Predicting Credit Card Default Using Deep Learning

Jonas Gann, Georg Grab

Rupert-Karls-University Heidelberg, Heidelberg, Germany
{jonas.gann, georg.grab}@stud.uni-heidelberg.de

Abstract. Predicting credit card default is an important task for payment card providers to manage risk and take appropriate preventative action. It is often claimed that the tabular structure of performance indicators related to these events remains a unique challenge to deep learning based methods, despite their numerous breakthroughs in other domains. In this work, we aim to verify this claim by training and systematically analyzing several deep learning architectures for the task of credit default detection and subsequently comparing the results to a carefully tuned decision tree based classifier. The best performing deep learning architecture achieves a final evaluation score of 0.783, whereas the decision tree based model achieves 0.791. This result confirms prior work in the area of credit default prediction, stating that decision tree based models reach state-of-the-art performance that deep learning based method fail to surpass.
The code is publicly available at `https://github.com/J-Gann/AML`

Keywords: Credit Default Prediction · Gaussian Processes · Deep Learning · Time Series.

1 Introduction

Grab

Credit cards have become an indispensable asset of everyday life, allowing quick and easy payments and borrowing. Owing to this simplicity, it is easy to forget that every time we swipe or touch the card, we enter into a legally binding agreement with the card provider to eventually pay back the money we owe. A default event in this context is defined as the failure of a debtor to do so over a predefined time period, for example, the passing of 180 days after issuance of the latest credit card statement. If such an event occurs, payment card providers typically assume that the debt will never be settled, close the account, and sell the account to a debt collection agency. As only a fraction of debt can typically be recovered by debt collection, the payment card provider incurs substantial losses if such events occur frequently [5]. Assessing default risk is thus a vital part of business for lending institutions. Although there are centralized efforts to measure delinquency risk, such as credit scoring exist, payment card providers typically also develop in-house risk models. The Credit Default Prediction competition was organized by the payment card provider American Express in 2022

on the data science competition website Kaggle. Its purpose was to assess the performance of machine learning models when phrasing credit default prediction as a classification problem based on past performance indicators.

In this work, we investigate whether deep learning-based models can be successfully applied to this problem, where we define success as surpassing ensemble models in evaluation performance, as such models are typically the top performers in competitions based on similar tabular datasets [19]. To do this, we participate in the Credit Default Prediction competition, developing several deep learning based models, and comparing their performance to a carefully tuned ensemble method based on gradient boosted decision trees.

Section 2 briefly introduces related work and the machine learning techniques utilized. Section 3 elaborates in detail the challenge, the dataset on which it is based and the associated evaluation metrics, as well as our applied methodology, including preprocessing, training, and hyperparameter tuning. In Section 4 we present experimental results, which are discussed in Section 5 along with associated limitations and possibilities for future work.

2 Related Work

Grab

This section gives a brief overview of previous work related to credit default prediction and associated machine learning methods.

Due to the importance of assessing the default risk for various stakeholders, predictive models have been in demand for some time. Among the first risk models to find a major application is the Merton model proposed in 1974 [15], based on economic theory around company valuations and associated economic indicators. At the consumer level, the concept of credit scoring emerged as the primary measure of assessing default risk, where individual consumer actions are associated with a risk factor that accumulates in a global score per customer [8]. Simple regression models remain a popular choice to calculate this metric in practice, as the deployment of black box approaches is often prevented from a regulatory perspective [12]. Among machine learning approaches for computing such risk scores, Support Vector Machines are being applied successfully [20] [16], although Gradient Boosted Decision Trees (GBDT) and Nearest Neighbor approaches also show favorable results [4]. In 2020, Alman et al. published a comprehensive investigation of machine learning techniques for the default prediction problem, comparing a wide variety of methods across several independent datasets. GBDT emerged as the best performing models [4], although deep learning based approaches were not included in this discussion. Such a comparative evaluation of GBDT and random forest based methods against deep learning approaches is provided in a 2018 report by Addo et al. [2], the most similar prior work to ours we identified in the literature survey. In their analysis, logistic regression, GBDT, and random forest based models are compared with four different neural networks of varying complexity, the most complex of which has been selected through grid-based hyperparameter optimization. The results as measured by AUROC and the mean squared error on a dedicated testing dataset

are in favor of GBDT and random forest models, with all neural network configurations failing to achieve higher test performance. However, Addo et al. only use simple neural network architectures consisting of fully connected layers, few hidden neurons, and a maximum depth of four, raising the question of whether more contemporary architectures would perform better.

The datasets used for credit default prediction are usually in the form of a tabular time series of past performance indicators. The application of machine learning methods to time series data has been studied extensively. Possible deep learning approaches include feed-forward and convolutional networks, as well as more specialized approaches such as long short-term memory and recurrent neural networks [10].

A problem inherently associated with credit default prediction, which is often cited in the literature, is addressing dataset imbalance. As credit default is a rare event, the majority class of supervised training datasets are related to customers that do not default. Dataset imbalance in this context is usually addressed by undersampling the majority class or oversampling the minority class during training [11]. In the context of neural networks, weighted loss functions that result in steeper gradients for the minority class, such as focal loss, have been proposed [13].

3 Materials and Methods

In this section, we outline the materials and methods used to investigate the problem. First, we give a detailed overview of the challenge, the dataset on which it is based, and the evaluation metrics defined by the challenge organizers. Then, we describe the process of training and optimizing an ensemble model used for hyperparameter optimization as well as to provide a baseline performance measurement that can be compared to the deep learning based approaches described thereafter. Finally, we describe methods for evaluating and comparing our own models.

3.1 The Dataset

Grab

The tabular dataset consists of a total of 5.5 million records, each consisting of 190 features. While a time point and the related customer ID are available for each record, the meaning of all other variables is partially obfuscated, with only the broad category in which the feature falls being revealed. The broad categories are delinquency, spend, payment, balance, and risk. Per customer, a maximum of 13 such records are available. According to the challenge description, these records represent an 18 months performance window after the latest credit card statement. Most variables are continuous, with only 11 of 190 features being categorical. Categorical features are represented by integers or strings. Missing values are frequent throughout the dataset, both with regards to single values and entire records. On average, 12.1 records are available per customer, and no single record in the training data exists with all values non-empty.

To gain an initial overview of the available features despite anonymization, a preliminary exploratory analysis of the importance of the features was performed. The importance of given feature categories can be quantified by training a decision tree-based ensemble model with minimal preprocessing and counting the frequencies of features that are used as split criteria. For this exploratory analysis, individual records were joined with the target variable and temporal information was removed. The model was trained using an ensemble of decision trees with gradient boosting. A validation split of 15% was used to detect overfitting and perform early stopping. After training, feature importances were calculated using independent tree SHaP values [14]. Fig. 1 shows the mean of feature importances calculated per feature category, with feature *P_2* plotted separately. The plot shows that the feature *P_2* is an outlier in terms of predictive performance in the resulting ensemble model, indicating that this feature already represents a type of default risk. For the remaining features, delinquency is on average the category with the highest explanatory power, followed by, in descending order, balance, payment, spend, and risk-related variables.

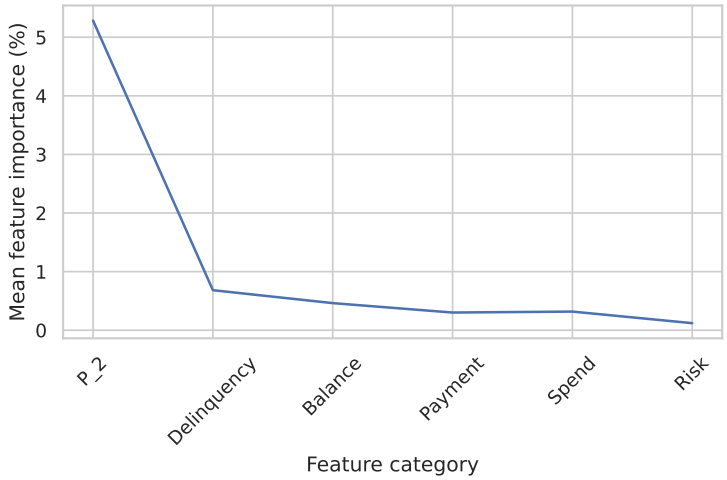


Fig. 1: Mean Feature Importance per Category using independent SHaP values [14], with outlier *P_2* shown separately.

3.2 Design of the Challenge

Grab

The challenge was held as a public event on the data science competition website Kaggle over a three month time window from May 2022 until August 2022. During that time, participants were free to register for the challenge, form teams, and submit candidate solutions derived from the competition dataset. Candidate solutions were submitted in the form of predictions of default probabilities

on the challenge's publicly available testing dataset. Upon submission of such a candidate solution, the challenge evaluation metric would be computed for the entire test dataset, but only reported on a subset thereof making up approximately 51% of all available data. This subset was kept secret by the challenge organizers. The evaluation metric achieved, called the public score, would be immediately published on a leaderboard that was public for the duration of the challenge. Only after passing of the challenge deadline, the evaluation metric achieved on the remaining 49% would be revealed, referred to as the private score. This private score was then used for the final ranking of the participants.

3.3 Official Evaluation Metrics

Grab

The mean of two separate ranking measures was used as the official evaluation metric of the challenge,

$$M = \frac{G + D}{2},$$

with G the Normalized Gini Coefficient, and D the default rate captured at 4%. The Normalized Gini Coefficient is a metric capable of describing the overall performance of a classifier across all possible operating points in a way equivalent to the more familiar Area under the Receiver Operating Characteristic curve (AUROC) [18]. For the default rate at 4%, the predicted default probabilities are sorted in descending order. The metric is defined by calculating the true positive rate for the first 4% of the predictions in this ordered list.

3.4 Baseline Hyperparameter Tuning

Grab

Data preprocessing poses a unique challenge for the given dataset for various reasons. First, the features are completely anonymized apart from a broad feature category, making preprocessing through manual feature engineering difficult. Second, missing values are frequent throughout the dataset, and lastly, the data is structured as a time series, with a variable amount of records available per customer. Therefore, multiple data preprocessing strategies have to be applied, each with associated hyperparameters.

To systematically determine the appropriate preprocessing hyperparameters for the given dataset, a hyperparameter optimization strategy was devised. First, suitable hyperparameters are identified, and associated preprocessing routines are implemented using these parameters. Then, using the data preprocessed according to a given set of hyperparameters, a baseline classifier is trained. Finally, this classifier is evaluated on a validation split according to the evaluation metrics defined in Section 3.3. This entire process is modeled as a function $f(x, \theta)$ with x the raw input data and θ the set of hyperparameters, returning the final challenge evaluation metric.

Using x as a constant subset of the full input data for efficiency reasons and allowing θ to vary, an optimization problem can be formulated:

$$\hat{\theta} = \arg\max_{\theta} f(x, \theta). \tag{1}$$

While computing $\hat{\theta}$ exactly is intractable due to the large input space of potential hyperparameters and the fact that many local optima exist, good approximations can be obtained through Gaussian Process based hyperparameter optimization. To perform the optimization, the Optuna framework [3] was used.

Hyperparameters that are part of the optimization are listed in Table 1. Hyperparameters are part of the whole machine learning pipeline and, as such, both parameters related to dataset preprocessing, as well as parameters directly related to the predictor, are subject to optimization.

Hyperparameter	**Description**	**Type**	**Range**
Impute Strategy $^{(*)}$	Statistic used to impute missing values	Discrete	mean, median, most frequent
Grouped Impute $^{(*)}$	Impute based on global or per-customer statistics	Boolean	T, F
Drop Percentage $^{(*)}$	Delete columns with high correlation coefficient	Continuous	$[0.8; 0.99]$
Denoise$^{(*)}$	Apply denoising strategies	Boolean	T, F
Depth	Tree depth	Integer	$\{3; 11\}$
Iterations	Maximum number of trees to add to GBDT ensemble	Integer	$\{20; 150\}$
Learning Rate	Learning rate for GBDT ensemble	Continuous	$[10^{-5}; 10^{-3}]$

Table 1: Hyperparameters optimized for baseline regressor. Dataset related parameters $(*)$ were reused for deep learning based experiments.

The first two dataset preprocessing parameters are related to imputation strategies. Proper handling of such values plays an important role in the challenge dataset, as missing values are so frequent that simple strategies such as deleting incomplete customer records would result in deletion of the entire training data.

While more elaborate imputation techniques using machine learning based regressors exist, this analysis is restricted to simple strategies that fill missing values given a column statistic, as otherwise hyperparameter optimization proves intractable given the large volume of data. The optimization is subject to both the kind of statistic used for imputation - mean, median, or most frequent - as well as the conceptual level within which the imputation is performed. The latter is modeled as the boolean hyperparameter *Grouped Impute* which, if true, performs imputation on the level of the customer time series and imputes based on global column statistics otherwise. Regardless of the *Grouped Impute* setting, imputation based on the global column statistic is always carried out again at the end of preprocessing. The reason is that many instances of customer time

series exist that do not have any value in a given column, and as such computing the column statistic fails if done on the time series level.

A further challenge associated with the dataset is that many features show a high degree of statistical correlation. Highly correlated features, in general, do not improve predictive performance and are known to decrease performance in some cases [22]. Using correlation coefficients, such features can be detected; however, it is not clear in general which threshold to use beyond which on selecting the threshold at which one of the highly correlated features should be deleted. Because of this, a *Drop Percentage* hyperparameter was added to the optimization, with possible values in the interval $[0.8; 0.99]$ tested.

Finally, analyzing the distributions of individual features reveals that uniform random noise in the range $[0, 0.01]$ might have been injected into the dataset. This claim stems from the fact that features with the majority of values distributed in the range $[0, 1]$ have minimum and maximum values that are slightly negative and slightly above one, respectively, with a maximum deviation of 0.01. As most of the features are likely to describe money, which is a discrete quantity with increments of \$0.01, it is unlikely such values are naturally occurring in the dataset. A possible denoising strategy is to round the values to the nearest two significant digits. This is enabled in the preprocessing pipeline if the *Denoise* parameter is enabled.

After all preprocessing steps outlined above are executed, the data is reshaped such that all records pertaining to one customer represent one row of the training data, with time series data added as additional columns.

The remaining hyperparameters in Table 1 are related to the baseline classifier, a Gradient Boosted Decision Tree ensemble that is trained using a set of dataset hyperparameters, and evaluated to quantify their effect.

3.5 Gradient Boosted Decision Trees

Grab

This work uses Gradient Boosted Decision Trees (GBDT) in two ways. First, as described in the previous Section, a round of dataset hyperparameter optimization is concluded with training GBDT on the preprocessed data in order to obtain a value of $f(x, \theta)$. Second, the results of GBDT are used as a baseline for comparison against deep learning approaches, as GBDT is a strong method for tabular data in general and credit default prediction in particular [2]. The main idea of GBDT is to first take a naive guess, typically the mean, to predict the output variable, to calculate the residual between the guess and the actual value, and then incrementally refine this guess by sequentially chaining decision trees [17]. Every decision tree is designed to contribute a certain amount, quantified by a learning rate parameter, to refining the error of the previous decision tree in the chain, quantified by a differentiable loss function. Trees are added either until a predefined maximum tree count is reached, overfitting is detected as measured on a validation split, or the gradient of the loss function drops below a certain threshold.

In this work, we use the CatBoost implementation of GBDT, which additionally extends the gradient boosting algorithm in certain ways to make it work well

on categorical features and noisy input data [17]. Data is preprocessed according to the strategy laid out in Section 3.4. For the final predictions, GBDT related optimized hyperparameters as summarized in Table 1 are reused. Weighted binary cross entropy is used as the loss function in order to counteract dataset imbalance. The rest of the hyperparameters of the CatBoost classifier remain at their default values. The final predictions are computed by an ensemble obtained through 5-fold cross validation, with individual predictions averaged.

3.6 Multi Layer Perceptron

Gann

As an initial approach to solving the classification problem with deep learning models, we utilize a Multi Layer Perceptron (MLP). A MLP is a feedforward artificial neural network consisting of multiple fully connected layers of perceptrons with a nonlinearity in between. An example of a single layer is shown in Figure 2. Here, one can see that an activation (which is the resulting numerical vector from the previous layer) is multiplied with a weight matrix in what is called a linear layer. The resulting output (which dimension is determined by the size of the activation and the weight matrix) is referred to as pre-activation to which an activation function is applied to generate the input used for the next layer.

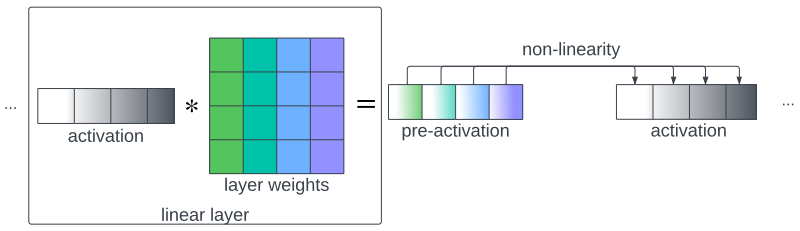


Fig. 2: MLP layer with associated terminologies.

The first layer is commonly used to represent the input data in the form of a one-dimensional vector. The activation resulting from the last layer can be used as an output of the model. In the case of binary classification, the activation of the last layer usually is a Sigmoid function to map the pre-activation to a probability between 0 and 1. As can be seen in Figure 2 the input to a linear layer must be a one-dimensional numerical vector. Therefore, we can use the preprocessed dataset described in 3.4 which has, numerical and categorical variables combined, the shape $[367130, 2314]$ where 367130 is the number of entries in the dataset and 2314 is the feature size. Thus, each feature is the ordered concatenation of the 13 time series values of size 178 per customer.

The concrete architecture of the MLP was optimized using Optuna as will be shown in Section 3.9.

3.7 Convolutional Neural Network

Gann

The MLP described in the previous section is not a time-invariant approach to time series classification [7]. This means that small local changes in the order of features can have a large effect on the overall classification. The reason is that each feature in an MLP has its own weight and cannot be shifted to the left or right. This behavior might be advantageous when each feature is independent of its neighboring features. However, for time series, this is typically not the case, as one can imagine patterns emerging at different time points and, therefore, different feature indices. It might be of advantage to utilize a time-invariant approach, which could improve classification accuracy by enabling the model to better maintain a notion of order while learning features where local changes in the input can be limited to only have an effect on neighboring features during classification. This means that some characteristic time series could be detected at time point t just as at time point $t+5$, thus making its detection invariant to its point in time. One example of a time-invariant approach is the use of Convolutional Neural Networks (CNN), which in this case can use one-dimensional convolutions to learn filters for expressive feature generation.

We use the preprocessed dataset described in 3.4 for CNN training. However, the data needs to be modified to be usable as input data to the CNN. Therefore, we reshape the feature vector to $[178, 13]$, where 178 is the number of channels and 13 is the length of the time series. As shown in Figure 3 during the convolution, a one-dimensional kernel (marked red) of variable size (size 3 in this case) slides over the time series and applies its filter to the input values at each step. Using the "same" padding, the feature vector produced by the convolution can be expanded to retain its original size by replicating the first and last elements.

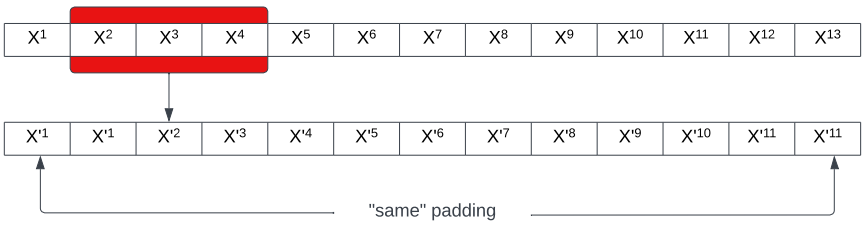


Fig. 3: One dimensional convolution over a time series of length 13 stride one, kernel size 3, and "same" padding.

This convolution process is performed for each of the input channels of a customer using individual kernels. The result of the convolution of each channel is summed up, resulting in a vector of size $[178, 1]$ when the "same" padding is used. To create multiple output channels, multiple convolutions can be performed

and their results concatenated. In this way, multiple convolutional layers can be stacked, each with the same input and output shapes. The purpose of the CNN is to generate expressive features by learning various filters for each time series. A filter applied to a time series representing the credit balance could, for example, be learned to detect a sudden large reduction within an otherwise even value. This could be one of many indicators of default risk.

Each of the learned filters is time invariant, as they detect patterns independent of where they appear in the time series by being applied in a sliding operation as shown in Figure 3. Therefore, a CNN approach for feature extraction of time series data takes advantage of the inherent structure of the time series, which can increase classification performance.

The features learned by the CNN are forwarded to an MLP that reduces the dimensionality to 1 for the final probability prediction of a credit default using a Sigmoid activation function.

The concrete architecture of the CNN is optimized using Optuna as discussed in Section 3.9.

3.8 MLP and CNN Training

Gann

Training was identical for both MLP and CNN despite differences in training hyperparameters described in Section 3.9.

The data set was used as described in Section 3.4 as well as the adoptions described in Section 3.6 and Section 3.7. We divided the data set into a training set and a validation set while 70% of the data points remained in the training set.

We used binary cross-entropy as a loss function and trained for 9 epochs where each epoch consisted of the entire training set. In the training loop we logged the average loss of the batch, and after every 100 batches we evaluated the model against the validation set and logged the resulting average loss. After each epoch, we stored the current model to select the best performing model afterwards and multiplied the learning rate with an optimized learning rate decay factor.

3.9 MLP and CNN Hyperparameter Tuning

Gann

We again use Optuna to optimize the hyperparameters of the MLP and CNN. The reason we again use automated hyperparameter optimization is that manually specifying hyperparameters such as number of layers, kernel size, output size, learning rate, and optimizer can take a very long time, and it is likely that without much experience, one does not find optimal values. However, the selection of hyperparameters to be tuned as well as the range in which they are optimized is still manual, giving the possibility of speeding up computation but leaving possibilities for missed optima.

For optimization, we used the adapted training data for the respective models as described in Section 3.6 and Section 3.7. However, in order to get reasonable

performance, we limited the size of the dataset to 10000 entries. To evaluate the performance of trials, we used the binary cross-entropy loss.

The hyperparameters selected for optimization are shown in Tables 2 and 3.

Hyperparameter	**Type**	**Range**
Number Linear Layers	Integer	1 - 10
Size Linear layer k	Integer	100 - 300
Activation	Discrete	LeakyReLU, ReLU, ELU
Learning Rate	Continuous	1e-7 - 0.1
Optimizer	Discrete	AdamW, Adam, SGD, Adagrad, NAdam
Learning Rate Decay	Continuous	0.4 - 0.99
Batch Size	Integer	16 - 128

Table 2: Hyperparameters optimized for Multi Layer Perceptron

The hyperparameters that were optimized concern either the model architecture or the training process. In the case of the MLP, the hyperparameters concerning the model architecture were the number of layers and the size of each layer. The trade-off here is between sufficient model complexity and overfitting as well as performance. Although ReLU activations are commonly used, we included some alternative activation functions in the optimization which might increase performance in some cases.

In case of the CNN, the hyperparameters concerning the model architecture were similar to the MLP, the number of layers, and output size, in this case for the linear layers as well as for the convolutional layers. In addition, we optimized the kernel size for each convolutional layer. As can be seen in Figure 3, the size of the kernel has an effect on the size of the patterns that can be detected. It is not trivial to determine the suitable kernel sizes beforehand. In case they are too small, they might not catch up on larger patterns; however, if they are too large, they might incorporate irrelevant information besides the target pattern. As for the MLP, we also evaluated different activation functions besides ReLU. To reduce possible overfitting, we also evaluated whether a max-pooling layer after the convolutions would improve performance.

For both MLP and CNN we additionally evaluated hyperparameters concerning the training process. This is, most importantly, the optimizer in combination with the learning rate, and batch size.

As we only used a subset of the available data for optimization, we also added the epoch number as a hyperparameter to be optimized in order to enable Optuna to sufficiently increase the number of epochs until the training progress saturates. This hyperparameter, however, was not reused for the final training process.

Hyperparameter	**Type**	**Range**
Number Convolutional Layers	Integer	1 - 30
Kernel Size Convolutional Layer k	Integer	1 - 12
Out Channel Size Convolutional Layer k	Integer	100 - 300
Convolution Activations	Discrete	LeakyReLU, ReLU, ELU
Number Linear Layers	Integer	1 - 10
Layer Size Linear Layer k	Integer	100 - 300
Linear Activations	Discrete	LeakyReLU, ReLU, ELU
Pooling after Convolution	Discrete	True, False
Learning Rate	Continuous	1e-7 - 0.1
Learning Rate Decay	Continuous	0.4 - 0.99
Batch Size	Integer	16 - 128

Table 3: Hyperparameters optimized for Convolutional Neural Network

As for the definition of values hyperparameters can take, we started with an educated guess, a wide range in case of numerical values, and a wide selection in case of categorical values. Through iterative evaluation of the optimization results we reduced the value ranges in case it did not significantly decrease performance and removed very bad performing options of categorical hyperparameters to reduce the importance of corresponding categories as computed by Optuna to a reasonable level.

3.10 Model Evaluation

Grab

Having introduced the different candidate models, we now describe the evaluation procedure used to determine the best performing methods.

As mentioned in Section 3.2, while a global testing split is available to publish predictions on the challenge leaderboard, the lack of associated labels prevents further performance analysis using this dataset. Thus, we reserve 20% of the training dataset for the purpose of final performance analysis. For evaluation, all models receive the same input data determined through the optimized dataset described in Section 3.4. Each model is then trained and optimized according to the relevant procedures laid out in previous sections. After final training, predictions on the reserved 20% of the training data are made for all models. Finally, predictions on the global testing split are computed to allow for comparison with other challenge participants in terms of the challenge evaluation metric.

Using the predictions obtained on the reserved 20% of the training data, we perform two types of analysis. First, we compute and compare Receiver Operating Characteristic (ROC) curves, the associated AUC metric, and the metric

defined by the challenge organizers. Second, we compute the binary cross-entropy for every prediction and then perform a Wilcoxon signed rank test at 5% significance in order to obtain a ranking representing classification accuracy [6]. To ensure ranking stability, we use bootstrapping with $b = 1000$ samples. For the latter analysis, we use the R package *challengeR* [21].

4 Experiments and Results

In this section, we present our experimental results regarding hyperparameter optimization, training, validation, and final evaluation performance.

4.1 Baseline Hyperparameter Tuning

Grab

The results of the baseline hyperparameter optimization as laid out in Section 3.4 are displayed in Figure 4. A total of 330 evaluations of the function $f(x, \theta)$ have been conducted, with the associated objective values in form of achieved validation performance indicated by the blue points.

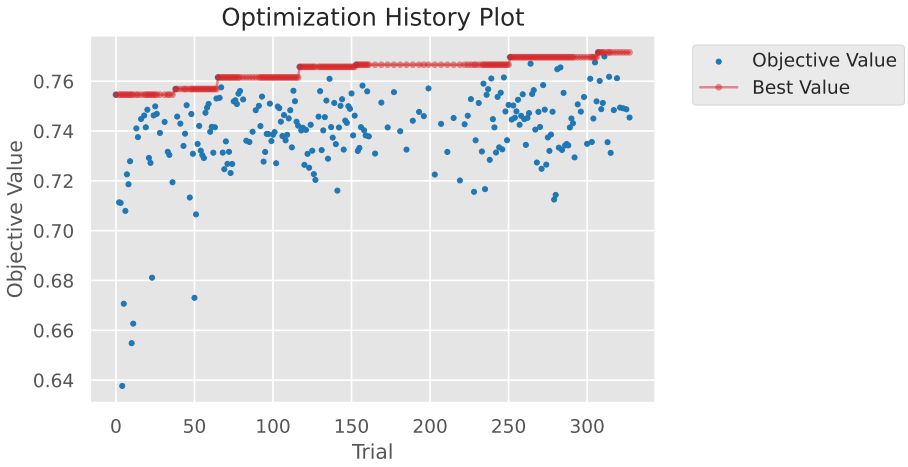


Fig. 4: Optimization history of the baseline model over a large number of trials, with individual points showing evaluations of $f(x, \theta)$, and points connected by the red line representing the best parameters $\hat{\theta}$ obtained up to that point.

Early trials result in low objective values because of the essentially random choice of hyperparameters. As the hyperparameter space is explored, the objective value improves logarithmically, though a large variance remains, with little practical performance gains beyond approximately 200 trials. Table 4 documents the values of dataset related hyperparameters as outlined in Section 3.4.

Hyperparameter	Description	Range	Best
Impute Strategy$^{(*)}$	Statistic used to impute missing values	mean, median, most frequent	**mean**
Grouped Impute$^{(*)}$	Impute based on global or per-customer statistics	T, F	$\boldsymbol{T}$
Drop percentage$^{(*)}$	Delete columns with high correlation coefficient	$[0.8; 0.99]$	0.94
Denoise$^{(*)}$	Apply denoising strategies	T, F	$\boldsymbol{T}$
Depth	Tree depth	$\{3; 11\}$	11
Iterations	Number of iterations for regressor	$\{20; 150\}$	130
Learning Rate	Learning rate for regressor	$[10^{-3}; 10^{-1}]$	$5 * 10^{-2}$

Table 4: Best hyperparameters $\hat{\theta}$ after 300 optimization trials achieving a challenge metric of $f(x, \hat{\theta}) = 0.76$. $(*)$ marks dataset parameters reused for deep learning experiments.

As a result of the optimization of the *Drop percentage* parameter, ten features were dropped that were highly correlated with other features in the dataset. The fact that the parameter reached the optimal value at 0.94 and not at an extreme end of the optimization range might indicate that dropping highly correlated features is indeed beneficial for model performance.

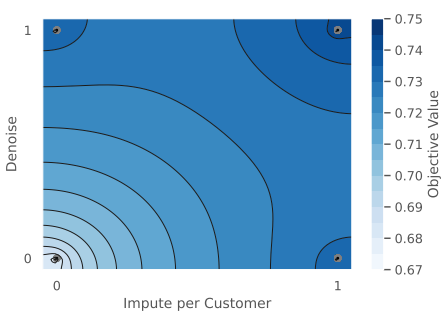
That denoising and groupwise imputation strategies might bring a significant performance benefit is further illustrated in the contour plot shown in Figure 5a, showing the value of $f(x, \theta)$ as pairs of hyperparameters are varied, filling missing values with Gaussian Process Interpolation. Between completely disabling and completely enabling these boolean parameters lies a significant difference of 0.08 in the final objective value as measured by the final challenge evaluation metric.

Continuous hyperparameters associated with the random forest regressor have a much more complex optimization landscape with many local optima as depicted in Figure 5b, comparing individual tree depth with total number of learning iterations. Despite this, a clear trend towards deeper trees and higher iterations is noticeable, showing that more complicated models tend to be required in order to achieve high evaluation performance. For the tree depth parameter, it is noteworthy this is the only parameter where the boundary of the optimization range has been reached, meaning that possibly tree depth beyond 11 could have further improved the results. However, values beyond 11 could not be explored as the resulting model would exceed the available GPU memory.

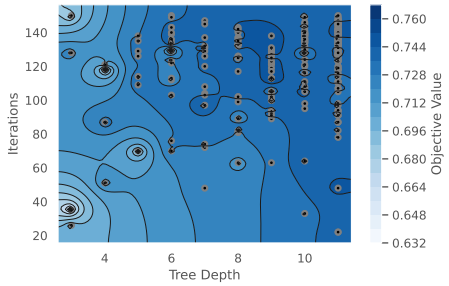
4.2 CNN and MLP Hyperparameter Tuning

Gann

In this section, the results of the hyperparameter tuning described in 3.9 are presented for the MLP and CNN.



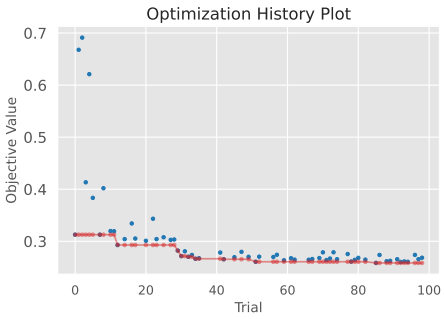
(a) Discrete dataset parameters



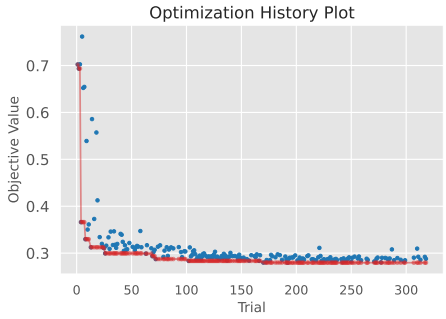
(b) Continuous parameters of GBDT classifier.

Fig. 5: Contour plots of Gaussian Processes depicting f-evaluations as two pairs of hyperparameters are varied (higher objective value is better).

Figure 6 shows the optimization progress for both MLP and CNN. On the x-axis, the number of trials is shown, while on the y-axis, the objective function is shown, which in this case is the BCE loss. The dots show the loss achieved by a model trained using a selection of hyperparameters. The red line shows the best objective value achieved in the current or earlier trials.



(a) Optimization Progress MLP

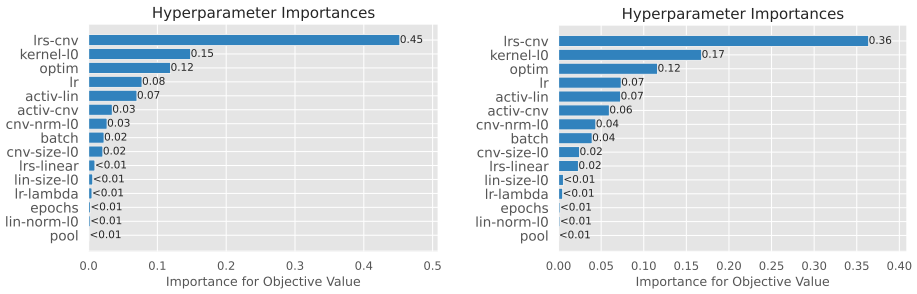


(b) Optimization Progress CNN

Fig. 6: Optimization history plots of deep learning related hyperparameters. The objective value is binary cross entropy (lower is better).

In both cases, Optuna quickly manages to select hyperparameters, which reduce the loss from about 0.7 to 0.3. It then further optimizes the hyperparameters to gradually achieve a loss of 0.279 in case of the CNN and 0.258 in case of the MLP. For the MLP the optimization was stopped at about 100 trials and in case of the CNN after about 300 trials as no further decrease of the objective value was to be expected.

Using Optuna, the importance of hyperparameters can be quantified. This is done using the fANOVA importance evaluator [9], which fits a random forest regression model that predicts the objective value, allowing for a feature importance analysis through the trained forest in a similar way as described in Section 3.1 for dataset feature importances. Figure 7 shows the importance of the hyperparameters for MLP and CNN.



(a) Hyperparameter Importances MLP (b) Hyperparameter Importances CNN

Fig. 7: Hyperparameter Importances

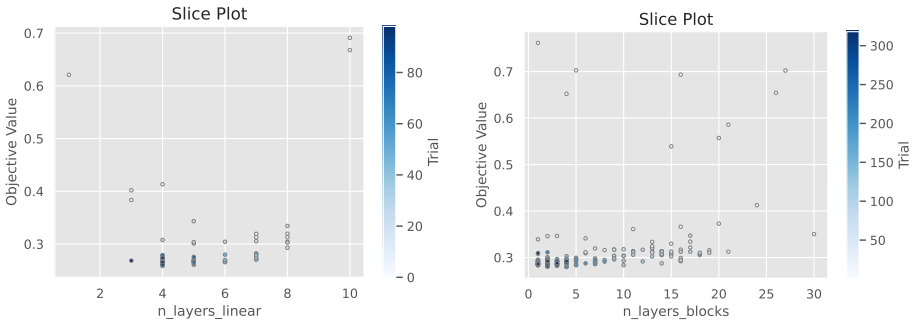
The most important hyperparameters in both cases are the total number of initial layers (linear layers in the case of the MLP and convolutional layers in the case of the CNN). The second most important hyperparameter for the MLP is the output size of the first linear layer. For the CNN it is the kernel size of the first convolutional layer. The next two most important hyperparameters in both cases are the optimizer and the learning rate.

Figure 8 shows the number of layers (linear and convolutional) at which Optuna achieved the best score.

In both cases, one can see a slight curvature upwards towards the left and right showing a clear optimal value in between.

Optimized MLP Architecture The optimized architecture consists of in total 5 linear layers, the first three layers followed by a ReLU activation, the final layer followed by a Sigmoid function. The output size of the layers are in order: 210, 130, 170, 280, 1. For training, the optimal values were batch size of 80, learning rate of 0.0003, multiplicative learning rate decay of 0.6 and AdamW as optimizer.

Optimized CNN Architecture The optimized architecture for the CNN consists of 4 convolutional layers and 7 linear layers. The convolutional layers have in order a kernel size of 9, 12, 5, and 6 and an output size of 300, 200, 180, and 140. Each convolution is followed by a ReLU activation and batch normalization is performed after the activation function on layer 2 and 4. The linear layers have in order an output size of 200, 220, 260, 280, 100, 130, 230, 1. Each linear layer



(a) Slice Plot Linear Layers MLP (b) Slice Plot Convolutional Layers CNN

Fig. 8: Slice Plots Layer Numbers

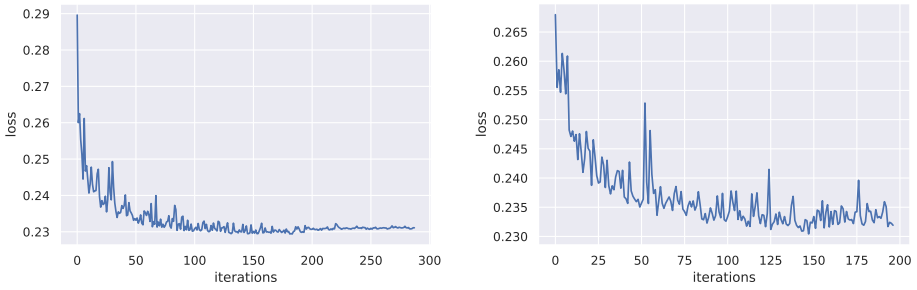
is followed by a ReLU activation, but the last activation is a Sigmoid instead of a ReLU. Between convolution and linear layers, a max pool with window size of 13 is performed.

For training, the optimal values were batch size of 112, learning rate of 0.0013, multiplicative learning rate decay of 0.89 and Adam as optimizer.

4.3 Training

Gann

Using the training method described in Section 3.8, the validation loss of the BCE after every 100 batches is visualized for the MLP and CNN model in Figure 9.



(a) BCE loss for the test set at every 100 batches during MLP training. (b) BCE loss for the test set at every 100 batches during CNN training.

Fig. 9: BCE loss curves during training of deep learning approaches.

Both curves show a steep decline in the beginning merging into a gradual decline with a slight incline towards the end. One can see that both models reach

their optimal classification performance at about 150 iterations, from where on the loss does not decrease but increase. It is also notable that the CNN experiences a greater variance in classification performance throughout the training process. This behavior can be observed up to the end of the training process. Both best performing models of MLP and CNN achieve a BCE loss of 0.23.

The members of the GBDT ensemble as described in Section 3.5 achieve a mean challenge metric of 0.77 and minimum BCE loss of 0.25 as measured on the respective validation folds. On average, peak performance on the validation set is achieved after building around 800 trees, after which the model overfits. For final prediction, trees built after this peak performance has been achieved are removed in order to obtain the state of the best model.

All models were trained on a single Nvidia Tesla V100 GPU with 32GB of available GPU memory using CUDA version 11.4.

4.4 Validation and Discussion

Figure 10 visualizes raw per-case assessment data individually for each method, as measured on a reserved testing dataset described in Section 3.10. While per-case performance is very similar for all methods, overconfident mispredictions resulting in high cross-entropy values are more common for the neural network based approaches. This observation is in line with findings in literature stating that neural networks are more likely to produce overconfident predictions than traditional learning approaches [1].

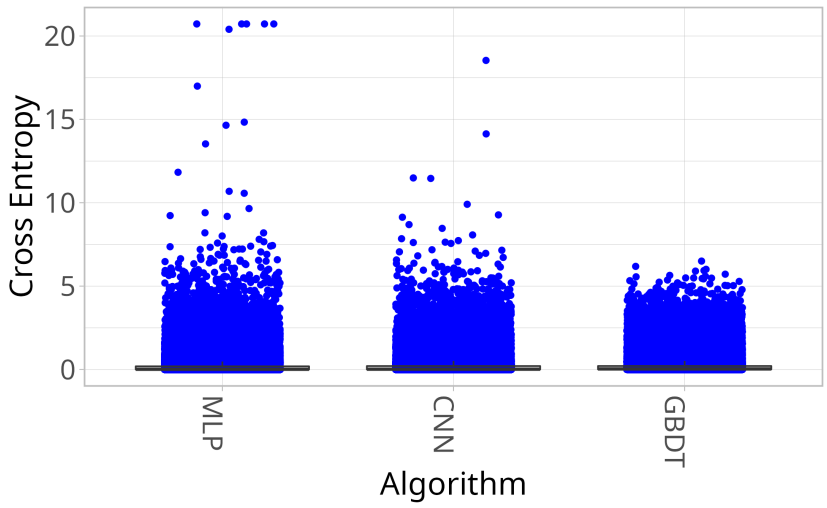


Fig. 10: Dot- and boxplots showing individual test-case performance for each method, as determined by binary cross entropy. Lower values are better.

Table 5 summarizes the final performance metrics achieved with each algorithm, where the first metric column describes the official challenge metric retrieved by submitting predictions to the competition website, and latter metrics were computed on our own test split of the data. The results show that despite GBDT achieving the highest metric values, all methods lie together closely. Notable in particular is that incorporating time series information in the model as done through the convolutional layers of the CNN had no measurable effect as compared to MLP and GBDT which did not use the time series. One of multiple plausible reasons for this might be that the time series are not long enough for the convolution to leverage its potential in pattern detection. As discussed in Section 3.7, the optimal kernel size determined for the first convolution is 9 which indicates that relevant patterns in the input data are about as long as the whole time series. As a result, a convolutional layer with kernel size similar to the channel length has a similar effect as a linear layer. In addition to that, as a result of a kernel size of 9 in the first layer and 12 in the second layer, most of the values in the resulting vectors are padded (for kernel size 12 in layer 2 only 2 values do not result from padding). This greatly diminishes the information on which a convolutional operation can train on. Combined with the observation that for the CNN Optuna optimized to attach 7 linear layers we believe the convolutions did not have a beneficial effect but were optimized to not negatively influence the predictive capabilities of the following MLP.

Method	Kaggle Score	Score (*)	AUC (*)
GBDT	**0.791**	**0.781**	**0.959**
MLP	0.781	0.777	0.957
CNN	0.783	0.776	0.956

Table 5: Final performance metrics of algorithms, with best score of column marked in bold. (*) marks metrics computed on the reserved testing dataset.

The Wilcoxon signed rank test described in Section 3.10 as performed on the cross-entropy metric on the reserved testing data revealed that no method was significantly superior to any other method above the significance threshold of 5%.

The winning submission of the entire challenge achieved a final metric of *0.809*, while our best submission was *0.791*. In terms of leaderboard ranking, comparison is difficult as multiple high scoring (> 0.80 score) approaches were publicly shared on discussion forums prior to the challenge deadline. Therefore, our best submission is around the center of the global challenge leaderboard.

Two main differences of the approaches presented in this report and the winning submissions to the challenge can be identified. First, many top performing approaches contain extensive manual feature engineering despite the unique challenge of features being anonymized. This includes deriving new kinds of features by means of aggregating the time series in certain ways, whereas we rely on the

Neural Network and GBDT architectures to implicitly derive useful combined features. Second, most winning approaches use weighted ensembles of different network architectures in a technique called stacking to take advantage of the fact that different machine learning approaches learn differently from the same data, and thus combine the advantage of multiple methodologies.

5 Conclusion

Grab, Gann

In this report, we investigated the problem of predicting whether credit card customers default on their loans given past performance indicators using a large, industry scale dataset provided by the lending institution American Express. In particular, we explored whether deep learning approaches could be successfully applied to the problem, whereby we mean whether deep learning approaches are able to surpass approaches known to work well on similar data such as GBDT. To answer this question, we trained GBDT, MLP, and CNN models using identical basic data preprocessing procedures. In both preprocessing and training, we heavily relied on automated hyperparameter optimization based on Gaussian Processes to systematically determine good parameters. Our results replicate prior work in the area indicating deep learning models approach similar performance, but fail to surpass GBDT in predictive accuracy [4]. In particular, we show that Neural Network approaches are more susceptible to overconfident misclassifications, again confirming prior work [1].

5.1 Limitations

Grab, Gann

In order to allow for fair comparison between the deep learning methods and GBDT, we used hyperparameter optimization in our preprocessing pipeline that itself uses GBDT for evaluation purposes. While the results of hyperparameter optimization are qualitatively sensible, they might thus be biased towards GBDT, giving the deep learning methods a disadvantage. However, as evaluating performance using deep learning approaches would have been too expensive computationally, the only alternative we see is manually specifying the data preprocessing pipeline, which also could be implicitly biased towards one specific method. A different consideration regarding the optimization is that the objective functions are non-deterministic in all cases and thus have some amount of variance attached, which we did not consider further in this report.

In this report, we only present two deep learning architectures. While it should be noted that the CNN architecture is mostly used for Computer Vision problems, we conjectured that the concept of convolutions might also bring a beneficial effect in the context of time series data, though we could not confirm this experimentally. In early experiments, different architectures such as Long Short-Term Memory (LSTM) and Recurrent Neural Networks were used, but results obtained were broadly similar to CNN and MLP and thus these experiments were not included in this discussion.

5.2 Future Work

Grab, Gann

Credit default prediction remains an interesting avenue for machine learning; however, it is important not to completely leave practical application out of sight. Considering a potential use case where, for example, high default risk customers are identified as early as possible and their contracts terminated, regulatory compliance and, more precisely, explainability should be a focus of future machine learning experiments. While GBDT are more explainable models than most, an interesting consideration could be whether alternative approaches with explanatory components such as Invertible Neural Networks could be used successfully in this context.

Returning to a more theoretical consideration, given the success of ensemble approaches stacking different kinds of machine learning methodologies, an interesting question is whether combining the approaches presented in this report in certain ways would lead to better predictive performance than the approaches individually.

Finally, in a broader picture than the specific problem of credit default prediction, it remains interesting to see that in learning from structured, tabular data, deep learning still does not live up to its high expectations set by unprecedented successes with processing unstructured, large-scale, or noisy data such as in computer vision or audio recognition. Determining why precisely this is the case and exploring alternative deep learning methodologies with increased potential to exploit tabular structures remains an exciting open area of research.

References

1. Abdar, M., Pourpanah, F., Hussain, S., Rezazadegan, D., Liu, L., Ghavamzadeh, M., Fieguth, P., Cao, X., Khosravi, A., Acharya, U.R., et al.: A review of uncertainty quantification in deep learning: Techniques, applications and challenges. Information Fusion **76**, 243–297 (2021)
2. Addo, P.M., Guegan, D., Hassani, B.: Credit risk analysis using machine and deep learning models. Risks **6**(2), 38 (2018)
3. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining. pp. 2623–2631 (2019)
4. Alam, T.M., Shaukat, K., Hameed, I.A., Luo, S., Sarwar, M.U., Shabbir, S., Li, J., Khushi, M.: An investigation of credit card default prediction in the imbalanced datasets. IEEE Access **8**, 201173–201198 (2020)
5. Beck, T., Grunert, J., Neus, W., Walter, A.: What determines collection rates of debt collection agencies? Financial Review **52**(2), 259–279 (2017)
6. Conover, W.J.: Practical nonparametric statistics, vol. 350. john wiley & sons (1999)
7. Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L., Muller, P.: Deep learning for time series classification: a review. CoRR **abs/1809.04356** (2018), `http://arxiv.org/abs/1809.04356`
8. Hand, D.J., Henley, W.E.: Statistical classification methods in consumer credit scoring: a review. Journal of the Royal Statistical Society: Series A (Statistics in Society) **160**(3), 523–541 (1997)
9. Hutter, F., Hoos, H., Leyton-Brown, K.: An efficient approach for assessing hyperparameter importance. In: Xing, E.P., Jebara, T. (eds.) Proceedings of the 31st International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 32, pp. 754–762. PMLR, Bejing, China (22–24 Jun 2014), `https://proceedings.mlr.press/v32/hutter14.html`
10. Katarya, R., Rastogi, S.: A study on neural networks approach to time-series analysis. In: 2018 2nd International Conference on Inventive Systems and Control (ICISC). pp. 116–119. IEEE (2018)
11. Kubat, M., Matwin, S., et al.: Addressing the curse of imbalanced training sets: one-sided selection. In: Icml. vol. 97, p. 179. Citeseer (1997)
12. Lee, J.W., Lee, W.K., Sohn, S.Y.: Graph convolutional network-based credit default prediction utilizing three types of virtual distances among borrowers. Expert Systems with Applications **168**, 114411 (2021)
13. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: Proceedings of the IEEE international conference on computer vision. pp. 2980–2988 (2017)
14. Lundberg, S.M., Erion, G., Chen, H., DeGrave, A., Prutkin, J.M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., Lee, S.I.: From local explanations to global understanding with explainable ai for trees. Nature machine intelligence **2**(1), 56–67 (2020)
15. Merton, R.C.: On the pricing of corporate debt: The risk structure of interest rates. The Journal of finance **29**(2), 449–470 (1974)
16. Moula, F.E., Guotai, C., Abedin, M.Z.: Credit default prediction modeling: an application of support vector machine. Risk Management **19**(2), 158–187 (2017)

17. Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A.V., Gulin, A.: Catboost: unbiased boosting with categorical features. Advances in neural information processing systems **31** (2018)
18. Schechtman, E., Schechtman, G.: The relationship between gini terminology and the roc curve. Metron **77**(3), 171–178 (2019)
19. Shwartz-Ziv, R., Armon, A.: Tabular data: Deep learning is not all you need. Information Fusion **81**, 84–90 (2022)
20. Trustorff, J.H., Konrad, P.M., Leker, J.: Credit risk prediction using support vector machines. Review of Quantitative Finance and Accounting **36**(4), 565–581 (2011)
21. Wiesenfarth, M., Reinke, A., Landman, B.A., Eisenmann, M., Saiz, L.A., Cardoso, M.J., Maier-Hein, L., Kopp-Schneider, A.: Methods and open-source toolkit for analyzing and visualizing challenge results. Scientific reports **11**(1), 1–15 (2021)
22. Yoo, W., Mayberry, R., Bae, S., Singh, K., He, Q.P., Lillard Jr, J.W.: A study of effects of multicollinearity in the multivariable analysis. International journal of applied science and technology **4**(5), 9 (2014)