Scuola universitaria professionale
della Svizzera italiana

SUPSI

University of Applied Sciences and Arts of Southern Switzerland
Department of Innovative Technologies

Data Challenge 3

PROJECT 2: DEFAULT ANALYSIS

Davide Gamba
davide.gamba@student.supsi.ch

Fabio Loddo
fabio.loddo@student.supsi.ch

Christian Pala
chirstian.pala@student.supsi.ch

Professor: Sandra Mitrovic
SUPSI, Lugano Switzerland

13/12/2022

Abstract

Project report for the seoond data challenge project on credit default analysis and prediction, with a focus on model generation and explainability. The topics covered in the report are:

1. *Problem definition*
2. *Preprocessing*
3. *Dataset analysis*
4. *Feature engineering and selection*
5. *Model selection*
6. *Dataset balancing*
7. *Hyper-parameter tuning*
8. *Global model explainability*
9. *Local model explainability*
10. *Experiments and results*
11. *Critical reflections*
12. *Conclusions*

We developed a robust pipeline to test many preprocessing and balancing strategies on three classes of models:

1. *trees, in particular, decision tree, random forest, gradient and extreme gradient boosting*
2. *neural networks, in particular, multilayer perceptron and convolutional neural network*
3. *other models, k-means, logistic regression, naive bayes and support vector machine classifiers.*

After testing we refined our pipeline to run efficiently and produce good results with the best models we found, by testing a few strategy combinations or selecting the dominant one if available. We then tuned the generated models using different approaches and finally explained the models, both globally on which features the models use to classify and how important they are, and locally for interesting instances where miss classification occurred.

The entire project was developed with the python language, version 3.9.13, following the material provided by professor Mitrovic during the Data Challenge 3 course [1].

Table of Contents

List of Figures

1. Problem definition

We received, from professor Mitrovic, a dataset comprising clients' data along with information aimed at predicting whether they would fail to repay their debt (i.e. the client defaulted). This was a binary classification, supervised machine-learning project, hence the observations were labeled with the true outcome, at the end of the period under consideration, spanning six months.

We had 23 features at our disposal in total, organized as follows by professor Mitrovic, in her introduction to the project [1]:

- ID: customer-id
- X1: balance limit for the customer
- X2: gender encoded as: (1 = male; 2 = female).
- X3: education encoded as (1 = graduate school; 2 = university; 3 = high school; 4,5,6 = others; 0 = missing value)
- X4: marital status encoded as (1 = married; 2 = single; 3 = others; 0 = missing value)
- X5: age (years)
- X6-X11: Past monthly payment records, encoded as (-2 = did not use; -1 = paid in full; 0 = using revolving credit; 1 - 9 = payment late by 1 - 9 months respectively)
- X12-X17: previously billed amounts
- X18-23: previously paid amounts
- default payment next month (target label) encoded as (0 = no default, 1 = default)

Our main objective was to implement a complete workflow, from preprocessing to model evaluation and explanation. In particular, we were asked to focus on model generation, experimenting with different classifiers and their requirements. We were also tasked with handling the skewness of the features and in particular of the target label. Finally, model understanding was also a key part of the project, we had to implement both different global and local interpretations of our final models, comparing and explaining their results.

2. Preprocessing

The key points in the preprocessing phase of the project were:

1. understand and clean the data
2. handle incongruent data and missing values
3. handle skewness
4. perform appropriate scaling and normalizing
5. explore and visualize the dataset to better understand it.

2.1 Loading the dataset

The dataset `Project 2 dataset` was provided to us in `.xls` format. We loaded it into our script and converted it into a pandas DataFrame [2] using the xlrd library [3].

2.2 Data cleaning

After inspecting the dataset, we noticed the naming convention often did not help in understanding the features, so we renamed them based on the available information. For details see our preprocessing implementation in our code-base [4].

2.3 Training, validation, and testing splits

From the experience in the first data challenge 3 project, one of the first things we did was agree on an 80:20 split between training and testing, also splitting the training with an 80:20 split for the validation dataset, to ensure there would be no data leakage influencing our final results. This was very relevant already in the preprocessing phase, as many of the methods we used had to be fitted on the training set, to then transform the validation and testing sets correctly.

2.4 Handling of missing values

After a clarification from professor Mitrovic, we determined the values encoded as 0 in the categorical marital status and education features were missing values. Since we were going to work with many models, possibly preferring different approaches, we decided on a general philosophy to first implement a broad set of strategies and later streamline the process. For the missing values we implemented the following methods:

- drop (default method), drops all missing values
- most frequent imputation, Imputes the missing values with the most frequent value in the column.
- supervised imputation, uses a random forest classifier to impute the missing values.
- unsupervised imputation uses the nearest neighbor to impute the missing values.

We used the pandas library [2] to implement dropping and imputing by the most frequent value, and scikit-learn [5] for the supervised and unsupervised methods. We later determined that for our setup, imputing the missing values with the most frequent observation was often the best strategy.

2.5 Feature skewness

After noticing many of the numerical features were strongly positively skewed, using pandas [2], in the `features_skewness_handling.py` file we implemented a series of methods to transform the features to approximately normal distributions. This was done to help models that prefer or assume such distributions. Those methods include:

- power transformations using box-cox and yeo-johnsons from scikit-learn [5]
- logarithmic transformations applying $log(x + 1)$ to the variable of interest.
- Square and cubic root transformations

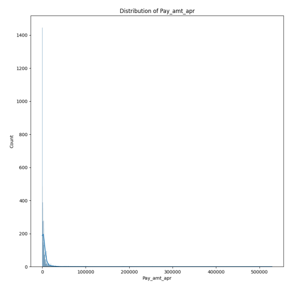
After testing the transformations performances we settled on the yeo-johnsons method [6], which is a refinement of the box-cox [7] transformation, for the mathematical implementation we point you to the papers. We selected the former in particular since in our case it was necessary to deal with negative values, which makes yeo-johnson more convenient, but also due to slightly better performances.

2.6 Scaling

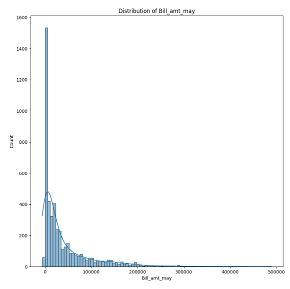
We scaled the data using the methods in `scaling.py`, keeping the training, validation, and testing separation in mind. We implemented a standard-, a robust- and a min-max scaler as well as a normalizer all using the scikit-learn library [5]. Running the pipeline a few times, we settled on the normalized robust scaler, as it provided the best in class results, based on the way we grouped models, which we will present in chapter 5.

3. Data analysis

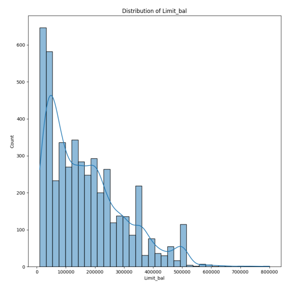
In parallel with our preprocessing of the dataset, we also performed the initial exploratory data analysis and we noticed some important observations. Plotting the numerical distributions, it's clear how skewed they were, see figure 1.



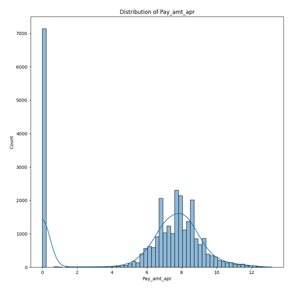
(a) April paid amount



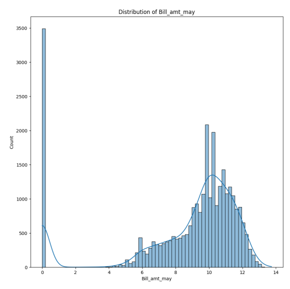
(b) May billed amount



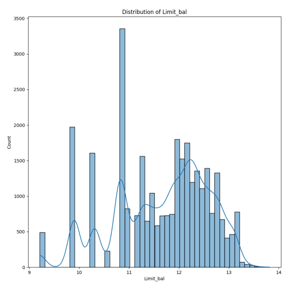
(c) Limit balance



(d) April paid amount log transformed



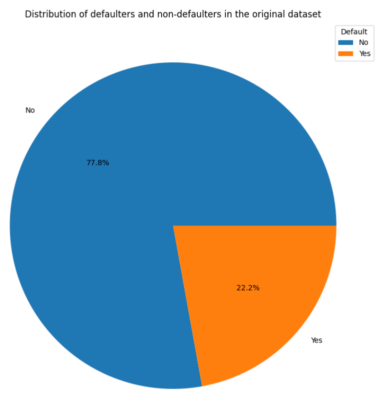
(e) May billed log transformed



(f) Limit balance log transformed

Figure 1: Skewness in the dataset

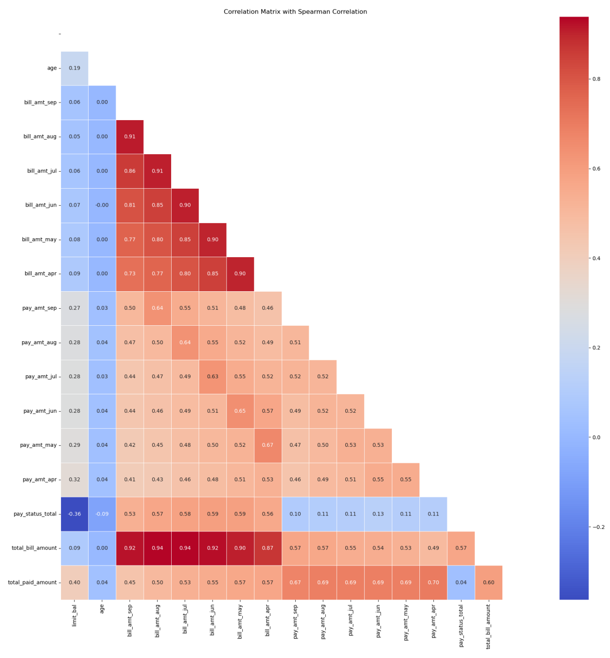
Plotting the target variable, quickly shows that the dataset is unbalanced, see figure 2.



(a) Default distribution

Figure 2: Target Variable "default" distribution

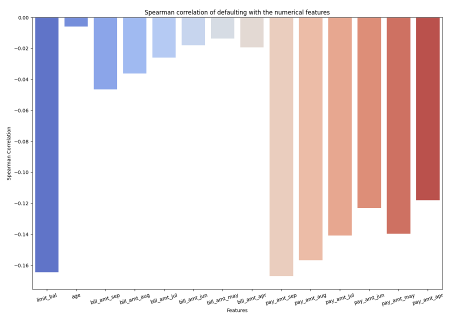
Plotting the correlation matrix of the numerical features, we confirmed that bill amounts in particular, but also paid amounts are correlated with each other, which could be a problem for some models, see figure 3.

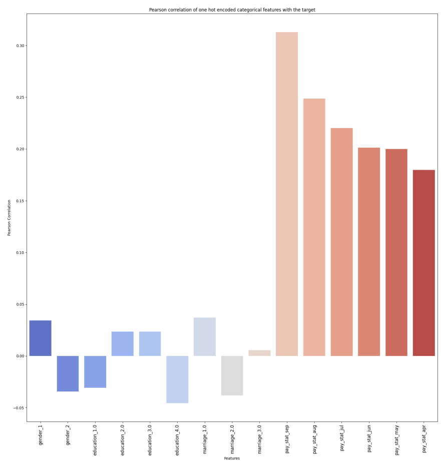


(a) Correlation matrix

Figure 3: Feature correlation

We also noticed how none of the features are strongly correlated with the target, so a fairly comprehensive portfolio would be necessary to obtain good results with our classifiers, see figure 4.





(a) Numerical correlation with target

(b) Categorical correlation with target

Figure 4: Correlation with the target variable

4. Feature engineering and selection

Feature engineering and selection were not the main topic of interest for this project, since we explored that part of a machine learning project in the first data challenge 3 project. We found a way to quickly implement both in the project pipeline, in a way that made sense, without spending too much of our time budget on them.

4.1 Feature engineering

Since we had sequences of six observations for monthly billed amounts, paid amounts, and payment statuses, correlated amongst each other as noted in chapter 3, we decided to create cumulative values, namely:

1. cumulative payment status
2. total billed amount
3. total paid amount

The initial idea was to help simple classifiers by later performing feature selection and leaving only the cumulative features for those models to analyse. However, after testing there was often a slight overall improvement for the complex models when leaving these features in the dataset as well. This had some other consequences we figured out later on and will present in chapter 8.

4.2 Feature selection

For the more biased models, we create a procedure to simplify the dataset by removing the monthly payment status, billed amount, and paid amount, leaving only the cumulative values we generated in the feature engineering part of the project. This helped us better understand how our white-box model, a simple decision tree, later used the features to classify the observations. We did not proceed with other feature selection methods since during our data exploration, we noticed we would need a large bundle of features to get a good classifier, hence removing more did not seem worth-wile.

5. Model generation

One of the main objectives of this project was to focus on model generation and use a variety of classifiers. We decided to proceed by dividing the work into three broad groups of models:

1. tree-based algorithms
2. neural networks
3. other classifiers

For each category, we tried selecting the representative algorithms.

5.1 Evaluation metric

Our initial evaluation metric for the project was the $F1$ score, defined as:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN}$$

Our dataset was unbalanced and the minority class, defaulting on repayment, is likely very relevant, hence the F1 score is a better choice compared to the usual accuracy. An argument can be made for other metrics and we later expanded our evaluation to include others of potential interest for a potential owner of the dataset.

5.2 Tree-based algorithms

The first kind of model we implemented was tree-based. Since they do not require particular preprocessing steps, other than dealing with missing values, we could train and validate the base models on the raw data. This was helpful in later selecting which strategies were helping and which we could neglect or ought to remove. In particular, we obtained the following results in table 9.1 using the base models from the scikit-learn [5] and xgboost [8] libraries. We used the fully feature-engineered dataset of 26 features. The results with 23 features are very similar, but usually slightly worse, depending on the model. Note that since these were preliminary results, we did not perform cross-validation, hence there is some variability. We provided a random seed on all models to reproduce our findings.

5.3 Neural networks

The second class of models we selected for the classification are neural networks, which we implemented with the Keras library from TensorFlow [9]. Given model explainability was one of the requirements of the project, having a neural network as a "black-box" model seemed appropriate. We started with a fully connected network, but later expanded the selection to add a convolutional neural network. The idea behind using a convolutional network architecture was to give the model more information on the monthly features like payment status, billed amount, and paid amount by providing the value of the moving average. Our theory was, being able to perform a moving average with a one-dimensional convolution and have a larger field of view should have helped. As a side note, we are also studying convolutional networks in professor Giusti's Computer Vision course, so we wanted to try implementing them for this project as well.

5.3.1 Dense neural network initial architecture

Our initial neural network had a simple sequential structure with an input layer, 2 hidden layers, and an output layer, all fully connected. We used Relu:

$$Relu(z) = max(0, z)$$

as the activation function for the inner layers and the sigmoid:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

for the outer layer. We also added a 0.1 dropout after each layer to make the training a bit harder, and hopefully, generalize better. The architecture is summarized in figure 5

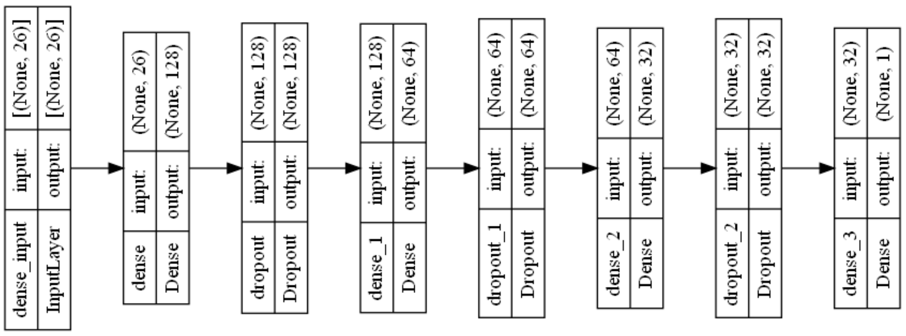


Figure 5: Dense neural network architecture

5.3.2 Convoluted neural network initial architecture

For this neural network, the idea was to first use the convolutional structure to generate the features, and then evaluate them with a simple sequential model similar to the one used for the fully connected neural network. We performed a one-dimensional convolution with a kernel of size

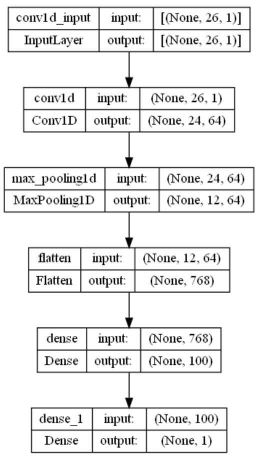

conv1d_input | input: | [(None, 26, 1)]
InputLayer | output: | [(None, 26, 1)]
conv1d | input: | (None, 26, 1)
Conv1D | output: | (None, 24, 64)
max_pooling1d | input: | (None, 24, 64)
MaxPooling1D | output: | (None, 12, 64)
flatten | input: | (None, 12, 64)
Flatten | output: | (None, 768)
dense | input: | (None, 768)
Dense | output: | (None, 100)
dense_1 | input: | (None, 100)
Dense | output: | (None, 1)

Figure 6: Convoluted neural network architecture

3 and we added a max-pooling layer to provide a more abstract representation of the features. The other layers follow the same design pattern as the fully connected neural network. The network architecture summary is provided graphically in figure 6.

5.4 Other models

We grouped the k-nearest-neighbor, logistic regression, gaussian naive-bayes, and support vector machine classifiers together in the ”other models” category since they are all models which require some preprocessing, either in the form of scaling, normalizing, or transforming the features to be approximately normally distributed. The models which performed the best at this stage were the gaussian naive-bayes with an $F1$ score of **0.517** and the support vector classifier with an $F1$ score of 0.515, logistic regression was the worst performing model with an $F1$ score of 0.44, see the modeling directory of our code base for more details [4].

6. Dataset balancing

We tested a variety of methods from the imbalanced-learn library [10] and also from smote-varients [11] which we later did not use. This exploration included random under and over-sampling, SMOTE oversampling and its variants, together with combinations of over and under sampling like SMOTEENN and SMOTE tomek links. We obtained the best F1 scores, with the design choices presented in the previous chapters, using SVM-smote. SVM-smote is a smote variant developed in 2009 [12], which approximates the border area between the classes, in our case defaulters and non-defaulters, with a support vector machine. It then generates synthetic data based on said approximation.

6.1 Tuning the sampling strategy

After trying a variety of over-samplers, and combinations of both, by design with SMOTEEN and SMOTE Tomek, but also with pipelines of over and under-sampling, we settled on a pure over-sampling strategy with SVM-smote of 0.88, which means having an 88:100 split between defaulters and non-defaulters in the dataset by generating the missing observations for the minority class, the defaulters, until that ratio is reached. Intuitively having a 1:1 ratio with a balanced dataset might seem like the best solution, but it also means creating more synthetic, and hence potentially problematic, data. This is why it is usually not recommended in the SMOTE documentation.

6.1.1 Under-sampling

Due to the time complexity in the tuning and explainability portions of the project, we decided to take a small performance hit of approximately 0.5 on the $F1$ scores of our best models and focus on the dataset created by under-sampling. We therefore also tuned the sampling strategy for under-sampling, and found a 1:2 ratio to be optimal for the models we selected. This choice was particularly necessary for the tuning of the neural networks and for some routines in the explainability part of the project, which require computations on permutations of the dataset and are therefore quite time consuming.

6.2 Model selection for tuning and explaining

Based on the above design choices, we selected the best in class model for each category as the main focus of the next steps, they ended up being:

1. Gradient boosting classifier for trees
2. Convoluted neural network for networks
3. Support vector machine classifier for the other models.

7. Hyperparameter tuning

In order to tune the hyperparameters of the models, we utilised a few different strategies. The simplest one consisted in tuning the models using GridSearchCV from scikit-learn [5], which we employed for tuning the most biased models on the simplified dataset. A second approach was inspired by one of professor Cannelli's Advanced Machine Learning lectures, where he presented us a relatively new implementation in scikit-learn of successive halving [13], HalvingRandomSearchCV, which we used as a tuner for the gradient boosting and support vector machine classifiers, it has the benefit of being aggressive and fast, which helped us keep the run times moderate. Finally for the more complex tuning tasks, we used Optuna [14] and Hyperopt [15], two open source frameworks designed for hyperparameters optimization in machine learning. We tuned all the models previously introduced, but in this section we will focus in particular on the tuning of Neural Networks (Dense and Convolutional), gradient boosting trees and support vector machines.

7.1 Neural networks

In tuning Neural Networks, it was important to tune not only the parameters of each layer, but also to pay attention to the overall architecture of the network.

7.1.1 Dense neural network

This type of neural network was only composed by fully connected layers plus the output layer which always contained only one neuron with a sigmoid activation function.

The search space we provided to the tuners was the following:

- *Number of layers*: Between 1 and 6 (without considering the output layer)
- *Optimizer*: chosen between "Adam", "SGD" and "RMSProp"
- *Learning rate*: chosen in a range between 0.00001 and 0.1
- *Epochs*: chosen in a range between 10 and 100
- *Batch Size*: chosen in a range between 8 and 128

For each layer we also tuned:

- *Neurons*: the number of neurons, chosen with a value from 10 to 300.

- *Activation function*: chosen between "tanh" and "relu"

We always used binary cross entropy for the loss. Usually a six layer structure with neurons around 100 per layer were preferred, interestingly the hyperbolical tangent was the best activator, and this proved to be true also for the convolutional neural network.

7.1.2 Convolutional neural network

For the convolutional neural network, we performed different test on whether several convolutional layers would be beneficial in predicting the target feature. We found, however, that the best results were reached by having an architecture composed by only one convolutional layer as the input layer, followed by several dense layers.

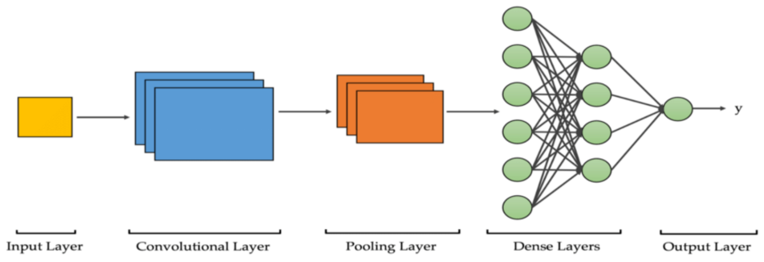


Figure 7: Overall convolutional network architecture

Once we found said architecture to be promising, we tuned its parameters using Optuna. The convolutional part of the network was made up of a 1D convolutional layer, a MaxPooling layer and a Flatten layer used to flatten the input for the subsequent dense layers.
The tunable parameters for the overall network were:

- *Number of layers*: Between 2 and 6 (without considering the output layer)
- *Optimizer*: chosen between "Adam", "SGD" and "RMSProp"
- *Learning rate*: chosen in a range between 0.0001 and 0.01
- *Epochs*: chosen in a range between 10 and 100
- *Batch size*: chosen in a range between 8 and 128

While for the convolutional layer we tuned:

- *Filter*: chosen in a range between 10 and 200
- *Kernel Size*: chosen in a range between 1 and 10
- *Pool Size*: chosen in a range between 1 and 10

The Dense layers were tuned in the same way as with the dense network. Filters around 100 with a kernel size around 7 or 8 yielded the best results.

7.2 Gradient boosting and support vector machine classifiers

With the advantage of a smaller dataset produced by undersampling, using both HalvingRandomSearchCV and Hyperopt we performed two separate tunings on the available hyperparamters provided by the scikit-learn [5] library for both models, the procedure is in our codebase [4] in the tuning module folder, `[model_name]_[_hyperopt or _successive_halving]_tuner.py` respectively.

8. Model explainability

To explain the three black-box models we generated, namely:

1. Gradient boosting classifier
2. Convoluted neural network
3. Support vector machine classifier

we used a variety of methods. First, we adopted a top-down approach, understanding how a white-box model and our black-box models used the features we provided, and how important they were for the classification, then we looked at particular observations for more fine-grained explanations of cases of interest, with our local explanations.

8.1 Global model explainability

To explain the models globally we used the following methods:

1. a white-box model in the form of a decision tree
2. permutation feature importance
3. partial dependency plots
4. surrogate models
5. global explanations using Shapley values and the SHAP library [16]

8.1.1 White-box model

We tuned a decision tree classifier on the simplified dataset, after handling missing values, before scaling and balancing to have more meaningful thresholds.

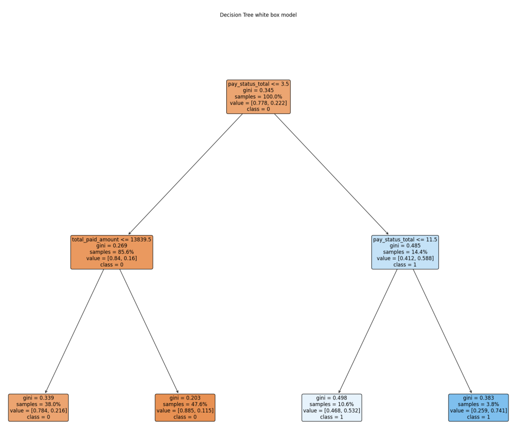
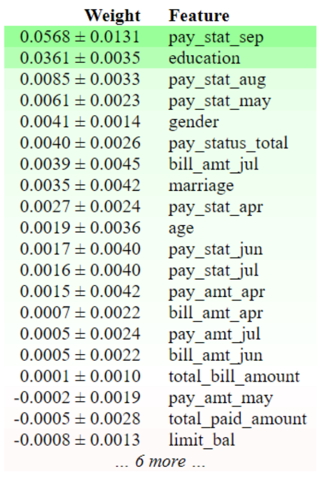


Figure 8: White-box decision tree

The precision, recall, and $F1$ scores, 0.592, 0.332, and 0.426 respectively, as well as figure 8, show the model is too simple for the dataset. Nonetheless, we can start making some hypotheses. As we expected, the payment status is a key feature. We omitted the feature importance, found in our codebase [4] since the model only cares about the payment status (0.8) and the total paid amount (0.2), but this reinforces the point above. For a more detailed explanation, and to understand the models we selected, we need more sophisticated approaches.

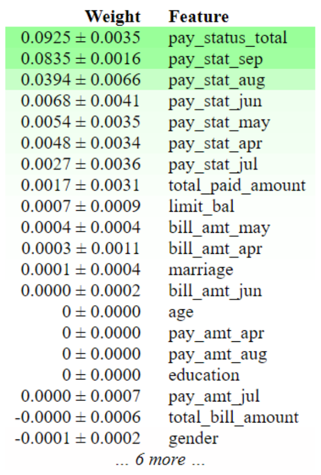
8.1.2 Permutation feature importance

One way to measure how important a feature is for a model, as we have seen in class [1], is to scramble one feature at a time and check how the model performance changes. We implemented the procedure for the gradient boosting and support vector machine classifiers using the ELI5 library [17] and our own custom implementation using 5-fold stratified cross-validation, for the convolutional neural network. We found this article [18] very helpful, particularly on how to quantify the performance variation. The results are summarized in figure 9.



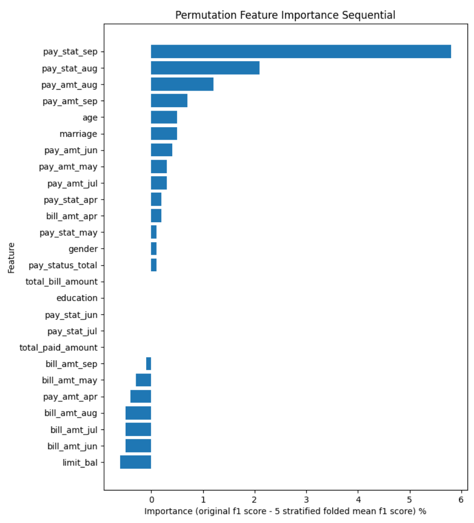
Weight	Feature
0.0568 ± 0.0131	pay_stat_sep
0.0361 ± 0.0035	education
0.0085 ± 0.0033	pay_stat_aug
0.0061 ± 0.0023	pay_stat_may
0.0041 ± 0.0014	gender
0.0040 ± 0.0026	pay_status_total
0.0039 ± 0.0045	bill_amt_jul
0.0035 ± 0.0042	marriage
0.0027 ± 0.0024	pay_stat_apr
0.0019 ± 0.0036	age
0.0017 ± 0.0040	pay_stat_jun
0.0016 ± 0.0040	pay_stat_jul
0.0015 ± 0.0042	pay_amt_apr
0.0007 ± 0.0022	bill_amt_apr
0.0005 ± 0.0024	pay_amt_jul
0.0005 ± 0.0022	bill_amt_jun
0.0001 ± 0.0010	total_bill_amount
-0.0002 ± 0.0019	pay_amt_may
-0.0005 ± 0.0028	total_paid_amount
-0.0008 ± 0.0013	limit_bal
... 6 more ...	

(a) Gradient booster



Weight	Feature
0.0925 ± 0.0035	pay_status_total
0.0835 ± 0.0016	pay_stat_sep
0.0394 ± 0.0066	pay_stat_aug
0.0068 ± 0.0041	pay_stat_jun
0.0054 ± 0.0035	pay_stat_may
0.0048 ± 0.0034	pay_stat_apr
0.0027 ± 0.0036	pay_stat_jul
0.0017 ± 0.0031	total_paid_amount
0.0007 ± 0.0009	limit_bal
0.0004 ± 0.0004	bill_amt_may
0.0003 ± 0.0011	bill_amt_apr
0.0001 ± 0.0004	marriage
0.0000 ± 0.0002	bill_amt_jun
0 ± 0.0000	age
0 ± 0.0000	pay_amt_apr
0 ± 0.0000	pay_amt_aug
0 ± 0.0000	education
0.0000 ± 0.0007	pay_amt_jul
-0.0000 ± 0.0006	total_bill_amount
-0.0001 ± 0.0002	gender
... 6 more ...	

(b) Support vector machine



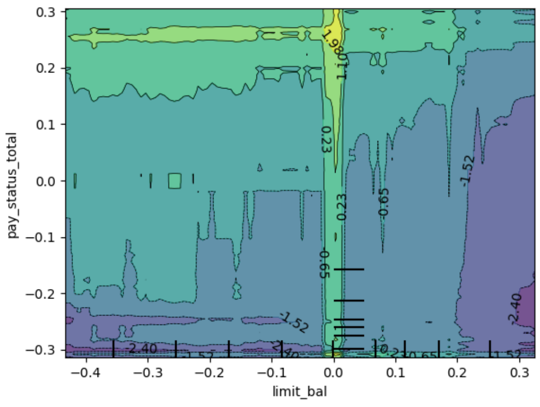
(c) Convolutional neural network

Figure 9: Permutation feature importance

Comparing the models in figure 9 we can again see the importance of the payment status. The convoluted neural network ignores the cumulative value, whereas the other two consider it very important, probably due to the neural network being more sophisticated, and this proved to be important later on. Another point that emerges from figure 9 is that some features would have improved the score if removed based on this 5-fold trial, but all values other than the payment status for September are close to 0 either way.

8.1.3 Partial dependence plots

As seen in class [1], we can use partial dependency plots to examine how we expect the target response to vary, in our case the change from non-defaulter encoded as 0 to defaulter encoded as 1, given a change in the value of our feature or features when we marginalize all the others. There are some assumptions one should be careful with however since it is assumed that the features for which you compute the partial dependence are independent, which is often not the case also for our dataset. We present the target response to limit balance and the total payment status in the case of the gradient boosting classifier in figure 10



]

Figure 10: Limit balance and payment status vs defaulting for the gradient booster classifier

Figure 10 showed to us how dominant the payment status was for the gradient boosting algorithm, regardless of the customer's available balance, arguably one of the more relevant features, it will classify someone with a good payment track-record as not going to default regardless of his limit. Keep in mind the values are normalized in the dataset, so the absolute numbers are not relevant. The single relationships are also interesting, especially for the support vector machine classifier where the trend is very apparent, higher available balance, education and being female equates to a lower default risk for the model, with different degrees of importance represented by the magnitudes. We point you to the partial_dependencies_results folder in our codebase [4] for a more in depth view. A final note on partial dependency plots, we were not able to interpret our convolutional neural network, from our reasearch this is a compatibility problem between scikit-learn and keras.

8.1.4 Global surrogate model

Global surrogate models are an ingenious way to explain black-box models. The idea is to use an interpretable model as a proxy for the model we would like an explanation from. This is done by training the surrogate on the results obtained by the black-box model, using the same observations. The hard part is finding a surrogate model good enough to explain most of the black-box model's outputs, for this we used the R^2 metric defined in the class slides [1]. We trained tuned logistic regression and random forest classifiers on both the fully connected and convolutional neural networks, since we had to exclude them from the partial dependency plots. After tuning, the results for the fully connected network were decent, with an R^2 of approximately 0.7 and 0.8 respectively, with the top feature being the payment status in September. We then tried with the convolutional network but it was harder for the surrogates to replicate. We present the results of the random forest feature importance, with an R^2 of 0.708 in figure 11, but we find the results ta bit surprising, in particular it seems strange that limit balance is the most important feature. We decided to focus our local analysis also on the instances where the surrogates were making the biggest mistakes and see if we could learn something, see chapter 8.2

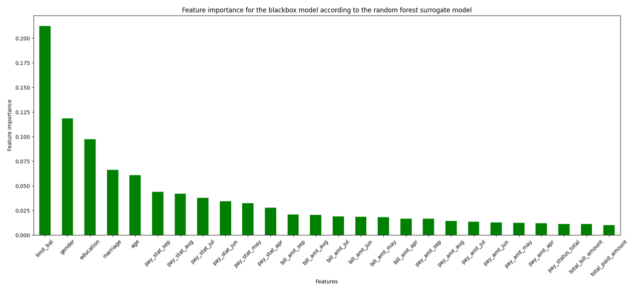


Figure 11: Convoluted neural network feature importance according to the random forest surrogate

8.1.5 Global explanation with SHAP

Using the SHAP library's [16] summary plot we plotted all instances of the model's explanations together in a dot plot for the gradient boosting model.

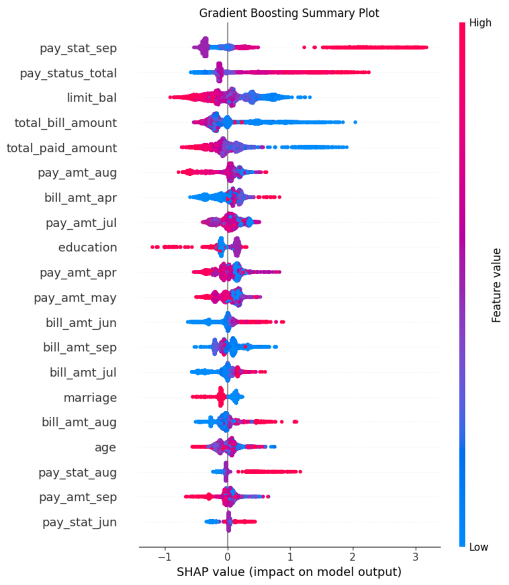
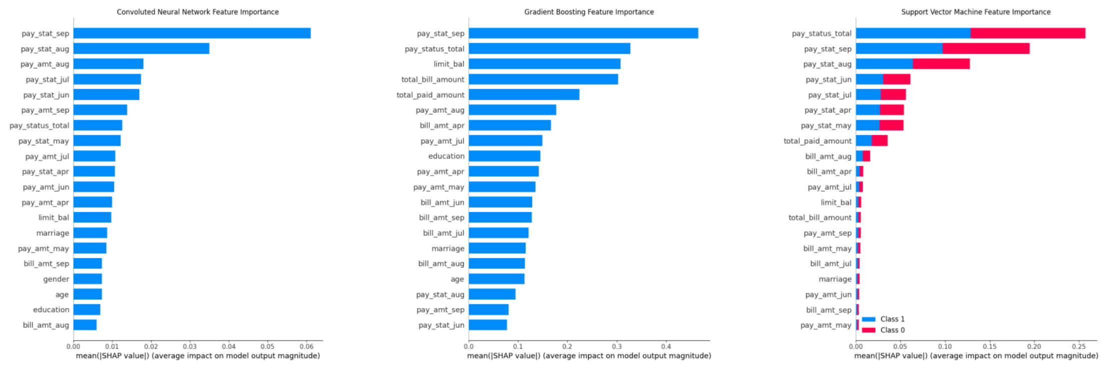


Figure 12: Gradient boosting classifier summary plot

We can see in figure 12 the payment status of September, the last month considered, remains very relevant, for high values the model will likely predict that this person will default, as expected by the previous analysis. Another interesting fact is that the model believes customers with a low total billed and paid amount are more likely to default, if it's correct then it seems defaulters are more likely to be smaller clients, which would mean the owner of the dataset is already doing a good job. Finally we find it noteworthy that all features can be quite relevant, depending on the specific instance.



(a) Convolutional neural network (b) Gradient booster (c) Support vector machine

Figure 13: Feature importance using SHAP

The plots in figure 13 represent how much each features influenced our model prediction according to the SHAP explainer. For the convolutional neural network and support vector machine the predictions are basde on a sample of the dataset using SHAP's kmeans sampler, to speed up the execution time. The SHAP feature importances agrees that the last payment status (September) had great influence on all the models. The pay statuses of the prior months are not as decisive but still relevant. The biggest difference amongst the models is again the importance of the total payment status, which should start to highlight the problem with the surrogate models.

8.2 Local model explainability

A local explanation is, as the name suggest, the local approximation of the model's behaviour close to an observation of interest. While the model may be very complex globally, it is easier to find a good approximation in a local neighbourhood. Since it is often not possible to directly interpret most machine learning models, like gradient boosting trees and neural networks (which we both used in this project), this is another way to understand their output at a more observation based level. In order to explain the models locally, we used the following methods:

1. Local explanations using Shapley values and the SHAP library [16]: The SHAP library is described as follows in the documentation: "SHAP (SHapley Additive exPlanations) is a game theoretic approach to explain the output of any machine learning model", we used the same method locally that we used globally on the whole dataset, where a single dot was the result of an observation.

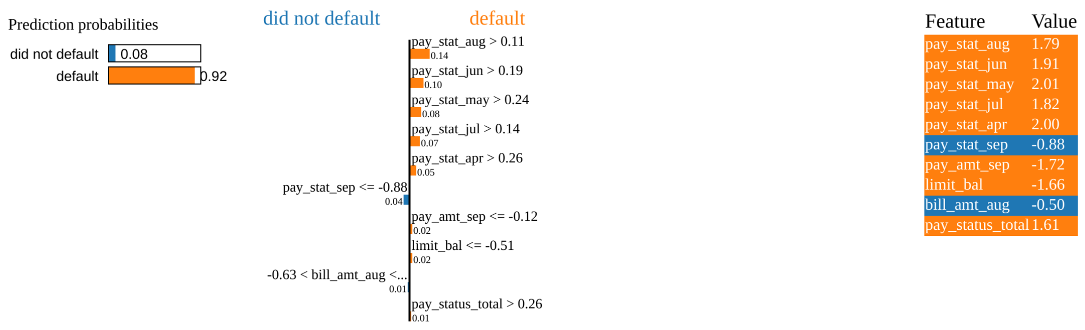
2. Local explanations using the LIME library [19]: When using the LIME library a local surrogate model is trained by sampling data in the neighbourhood, and feature selecting using lasso or ridge regressions. This surrogate model can be interpreted and is used to explain an individual prediction. In our case we used the LIME explainer for tabular data from the LIME library.

8.2.1 Predictions Analysis

For the local analysis we thought it would be interesting to check the observations which performed particularly poorly in the global surrogate model evaluation. We ranked the observations by the largest mistake the surrogate models made, compared to the output of the black-box model. In order to compare the two, both the black-box model and the surrogate predicted the probabilities of defaulting and non-defaulting.

The model used as a base for the surrogates was the convolutional neural network but to further extend the analysis we also examined how the other two black box models, the gradient boosting and the support vector machine classifiers, interpreted the data.

8.2.2 First Local explanation



Feature	Value
pay_stat_aug	1.79
pay_stat_jun	1.91
pay_stat_may	2.01
pay_stat_jul	1.82
pay_stat_apr	2.00
pay_stat_sep	-0.88
pay_amt_sep	-1.72
limit_bal	-1.66
bill_amt_aug	-0.50
pay_status_total	1.61

Figure 14: Lime explanation of the black box model (convoluted neural network) for the observation 4313.

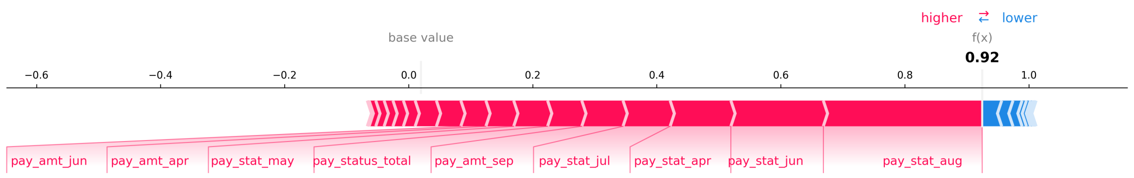


Figure 15: Shap values of the convoluted neural network for the observation 4313.

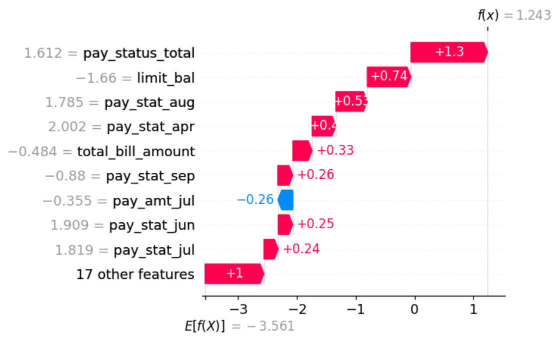


Figure 16: Shap values of the gradient boosting classifier for the observation 4313.

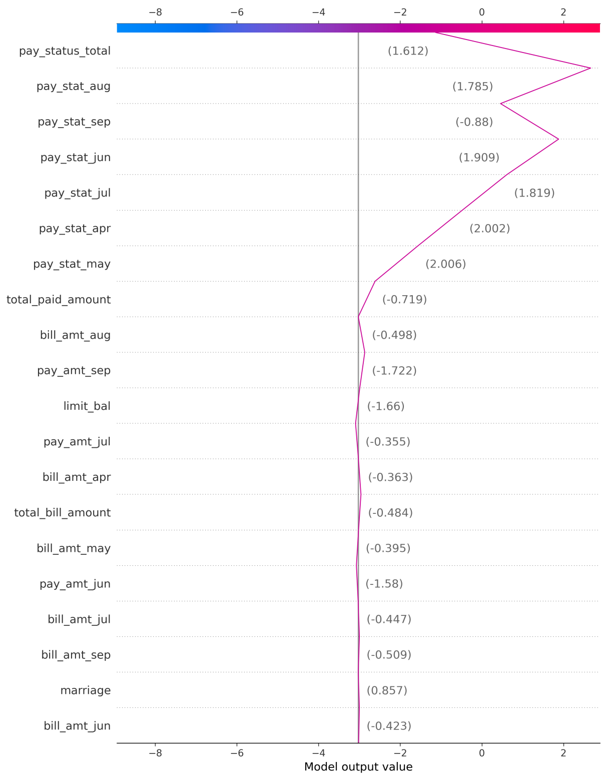
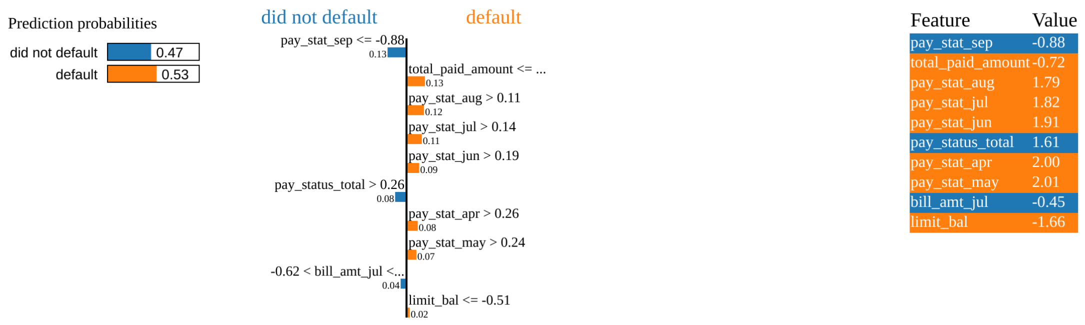


Figure 17: Shap values of the support vector machine classifier for the observation 4313.

In figure 14 we see that, for this particular observation, the neural network outputs with a high probability that the client will default. In the figures 15, 16 and 17 we can see that the three black box models reached more or less the same conclusion, and the values of pay status august and pay status total were highly correlated with the choice of the default class. In this case all of the models predicted the wrong class compared to the ground truth, so we are analyzing a false positive prediction.



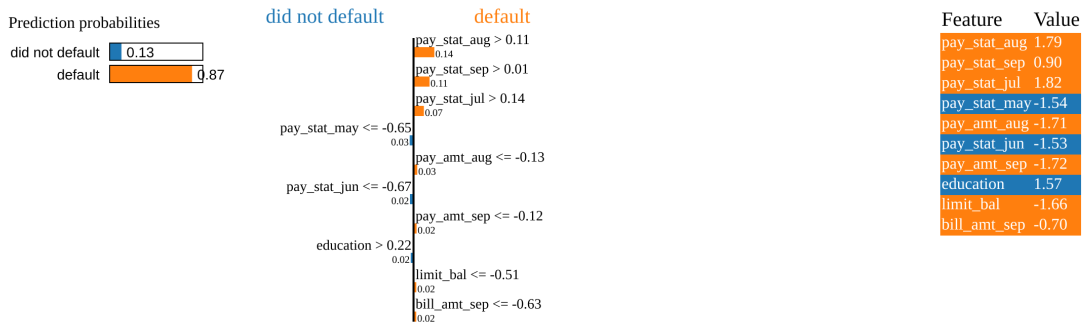
Feature	Value
pay_stat_sep	-0.88
total_paid_amount	-0.72
pay_stat_aug	1.79
pay_stat_jul	1.82
pay_stat_jun	1.91
pay_status_total	1.61
pay_stat_apr	2.00
pay_stat_may	2.01
bill_amt_jul	-0.45
limit_bal	-1.66

Figure 18: Lime explanation of the surrogate model using logistic regression for the observation 4313.

We can now compare the two predictions made by the black box model and the logistic regression surrogate in the figures 14 and 18. Looking at the weights attributed by LIME we see that the problem in this case was the total payment status feature which, contrary to the black box model, is considered to be a relevant indicator of the non default class by the surrogate model, leading to a mistake in copying the black-box model.

8.2.3 Second Local explanation

We then examined the worst prediction by the random forest surrogate model.



Feature	Value
pay_stat_aug	1.79
pay_stat_sep	0.90
pay_stat_jul	1.82
pay_stat_may	-1.54
pay_amt_aug	-1.71
pay_stat_jun	-1.53
pay_amt_sep	-1.72
education	1.57
limit_bal	-1.66
bill_amt_sep	-0.70

Figure 19: Lime explanation of the black box(convoluted neural network) model for the observation 217.

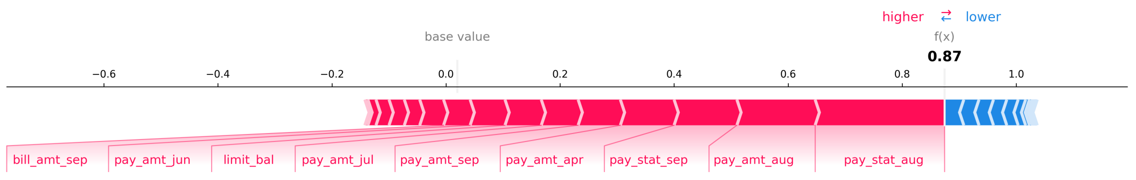


Figure 20: Shap values of the convoluted neural network model for the observation 217.

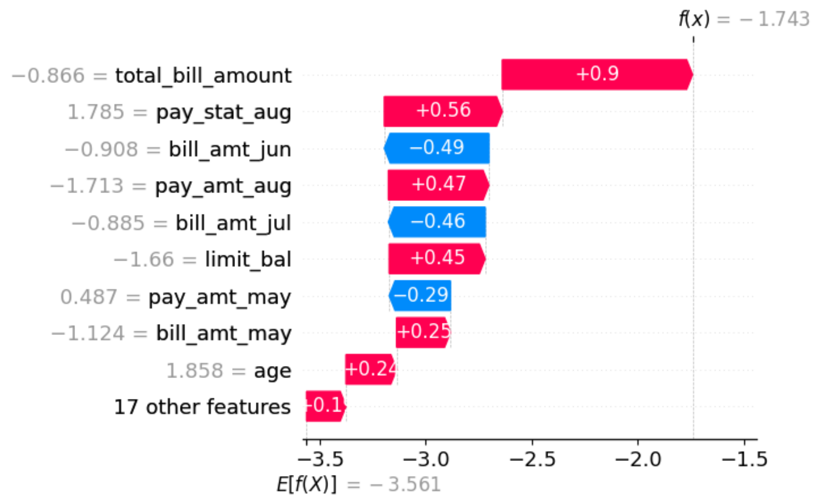


Figure 21: Shap values of the gradient boosting classifier for the observation 217.

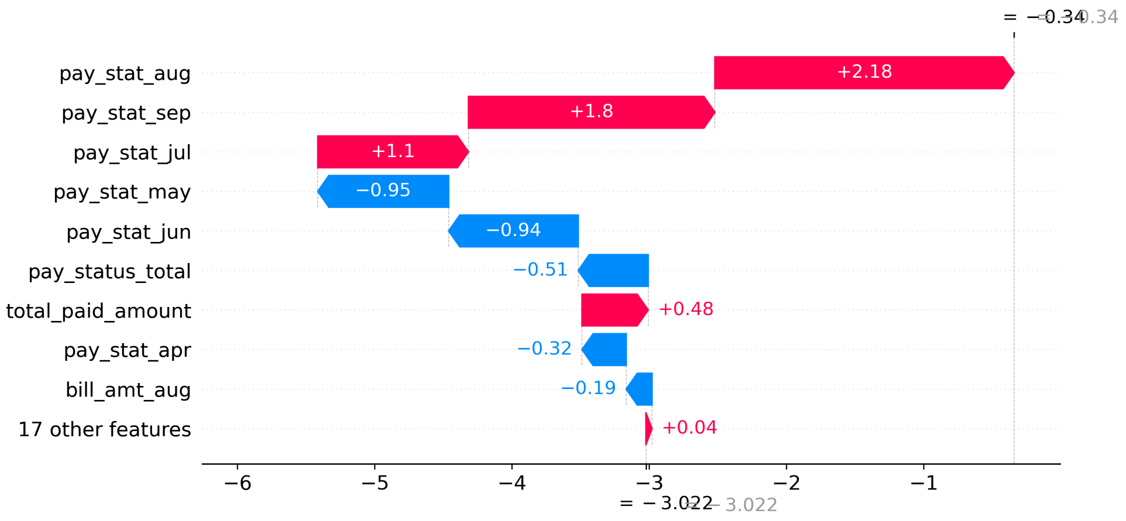
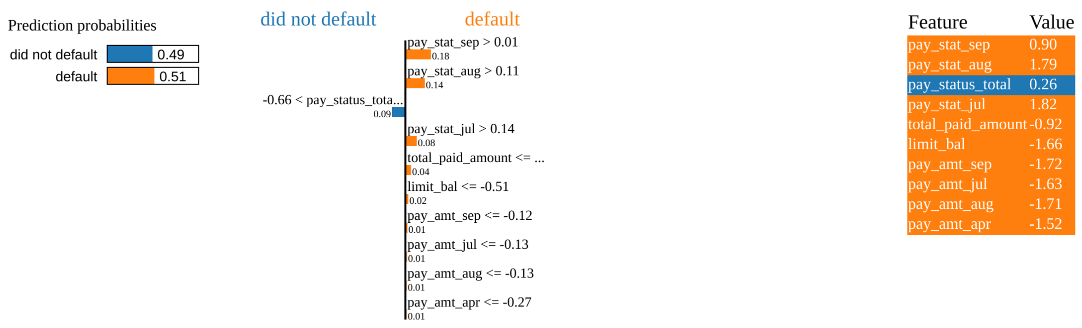


Figure 22: Shap values of the support vector machine classifier for the observation 217.

In 19 the neural network outputs again, similarly to the previous example, a high probability for the default class. In the figures 20, 21 and 22 we can see from the Shapley values that the interpretation is very similar for the three black box models. This was also a false positive.

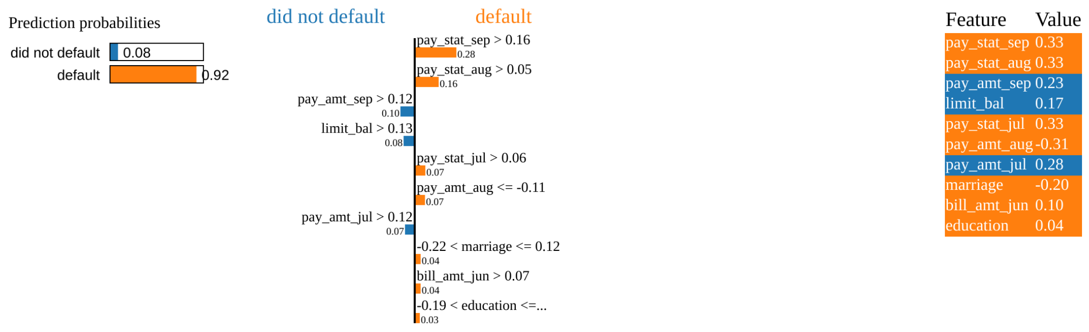


Feature	Value
pay_stat_sep	0.90
pay_stat_aug	1.79
pay_status_total	0.26
pay_stat_jul	1.82
total_paid_amount	-0.92
limit_bal	-1.66
pay_amt_sep	-1.72
pay_amt_jul	-1.63
pay_amt_aug	-1.71
pay_amt_apr	-1.52

Figure 23: Lime explanation of the surrogate model using random forest for the observation 217.

After we compare the random forest surrogate with the black box model in figures 19 and 23, we noticed the misleading factor was yet again the total payment status feature.

8.2.4 Third Local explanation



Feature	Value
pay_stat_sep	0.33
pay_stat_aug	0.33
pay_amt_sep	0.23
limit_bal	0.17
pay_stat_jul	0.33
pay_amt_aug	-0.31
pay_amt_jul	0.28
marriage	-0.20
bill_amt_jun	0.10
education	0.04

Figure 24: Lime explanation of the black box model (convoluted neural network) for the observation 3737.

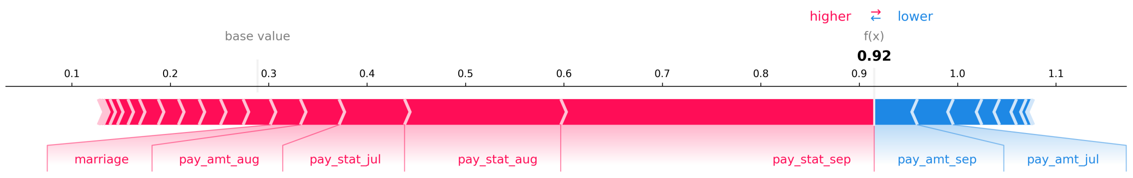


Figure 25: Shap values of the convoluted neural network for the observation 3737.

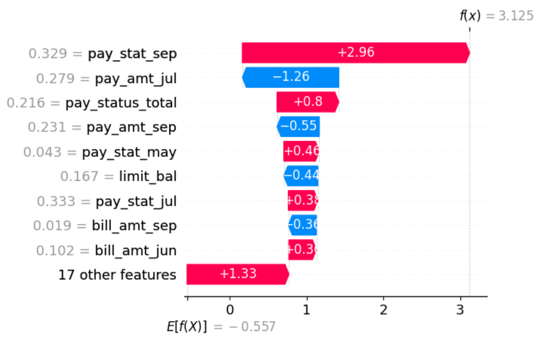


Figure 26: Shap values of the gradient boosting classifier for the observation 3737.

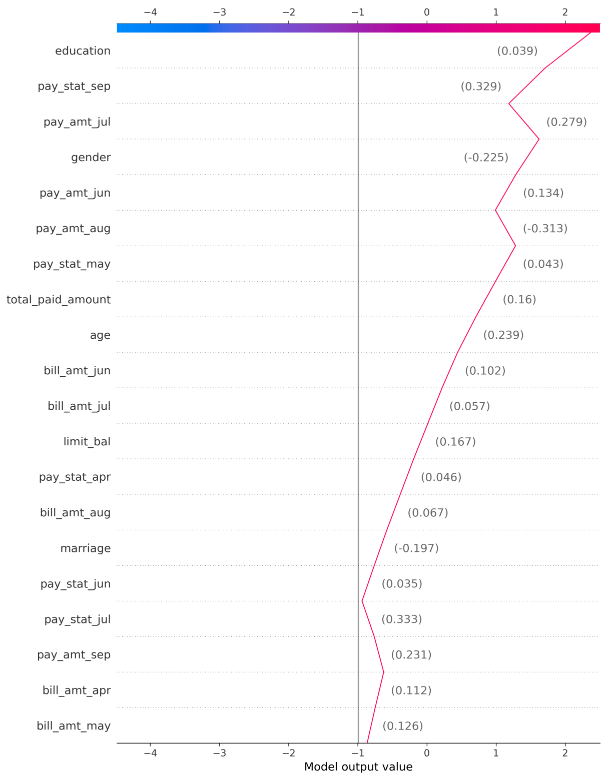
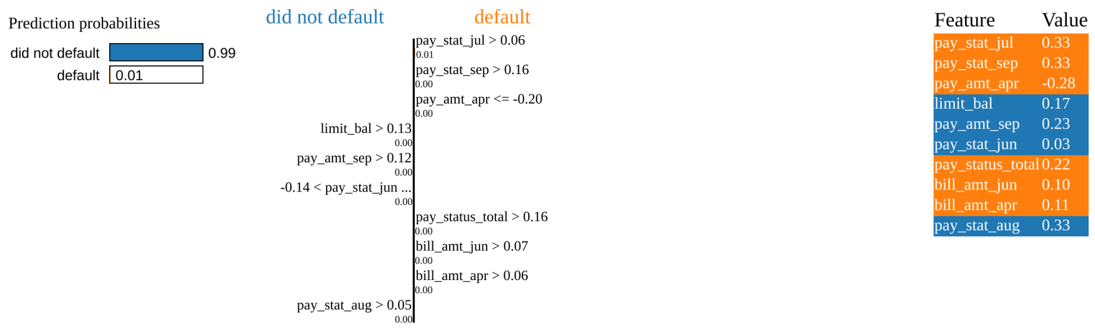


Figure 27: Shap values of the support vector machine classifier for the observation 3737.

In figure 24 we analyzed a final observation. This time we selected a true positive prediction by the black box models 25, 26 and 27 to see if something changed. Coherently with the global explanations the decisive feature was the payment status in September. The only model with a more uncertain output was the support vector machine, which in general seemed to give more relevance to education and gender.



Feature	Value
pay_stat_jul	0.33
pay_stat_sep	0.33
pay_amt_apr	-0.28
limit_bal	0.17
pay_amt_sep	0.23
pay_stat_jun	0.03
pay_status_total	0.22
bill_amt_jun	0.10
bill_amt_apr	0.11
pay_stat_aug	0.33

Figure 28: Lime explanation of the surrogate model using random forest for the observation 3737.

In this case, the random forest surrogate in figure 28 mis-classified due to limit balance and payment status in September. In general we believe that building a surrogate model without the total payment status, which may act as a confounder for some models, might have improved our global surrogate. It's however also possible that the architecture of the convolutional neural network was not interpretable, in particular by the logistic regression.

9. Experimental setup and results

For the baseline and balanced results, we had a high volume of datasets and models, which did not allow us to perform cross-validation, the results are fairly representative however, we also provided a seed for replicability, see our codebase for more details [4].

9.1 Baseline trees

Baseline tree models F1 score

imputation	decision tree	gradient boost.	random forest	xgboost
Drop	0.393	0.498	0.483	0.473
Most frequent	0.451	**0.545**	0.507	0.52
Supervised	0.392	0.466	0.448	0.476
Unsupervised	0.471	0.538	0.508	0.516

The trees were evaluated on the data after handling missing values, without further modifications. They worked as our initial baseline for what we should have expected. From the results above, dropping and supervised imputing with the random forest are dominated strategies, so we removed them from the main pipeline. The gradient boosting was also an early candidate for the best algorithm in this category.

9.2 Baseline neural networks

Baseline most frequent imputation

scaler	convolutional	dense
min-max	0.447	0.483
robust	0.521	0.507
standard	0.485	0.499
norm robust	0.530	**0.535**
norm standard	0.479	0.524

Baseline unsupervised imputation

scaler	convolutional	dense
min-max	0.503	0.457
robust	0.479	0.515
standard	0.499	0.495
norm robust	0.515	0.507
norm standard	0.489	0.511

Based on the run recorded, normalization with a robust scaler is the best scaling option for our networks. To simplify the pipeline we decided to keep this strategy going forwards. Not having a seed, these results are the most volatile, but the general trend is similar to what we provided in table 9.3.

9.3 Baseline other models

Baseline normalized robust scaler F1 score

imputation	knn	logistic	naive-bayes	svm
most frequent	0.483	0.44	**0.517**	0.5
unsupervised	0.484	0.45	0.515	0.5

There is no noticeable difference between the other models, with respect to the imputation method. We could have obtained better results by simplifying the pipeline, but we decided to make the cut as a design choice to limit complexity. We also decided to only keep the most frequent imputation in the main pipeline, since it was always providing the best in class result.

9.4 Balanced trees

We already obtained a respectable F1 score, for the dataset, of **0.572** using a 50:50 balancing strategy with the gradient boosting algorithm, this was the best score we obtained.

9.5 Balanced neural networks

We obtained an F1 score of 0.547 with a 50:50 split, with the convolutional neural network. After tuning the sampling strategy we managed to improve to **0.552**.

9.6 Balanced other models

We obtained the best score with the support vector classifier, with an F1 score of 0.557. After tuning we managed to improve to **0.567**

9.7 Undersampling

After deciding to concentrate on the undersampled dataset, despite the performance hit, we obtained the following $F1$ scores:

1. 0.567 for the gradient boosting classifier
2. 0.533 for the convoluted neural network
3. 0.544 for the support vector machine classifier

9.8 Tuned models

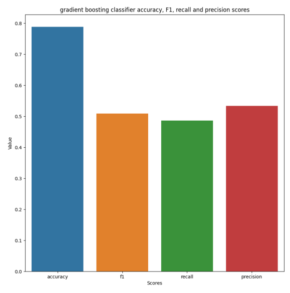
After checking the best combinations for preprocessing the data and the models with the best F1 score performances, we first focused on tuning the gradient booster, the convolutional neural network, and the support vector machine classifier. We then also tuned the simpler models. This was important for the decision tree in particular since we used it as our white-box explainer. For the tuning we used various libraries, including Optuna [14], Hyperopt [15], and Successive halving from scikit-learn [5].

9.9 Final models

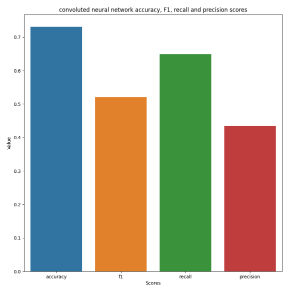
After tuning we evaluted the models on the test set, which we had set aside at the start of the project. For all models we computed accuracy, $F1$ score, precision recall, in figure 29, the confusion matrices, the ROC AUC and the precision-recall curves, as those are the metrics we believe the client would be most interested in.

The final $F1$ scores on the test sets are:

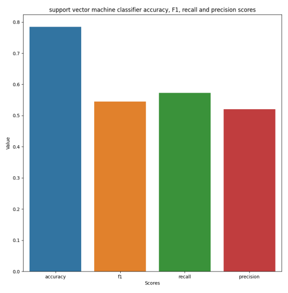
- **0.52** for the gradient boosting classifier.
- **0.52** for the convolutional neural network.
- **0.51** for the support vector machien classifier.



(a) Gradient booster scores

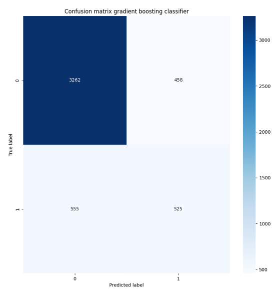


(b) Convolutional network scores

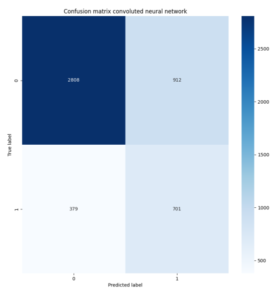


(c) Support vector machine scores

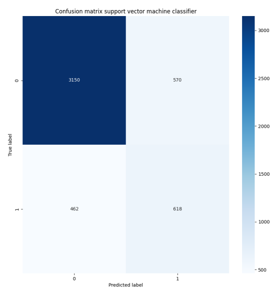
Figure 29: Evaluation on the test set, accuracy, f1, recall, precision



(a) Gradient booster confusion matrix



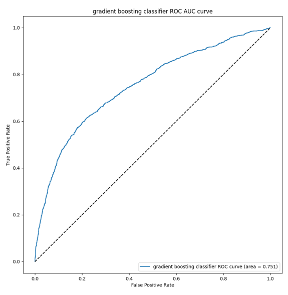
(b) Convolutional confusion matrix



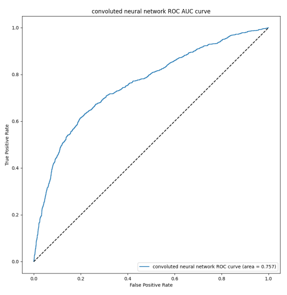
(c) Support vector machine confusion matrix

Figure 30: Confusion matrices on the test set

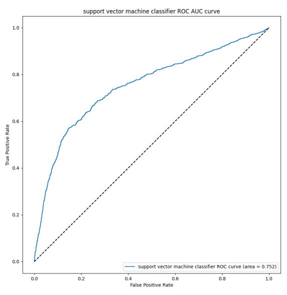
The results are fairly similar for all three classifiers, with an $F1$ score on the test set between 0.5 and 0.54. We likely could have achieved values closer to 0.6 with SVM-smote, but it seemed a good compromise to us. The gradient boosting model has a better precision to recall ratio which might also be a factor.



(a) Gradient booster ROC

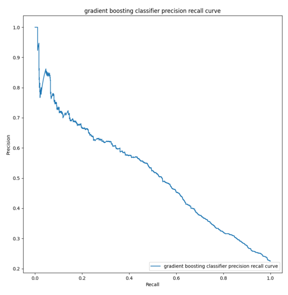


(b) Convolutional network ROC

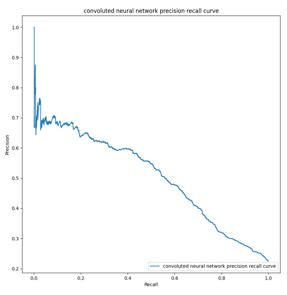


(c) Support vector machine ROC

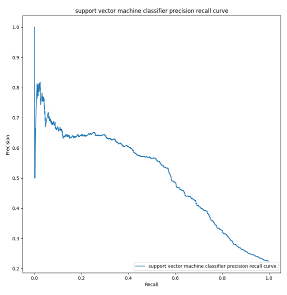
Figure 31: Evaluation on the test set, receiver operator curve



(a) Gradient booster PR curve



(b) Convolutional network PR curve



(c) Support vector machine PR curve

Figure 32: Evaluation on the test set, precision-recall curve

A procedure we would have like to do was thresholding to see if we could improve the $F1$ score, but given the generate precision recall curves it does not seem there was a lot of room to optimize with our design choices.

10. Critical reflections

This project was fairly short and we spent a considerable amount of time creating the pipeline and optimizing preprocessing and balancing strategies. If the primary objective was to get the best performing models, this would have been a good choice, but since a strong focus was on model explainability, and we later had to compromise on performance anyway, we could have allocated better the time. We would have also liked to implement thresh-holding to see if changing the probabilities for defaulting from 0.5 to a higher or a lower value would have improved the overall score. For the explainability part, we would have liked to explore more the relationships between models and implement the partial dependence plots also for the neural network. We learned a lot implementing the methods by hand, like in the case of the white-box model, the global surrogate and the feature permutation. It would have been nice to also create our custom implementations for the other explainable parts. We also tried retraining the global surrogate without the features we added during the feature engineering part of the project and the results are indeed slightly better, but on the other hand it also made for a good topic to explore in the local interpration part. Finally we dedicated little time to feature selection, looking at some of the results it would have been interesting to compare models with different feature portfolios to see if something changes.

11. Conclusions

Thank you for reading our report, we hope you could follow our logic regarding the design choices we made and how they changed aspects of the project. We are reasonably satisfied with the results we obtained during the preprocessing, dataset balancing and tuning phases of the project, we experimented with many different libraries and automated most of the workflow. Regarding interpreting the models, to us it seems more of an art than a science, one has to first know intimately the dataset and what each model is capable of, but also what the pitfalls of using different models and strategies are. Then more than one of the methods we have seen in class will likely lead to a good interpretation.

References

[1] S. Mitrovic, *Data Challenge 3 Slides*, (accessed: 12.12.2022).

[2] T. pandas development team, *Pandas-dev/pandas: Pandas*, version latest, Feb. 2020. DOI: 10.5281/zenodo.3509134. [Online]. Available: https://doi.org/10.5281/zenodo.3509134.

[3] C. W. Stephen John Machin, *Xlrd 2.0.1 documentation.* Version latest. [Online]. Available: https://xlrd.readthedocs.io/en/latest/index.html.

[4] C. P. Davide Gamba Fabio Loddo, *Data challenge 3, project 2*, Dec. 2022. [Online]. Available: https://github.com/ChristianPala/data_challenge_3_project_2.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] I.-K. Yeo and R. A. Johnson, "A new family of power transformations to improve normality or symmetry," *Biometrika*, vol. 87, no. 4, pp. 954–959, 2000, ISSN: 00063444. [Online]. Available: http://www.jstor.org/stable/2673623 (visited on 12/11/2022).

[7] G. E. P. Box and D. R. Cox, "An analysis of transformations," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 26, no. 2, pp. 211–252, 1964, ISSN: 00359246. [Online]. Available: http://www.jstor.org/stable/2984418 (visited on 12/11/2022).

[8] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16, San Francisco, California, USA: ACM, 2016, pp. 785–794, ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785.

[9] Martín Abadi, Ashish Agarwal, Paul Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[10] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-365.

[11] G. Kovács, "Smote-variants: A python implementation of 85 minority oversampling techniques," *Neurocomputing*, vol. 366, pp. 352–354, 2019, (IF-2019=4.07). DOI: 10.1016/j.neucom.2019.06.100.

[12] H. M. Nguyen, E. W. Cooper, and K. Kamei, "Borderline over-sampling for imbalanced data classification," *Int. J. Knowl. Eng. Soft Data Paradigms*, vol. 3, pp. 4–21, 2009.

[13] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, A. Gretton and C. C. Robert, Eds., ser. Proceedings of Machine Learning Research, vol. 51, Cadiz, Spain: PMLR, Sep. 2016, pp. 240–248. [Online]. Available: https://proceedings.mlr.press/v51/jamieson16.html.

[14] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[15] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox, "Hyperopt: A python library for model selection and hyperparameter optimization," *Computational Science Discovery*, vol. 8, no. 1, p. 014 008, 2015. [Online]. Available: http://stacks.iop.org/1749-4699/8/i=1/a=014008.

[16] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., Curran Associates, Inc., 2017, pp. 4765–4774. [Online]. Available: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[17] M. Korobov, *Overview — ELI5 0.11.0 documentation.* [Online]. Available: https://eli5.readthedocs.io/en/latest/overview.html.

[18] C. . Molnar, *8.5 Permutation Feature Importance — Interpretable Machine Learning*, Nov. 2022. [Online]. Available: https://christophm.github.io/interpretable-ml-book/feature-importance.html.

[19] M. T. Ribeiro, S. Singh, and C. Guestrin, “”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, San Francisco, CA, USA, August 13-17, 2016*, 2016, pp. 1135–1144.