# CAS ETH ML in Finance and Insurance
## BLOCK I: Introduction to Machine Learning

Project Credit Analytics: Group Report

Authors: Navarro Diego, Rizzi Michele, Hrovatin Lucia

Spring 2025

# 1 Business Context

Credit risk assessment is a cornerstone of sound banking practice, directly impacting profitability and regulatory compliance. In recent years, the landscape has evolved significantly due to regulatory reforms (e.g., IFRS 9, Basel IV), macroeconomic disruptions (e.g., COVID-19) and technological advances[8]. Taking into account the above, this study addresses the probability of default and investigates whether neural networks can outperform traditional benchmark models such as logistic regression [8].
Using synthetic borrower data, we simulate credit environments to evaluate the predictive performance and robustness of multiple models under different data conditions. If our solution proves better performance, it may be scaled and deployed in production settings, thereby enhancing the precision and reliability of lending decisions in data-sparse contexts.

# 2 Problem Formulation

The objective of this machine learning initiative is to develop a binary classifier that reliably estimates the probability of default for individual borrowers and assigns them to a default class (i.e., default vs. loan paid back). To support the development and validation of the model, we generate a dataset comprising 30 000 borrower profiles and default labels. Synthetic data enables controlled experimentation by simulating a realistic borrower population, while allowing full control over predictors and without requiring access to confidential data.

**Dataset Features and Labels**

Each borrower is profiled based on the following features:

- `age`: uniformly distributed between 18 and 80 years;
- `income`: uniformly distributed between CHF 1 000 and CHF 15 000;
- `employment status`: classified as salaried (90% probability) or self-employed.

Each borrower is assigned a binary label indicating their default status, where `1` denotes default and `0` indicates full loan repayment. The label assignment is based on an individual probability of default, which is modelled using a sigmoid function:

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

To evaluate models performance under varying probability of default distributions, we generate two distinct sets of labels:

1. Linear relationship: The first label set assumes a linear relationship between predictors and probability of default, modelled as:

$$p_1(x) = \psi(-13.3 + 0.33x_1 - 3.5x_2 + 3x_3) \tag{2}$$

2. Nonlinear relationship: The second label set introduces nonlinearity, where the default probability is significantly influenced by age. This results in more complex borrower profiles:

$$p_2(x) = \psi(-5 + 10[1_{(-\infty,25)}(x_1) + 1_{(75,\infty)}(x_1)] - 1.1x_2 + x_3) \tag{3}$$

# 3 Machine Learning Pipeline

## 3.1 Import of Resources

To develop and evaluate any credit risk model, we rely on a set of widely recognised Python libraries. `NumPy` and `Pandas` are used for efficient numerical computations and data manipulation. `Scikit-learn` library provided a comprehensive suite of traditional statistical learning and machine learning algorithms along with essential tools for data pre-processing and model evaluation. For deep learning models, we introduce `Keras` and its sub-library `keras-tuner`. To generate insightful and visually compelling visualizations, we use `Matplotlib` and `Seaborn`.

## 3.2 Data Generation and Exploration

As outlined in Section 2, we generate synthetic borrower data with age, income, and employment status as features, with associated binary labels indicating default status. To gain an initial understanding of the data and explore any impact of the different label generation procedures, we visualize the relationship between `age` and `income` (see Figure fig. 1). `Employment status` is excluded from this plot due to its binary nature. This visual comparison reveals distinct patterns across the two scenarios: while the linear case exhibits relatively clear class separation, the nonlinear case (as defined in Equation 3) presents a more complex pattern with nonlinear decision boundaries. These initial observations suggest the need for flexible modelling techniques capable of capturing such nonlinearity and potentially requiring distinct solutions for the two cases.

## 3.3 Train vs. Test Split

The data is partitioned into training and test subsets, keeping the two scenarios (linear and nonlinear) separate throughout the analysis. While common practice often applies an

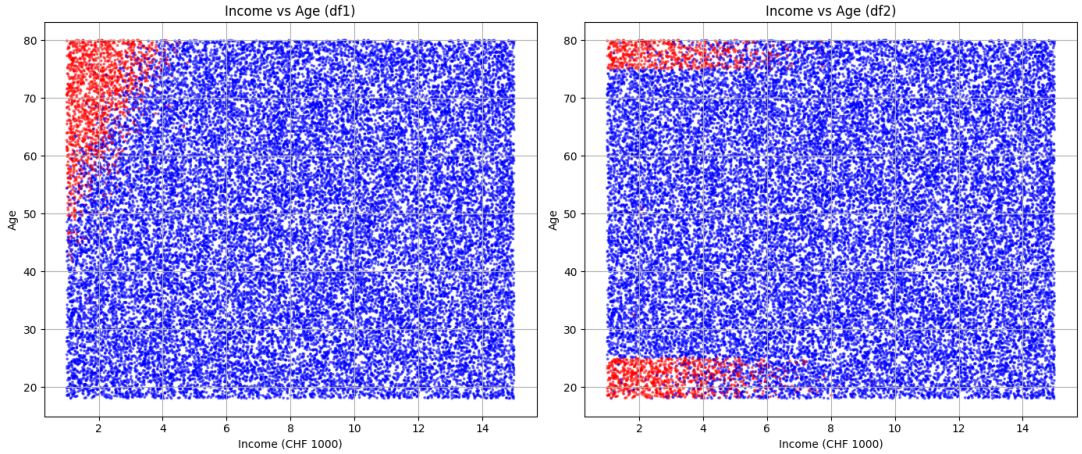


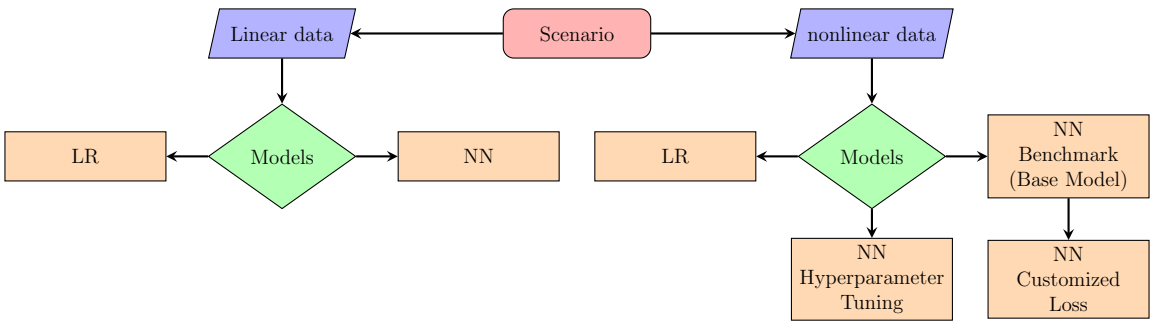

Figure 1: Default probability distributions considering Age vs Income

80/20 or 70/30 train-test split [2], we follow project-specific requirements by assigning $\frac{2}{3}$ of the data to training and the remaining to test. Additionally, we use 20% of the training data as a validation split. This allows for performance monitoring during training without introducing bias into the final model evaluation.

To slightly improve training efficiency and overall model performance, feature normalization is applied after the train-test split to prevent data leakage. Given the uniform distribution of the predictors and the absence of outliers, min-max scaling is the selected technique. This preprocessing step is implemented using the `MinMaxScaler`[1] utility from the scikit-learn library. The method rescales the feature values to the fixed range $[0, 1]$ while preserving the uniformity of the original distribution.

## 3.4 Modelling

For both linear and nonlinear scenarios, we designed a comparative framework to assess the relative performance of two predictive models in estimating default probabilities: *Logistic Regression (LR)* and *Neural Network (NN)*. LR serves as a baseline due to its simplicity, interpretability, and linear decision boundaries [8]. In contrast, NN models do not assume linearity and are expected to capture more complex, nonlinear patterns in the data.




[1] sklearn.preprocessing.MinMaxScaler

### 3.4.1 Logistic Regression

LR is a binary classifier based on the assumption that the default status classes are almost or perfectly linearly separable. Thus, LR models the probability of default and aims to find the best hyperplane to linearly separate the classes. In our implementation, we use the `sklearn` built-in function:

```
from sklearn.linear_model import LogisticRegression
LR_model = LogisticRegression(penalty=None).fit(X, Y)
```

The small number of predictors ($p = 3$) allows the model to effectively generalise on unseen data, without the need for regularization (i.e., `penalty=None`). To validate this assumption, we compare an LR model with no penalties and a Ridge Regression with $\lambda = 0.1$. We choose a Ridge regression over a Lasso regression, as the former is supposed to perform better when the model response is a function of predictors with coefficients of roughly the same size [6]. Both models performed satisfactorily on the test set, with nearly identical accuracy. Therefore, we chose standard logistic regression over the slightly more complex Ridge regression.

| | **LR no penalties** | **Ridge ($\lambda = 0.1$)** |
|---|---|---|
| **Linear** | 0.985 | 0.967 |
| **Nonlinear** | 0.954 | 0.954 |

Table 1: Comparison of test set accuracy measure

### 3.4.2 Neural Network models

**Linear Architecture** To establish a baseline model capable of generalising on linearly separable data, we implement a straightforward neural network architecture as outlined in the project description. The first model consists of two fully connected hidden layers, each comprising 20 units, with `ReLU` activation function. Given the binary classification task, we add a sigmoid activation function in the output layer and train the network by minimising the cross-entropy loss function for 100 epochs using a batch size of 1024. Following the literature, `Adam` algorithm is the default choice for many neural network implementations and the best among adaptive optimizers in most cases [11]. Therefore, our network embeds `Adam` optimizer with an initial learning rate of 0.01, which is a default value and typically works for standard multi-layer neural networks [1].

**Nonlinear Architecture(s)** To address the increased complexity and nonlinearity, our assessment starts with a baseline model, named *NN Benchmark*, that mirrors the linear NN architecture described above, but increases the number of units to 50 on each of the two hidden layers. However, aiming to improve performance, we explore several combinations of parameters by running `keras_tuner` with cross-validation ($k = 5$). Specifically, the tuner performs a random search over several hyperparameters that influence the performance of the architecture [3]:

- **Learning Rate** Considering the key role of this hyperparameter [1], we tune it setting a logarithmic scale $\eta = [10^{-3}, 10^{-2}, 10^{-1}]$.
- **Batch Size** While using a batch size of $B = 1024$ significantly reduces training time, we opt for a more conservative approach by exploring smaller batch sizes,

starting from the widely accepted default of $B = 32$ [1]. Specifically, we focus on a geometric progression with batch sizes $B = [32, 64, 128]$, allowing the hyperparameter tuner to select the most appropriate value.

- **Model depth and size** Building on the heuristic proposed by Goodfellow et al. [3], which suggests that increasing network depth often improves generalisation across a wide range of tasks, we explore the impact of multiple hidden layers with different sizes. Whereas the number of layers ranges from 2 to 5, the hidden units vary from 32 to 64 neurons.

- **Optimizer** Although `Adam` is widely used as the default optimizer in many neural network implementations [11], we employ `RMSprop` as an alternative. `RMSprop` offers a reduced number of hyperparameters, lower computational overhead, and is well suited for mini-batch learning scenarios [7]. These characteristics make it an appropriate choice given the batch-size configurations evaluated in our experiments.

- **Inner activation function** We investigate alternatives to `ReLU` activation function, motivated by the well-documented *dying ReLU* problem. As discussed in Xu et al. [12], out of several `ReLU` variants empirically evaluated, leaky versions consistently outperform standard `ReLU`. Based on these insights, we incorporate `LeakyReLU` as an alternative activation function. We also adjust the leak coefficient, setting it to $\alpha = 0.2$, since larger leak values should yield better performance than smaller values [12].

In addition to hyperparameter optimization, *Batch Normalization* is incorporated to mitigate vanishing and exploding gradient risks [5]. This technique is applied before the activation function of each hidden layer, resulting in reduced sensitivity to weight initialization, accelerated convergence during training, and a decreased dependence on other regularization methods such as dropout [5]. To complement these changes, an appropriate weight initialization strategy is chosen. While `Keras` uses `Glorot` initialization as the default kernel initializer, we select the `He normal` initialization strategy, which better fits `ReLU` activation function and its leaky variants [4, 2].

To identify the optimal model, understood as the most effective binary classifier, we evaluate performance not solely based on *accuracy*, but also on *sensitivity* (or *recall*). This focus is motivated by the objective of minimizing false negatives, where individuals who are likely to default are incorrectly classified as non-defaulters. Additionally, early stopping is implemented in the training phase to reduce the risk of overfitting [3].

The resulting model, referenced below as *NN tuned*, has 2 hidden layers with `ReLU` activation function, `Adam` optimizer, an initial learning rate of 0.001, and batch size of 64. As *NN Benchmark* the output layer has a sigmoid function to accommodate the binary classification task.

To further enhance the optimization process, our objective is to incorporate the inherent asymmetry of credit decision-making. Specifically, the impact of a false negative (i.e., granting credit to an individual with a high probability of default) is 100 times higher than the consequence of a false positive, which implies denying credit to a low-risk applicant. Also, the additional reward of a true negative is introduced in the loss function. To reflect this condition, we introduce a *penalised* version of the cross-entropy loss function:

$$L(y_i, \hat{y}_i) = -100 \cdot y_i \log(\hat{y}_i) - (1 - y_i)(\log(1 - \hat{y}_i) + 1) \tag{4}$$

The parameters and architecture identified in the *NN benchmark* are used to train *NN customized loss*, a final version of the binary classified with ur custom loss function (4).

| | **NN Benchmark** | **NN tuned** | **NN customized loss** |
|---|---|---|---|
| **Number of hidden layers** | 2 | 2 | 2 |
| **Number of neurons** | 50-50 | 54-42 | 50-50 |
| **Inner activation function** | ReLU | ReLU | ReLU |
| **Learning Rate** | 0.01 | 0.001 | 0.01 |
| **Optimizer** | Adam | Adam | Adam |
| **Batch Size** | 1024 | 64 | 1024 |
| **Loss** | Cross-entropy | Cross-entropy | Customized Loss |

Table 2: Comparison of Neural Network models

## 3.5 Performance and Interpretability

Choosing the appropriate performance metric for classification tasks in credit decision-making is crucial to minimize monetary risk. Thus, we focus our attention not only on the accuracy of the models but also on their misclassification rate. Specifically, we evaluate ROC and AUC scores along with confusion matrices to assess the capability of the models in distinguishing between defaulting and non-defaulting borrowers.

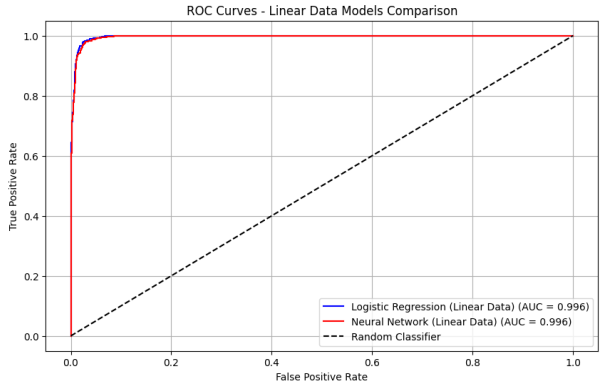



Figure 2: ROC curve - linear case

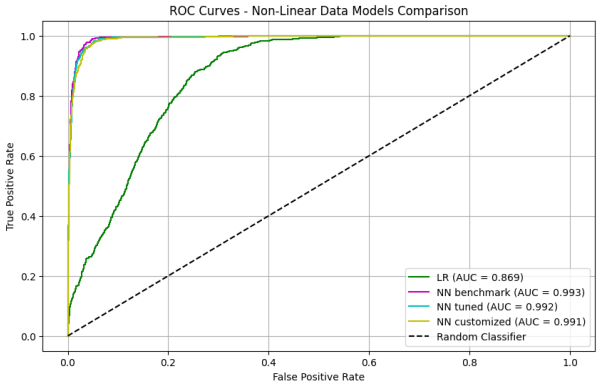



Figure 3: ROC curve - nonlinear case

In the linear case, the ROC curve and the corresponding AUC score show how the LR model performs as well as a neural network (see Figure 2). This result reinforces the value of logistic regression as a benchmark model in credit risk assessment, particularly when the underlying relationships are approximately linear [8]. Moreover, its inherent transparency enhances model interpretability: each feature is associated with a coefficient, enabling business stakeholders to clearly understand how individual inputs influence the

probability of default. This level of interpretability not only supports regulatory standards but also strengthens trust in model-driven decisions. Note that data normalization affects the interpretability of the coefficients by altering their scale.

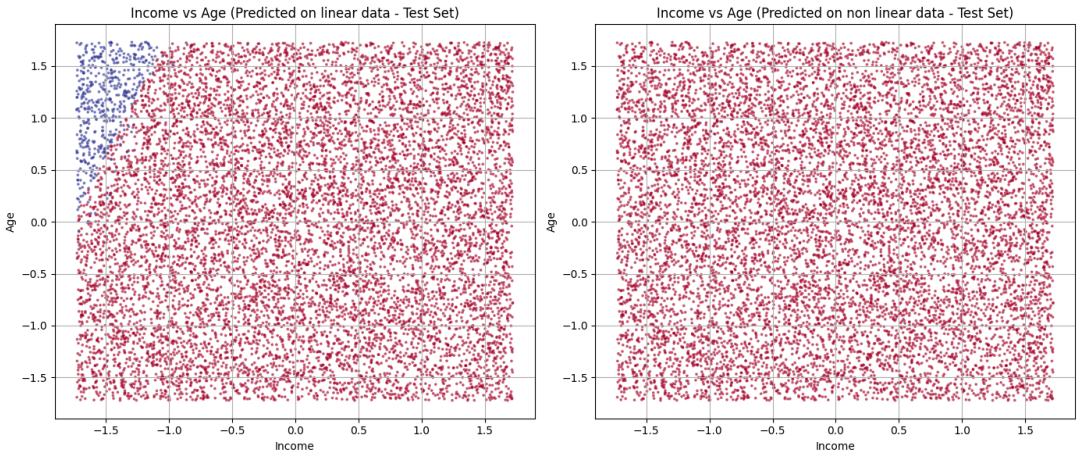



Figure 4: Linear vs Nonlinear scenario (LR)

LR model appears to achieve acceptable accuracy also in the second scenario; however, a more detailed examination of the confusion matrix (see Figure 5) reveals a critical limitation: notably low recall. Thus, actual default cases are misclassified as non-defaults and the model does not identify any borrower as being at risk of default (Figure 4).

In contrast, neural network models demonstrate better predictive accuracy and significantly reduce the number of defaulting borrowers erroneously classified as non-defaulters (Figure 5). This improvement is particularly evident in the last two confusion matrices, which underscore the enhanced classification capability of the optimized models. During the hyperparameter tuning, the models' performance is evaluated primarily on recall (and accuracy), given its importance in minimising missed defaults. Furthermore, replacing the standard `cross-entropy` loss function with a customised version allowed the model to converge more efficiently by explicitly penalising a certain type of misclassification.

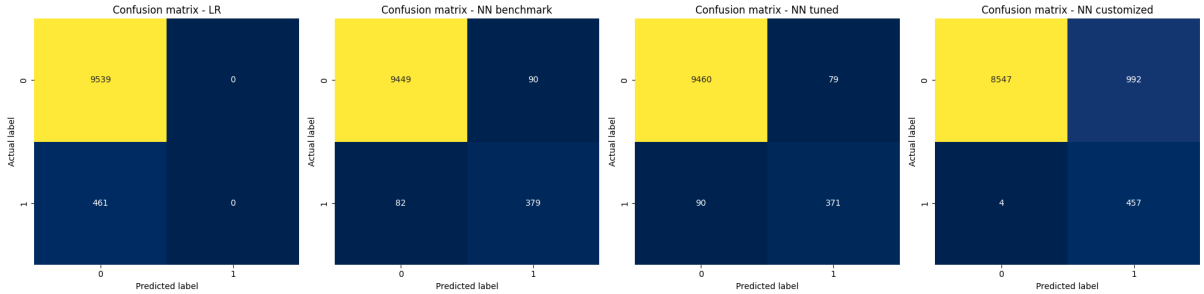



Figure 5: Confusion Matrices for nonlinear models (classification threshold of 0.5)

It should be noted that the customized model exhibits a strong tendency to classify true positives with high confidence (i.e. assigning probabilities close to 1.0). However, it has difficulty assigning low probabilities (near 0.0) to true negatives. This behavior results in a mixed performance profile: a low number of false negatives but a high number of false positives, as illustrated in the confusion matrices in Figure 5 and the probability distribution in Figure 6.

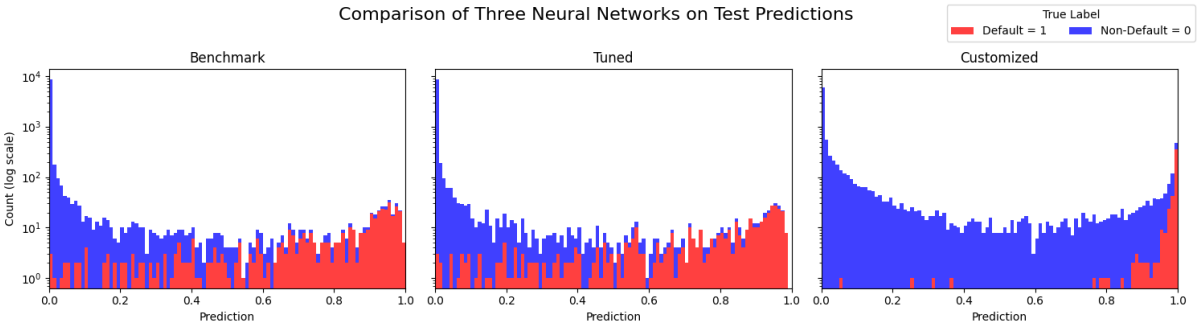



Figure 6: Neural Networks Default Probability Distributions

Regardless of the chosen neural network model, improved predictive performance comes at the cost of interpretability and increased complexity. Thus, architecture and parameters are explicitly defined and measurable, but the process behind individual predictions remains largely opaque. This lack of transparency presents a critical limitation, especially in highly regulated sectors such as finance, where explainability is essential to comply with regulatory requirements. Although recent research has introduced interpretability techniques [9], the challenge of providing robust and actionable explanations remains unresolved.

# 4 Business Application of the ML Models

## 4.1 Lending Strategy Simulation

To explore the performance of our models, we implement three lending strategies focusing on the nonlinear scenario and simulating 1000 scenarios, with 10000 borrowers each. The first strategy, based on randomisation, consists of selecting all customers while applying a higher interest rate to mitigate potential losses (5.5%). Although the high interest rate helps contain defaults, it yields suboptimal results compared to more targeted strategies ($VaR_{All} = 0.257$). The second lending strategy leverages Logistic Regression by allowing loans only if the predicted probability of default is $\hat{p}_{LR}(x) \leq 0.05$. However, as mentioned in Section 3.5, the LR model struggles due to its poor predictive power in capturing nonlinearity. As a result, its accuracy is comparable to the first lending strategy, but its lower interest rate (1%) leads to a worse Value at Risk ($VaR_{LR} = -0.006$). The third strategy adopts the same lending criterion as the second ($\hat{p}_{NN}(x) \leq 0.05$), but leverages neural networks to evaluate nonperforming customers. NN models deliver the best performance by accurately identifying high-risk borrowers, enhancing decision quality, and achieving a Value at Risk (VaR) significantly higher than previous strategies. Both the Benchmark and Tuned NN models outperform alternative approaches with $VaR_{NN} \approx 0.74$. At the arbitrary 5% threshold, the Customized model records a lower average profit per borrower than the Benchmark and Tuned models. However, it demonstrates reduced variability across all simulations (see Figure 7).

## 4.2 Threshold Optimization

Finally, to optimize the threshold $\gamma$, we focus on maximising profitability and minimising the risk of default while keeping the loan amount stable (i.e., CHF 100). This is done

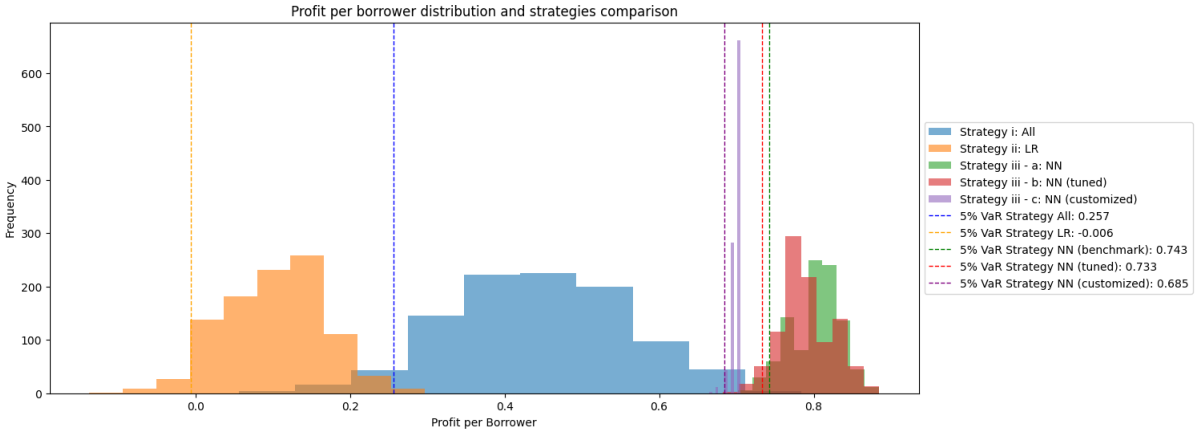



Figure 7: Lending Strategies simulation

by simulating net profit using the previous lending strategies but ranging over different threshold values within the interval $[0.01, 0.99]$. We focus our analysis on the three neural network models (Benchmark, Tuned, Customized) and all of them show good performance on their respective optimized thresholds (see Figure 8). A key observation is that the Customized model is less sensitive to threshold selection, maintaining strong performance across a wide range of values. In contrast, the Benchmark and Tuned models exhibit greater sensitivity to the chosen threshold.

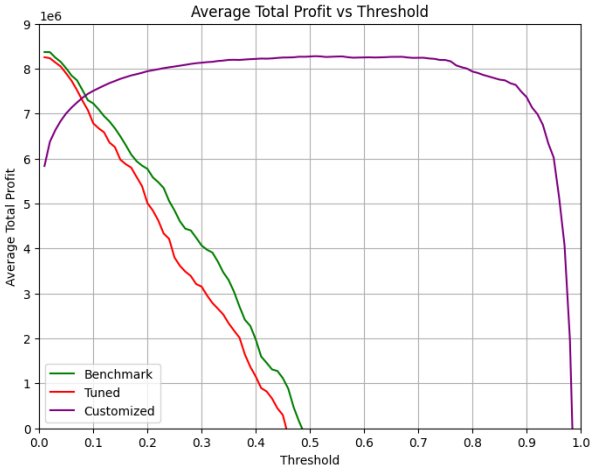



Figure 8: Threshold value optimization

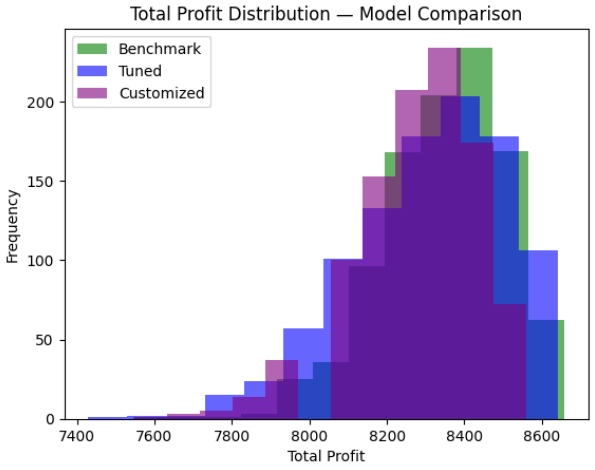



Figure 9: Total Profit Scenario Simulation

Finally, using the optimal threshold for each model, we rerun 1̇000 scenarios to estimate the average total profitability and 5% VaR, as illustrated in Figure 9. The three models demonstrate very similar results in terms of expected profit and variability between scenarios, as detailed in Table 3. Given the comparable profit-and-loss performance, we recommend implementing the Customized model due to its superior robustness across different threshold values, which is critical for reliable decision-making in a banking environment.

| NN model | 5% Var | Avg. PnL | Std Dev | Threshold |
|---|---|---|---|---|
| **Benchmark** | 8052 | 8371 | 173 | 0.01 |
| **Tuned** | 7934 | 8256 | 203 | 0.01 |
| **Customized** | 7953 | 8279 | 173 | 0.51 |

Table 3: Total PnL Simulation Results

# 5 Limitations and Future Work

## 5.1 Potential Modeling Improvements

In this project, we present models trained on a limited synthetic dataset. The predictive power of these models could be improved by applying them to real-world data or by incorporating additional synthetic features such as debt-to-income ratio, loan-to-value ratio, and household size. These features would provide a more comprehensive view of borrowers' financial stability and risk profiles, aligning with the traditional five Cs of credit analysis: capacity, character, capital, collateral, and conditions [10]. A more complete dataset would also enable further exploration of ensemble methods and more advanced neural network architectures, which have consistently demonstrated superior performance in the literature [8]. However, employing more advanced methods requires enhanced model transparency and interpretability. Therefore, we propose adopting explainable AI techniques such as SHAP (SHapley Additive exPlanations), which provide detailed insights into the contribution of each feature to individual predictions. This approach not only fosters trust in model outputs but also satisfies regulatory requirements for transparency in credit risk assessment.

## 5.2 Improving Business Use

To improve the effectiveness and resilience of our credit risk model, there is an opportunity to jointly optimize interest rates alongside decision thresholds. This approach would serve two primary objectives: (a) aligning pricing more closely with customer risk profiles, and (b) enhancing loan sales through more targeted and competitive interest rates. Accurate pricing maximises profitability by increasing profitability per customer, reducing default rates, and growing overall lending volumes. Implementing this would require a robust understanding and modeling of price sensitivity through additional input features and outcome labels. The model should ideally provide three outputs: the probability of default, the recommended pricing, and the probability of loan acceptance.

In parallel, it would be beneficial to embed explicit profitability criteria into the modeling process. This could include factors such as the cost-to-serve between customer segments, projected losses under different market conditions, and minimum acceptable profitability thresholds. Assuming a broader view ensures that the model does not focus solely on risk, but also supports the profitability targets, factoring in both expected returns and risk mitigation strategies. This dual focus on risk and profitability will enable more strategic decision-making, ultimately supporting sustainable portfolio growth and long-term profitability.

*Note*: The project code is available at the following link.

# References


[1] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *Neural networks: Tricks of the trade*, pages 437–478, 2012.

[2] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 2nd edition, 2019.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[6] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, et al. *An introduction to statistical learning*, volume 112. Springer, 2013.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[8] Anton Markov, Zinaida Seleznyova, and Victor Lapshin. Credit scoring methods: Latest trends and points to consider. *The Journal of Finance and Data Science*, 8:180–201, 2022.

[9] Marco Tulio Ribeiro, Sameer Singh, and Carlos 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*, 2016.

[10] Peter S Rose and Sylvia C Hudgins. Bank management & financial. *Services Published by Mc Graw–Hill/Irwia, USA*, 2005.

[11] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[12] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.