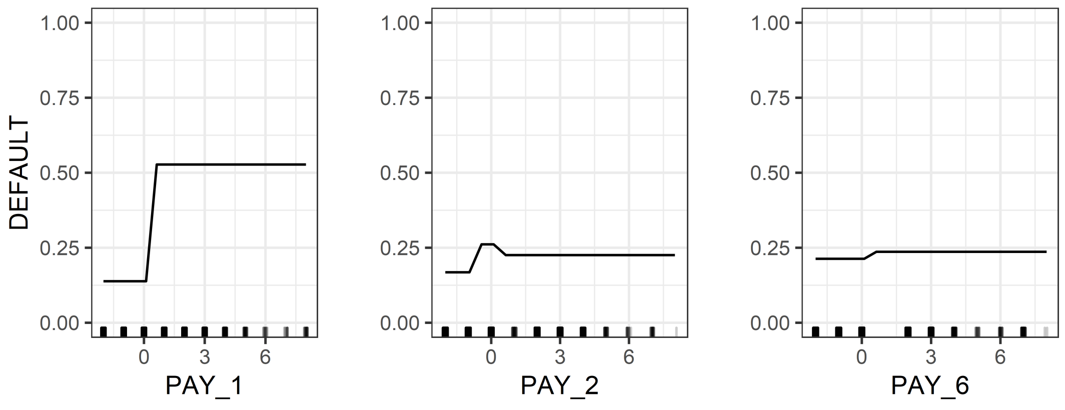


Figure A.7: Partial dependence plots of CART 4.