Machine Learning Project

Credit Risk Assessment with Classification Models

Edoardo Olivieri[1], **Federica Romano**[1] **and Francesca Verna**[1]

[1] *Università degli Studi di Milano-Bicocca, CdLM Data Science*

Abstract

The objective of this study is to develop a predictive model for assessing the loan risk of applicants across various cities in India, based on diverse socio-economic factors. By leveraging these data points, the research aims to support financial institutions in making informed loan approval decisions, thereby addressing the critical challenge of credit risk assessment. Effective risk evaluation is essential for reducing defaults and enhancing the efficiency of lending practices. We conducted a comprehensive data exploration, applied different pre-processing techniques, and dealt with the problem of imbalanced classes, offering a possible solution. We also built different classification models in order to identify which one is the most suitable for addressing the goal of our research.

Keywords: *Loan, Prediction, Risk, Imbalanced Classes, Machine Learning*

1. Introduction

Risk management is a cornerstone of the financial industry, encompassing the systematic processes of identifying, assessing, and mitigating potential financial losses that stem from uncertainty. In lending, it focuses on evaluating the likelihood of loan defaults and implementing strategies to minimize associated risks.

The assessment of a risky loan applicant involves multiple factors. Initial indicators may include unstable income, limited professional experience, or lack of home ownership. A deeper evaluation often incorporates contextual elements such as marital status, employment stability, and geographic location. Although these factors provide valuable information, risk assessment remains inherently complex and subjective, as attributes such as high income do not always guarantee low risk if other variables, such as duration of employment, are unfavorable. Furthermore, objective and measurable characteristics, such as years in a current job or residence, offer structured analytical frameworks but may not fully capture the multidimensional nature of risk.

Despite these challenges, it is vital for lending institutions to explore whether a combination of quantitative and categorical features can accurately predict the risk of a loan applicant.

1.1. Overview of the Dataset

In order to answer this research question, we used the *Loan Approval Dataset*, available on the Kaggle platform [7]. This dataset, consisting of 252,000 entries and 13 features, was analyzed using KNIME, a data analytics platform [2].

- **Id**: unique identifier for each loan applicant;
- **Income**: income of the applicant.
- **Age**: age of the applicant.
- **Profession**: occupation or profession of the applicant.
- **City**: city of residence of the applicant.
- **State**: state of residence of the applicant.
- **Experience**: years of professional experience;
- **Married/Single**: categorical variable (*married*/*single*) that states the marital status of the applicant;
- **House_Ownership**: categorical variable that indicates whether the applicant owns or rents a house. It can take three values: *rented*, *owned*, *norent_noown*;
- **Car_Ownership**: categorical variable (*yes*/*no*) that represents whether the applicant owns a car;
- **Current_Job_Yrs**: how many years the applicant has been working in their current profession;
- **Current_House_Yrs**: how many years the applicant has been living in their current home;
- **Risk_Flag**: binary variable that indicates the loan risk, where 1 represents a risky applicant and 0 represents a non-risky applicant. It is our target variable.

The objective of the analysis is to accurately predict, using appropriate machine learning techniques, whether a new loan applicant should be considered risky or not.

2. Exploratory Data Analysis

During the initial stages of the exploratory analysis, we excluded **Id** from consideration, as it serves solely as an identifier and does not provide a meaningful explanatory value for the analysis.

We later observed that, after removing the **Id**, approximately 82.86% of the rows in the dataset are duplicates.

Despite this, we chose to proceed with both the exploratory and subsequent analyses using the original dataset, including the "duplicates". We made this decision because the **Id** uniquely identifies each applicant, and since the dataset contains no duplicate Ids, it ensures that all entries represent distinct individuals.

2.1. Class Imbalance Problem

As an initial step in the exploratory data analysis, we focused on the target variable **Risk_Flag** in order to evaluate the number of observations belonging to categories 0 and 1 and to check for any imbalance problem. We were able to observe that the classes are significantly imbalanced, with a strong predominance of class 0, representing non-risky candidates. Notably, approximately 87.70% of the candidates are considered non-risky, while only 12.30% are considered risky. This suggests the need to apply specific balancing techniques before implementing any machine learning model.

2.2. Variables Distributions and Analysis

To gain deeper insights into the dataset, we conducted a detailed analysis of both categorical and numerical variables.

2.2.1. Categorical Variables

For the categorical variables **Married/Single**, **Car_Ownership** and **House_Ownership** with the use of two types of bar charts, one illustrating the number of observations per category and another showing the average value of **Risk_Flag** for each category, we gathered information about their relationship with the target variable.

In terms of marital status, the majority of observations fall into the *single* category, with 226,272 individuals compared to 25,728 in the *married* group. Single applicants exhibit a slightly higher average **Risk_Flag** value of 0.13, compared to 0.10 for married individuals, indicating a marginally greater risk associated with single applicants.

Regarding **Car_Ownership**, applicants without a car outnumber those with one by a significant margin of 100,000. Non-car owners have an average **Risk_Flag** of 0.13, while car owners show a slightly lower risk profile, with an average of 0.11. This suggests that individuals without vehicles are modestly more likely to present a higher risk.

House_Ownership further reflects notable differences. Among the observations, 231,898 individuals live in rented properties, 12,918 own their homes, and 7,184 fall into the *norent_noown* category. The corresponding average **Risk_Flag** values are 0.13 for renters, 0.09 for homeowners, and 0.10 for those without rental or ownership status. These findings suggest that renting is associated with a slightly elevated risk, while home ownership correlates with a lower-risk profile.

Additionally, the dataset includes 51 unique professions. *Physicians* represent the largest group, with 5,957 observations, while the least common profession, *Engineer*, has 4,048 occurrences. The **Profession** distribution is relatively uniform, with no significant concentration in any particular location.

For what concerns the **City** and **State** variables, we first needed to address some encoding and formatting issues present in the original dataset. In fact, we observed that some city names in the **City** variable and that some state names in the **State** variable contained numbers enclosed in square brackets at the end of the string. We decided to conduct an investigation to determine whether this notation was used to differentiate cities or states with the same name.

For **State**, this issue did not occur, so we simply removed the numbers and brackets to ensure that the machine learning models recognized them as the same category.

For **City**, however, additional checks were necessary. We found that *Antanpuram* and *Antanpuram[24]* refer to the same city, and so we standardized them as *Antanpuram*. Further investigation revealed that there is no distinction between *Anantapur* and *Anantapuram*, as they are the same city in Andhra Pradesh, India, with *Anantapuram* being the traditional, longer name, and *Anantapur* the more common abbreviation. For this reason, we also standardized *Antanpur* as *Antanpuram*.

A different case involved *Aurangabad* and *Aurangabad[39]*, which are distinct cities sharing the same name. Aurangabad in Maharashtra is known for UNESCO sites like the Ajanta and Ellora Caves, while Aurangabad in Bihar serves as an agricultural and administrative hub.

Finally, we recoded the name *Hubliâ€"Dharwad* to *Hubli_Dharwad* to correct a likely transcription error. For the remaining cities, we followed the same procedure applied to **State** which involved the removal of the numbers. Once we made the changes, we looked at the diversity of these last two variables.

The data spans 316 unique cities, with *Anantapuram* being the most frequent, appearing 1,573 times, while the least frequent city, *Kairaikudi*, appears 431 times. Like the **Profession** distribution, the **City** variable is relatively balanced without major skewness.

State representation, on the other hand, shows greater variability. Among 28 unique states, *Uttar_Pradesh* is the most prevalent, with 29,143 observations, whereas the least represented state, *Sikkim*, accounts for just 608 observations.

2.2.2. Numerical Variables

For the numerical variables, using boxplots conditioned on the two categories of **Risk_Flag**, we observed only minor variations between the distributions of the different categories. These subtle differences suggest a limited ability to differentiate between risky and non-risky applicants based solely on the numerical features.

To further investigate, we also used a *ggpairs* plot from the *GGally* library in R [11], which displays density plots along the diagonal for each variable, split by **Risk_Flag** categories. The density curves are nearly identical across all numerical variables, further confirming the lack of significant differentiation.

Additionally, the *ggpairs* plot highlights the correlation between variables. Most variables exhibit correlation values close to 0, indicating little to no linear relationship. However, a notable exception is observed between **Current_Job_Yrs** and **Experience**, which show a positive correlation of 0.646. This suggests that applicants with more years of experience tend to have longer job tenures. Despite this correlation, we decided to retain both variables. The correlation value was not deemed excessively high [3], as in financial risk modeling, both long-term stability and recent stability could play crucial roles. Removing either variable could result in the loss of valuable insights into applicant behaviour and stability.

3. Pre-Processing

First, we divided the dataset into training and test sets, allocating 80% of the observations to the training set and 20% to the test set. We also applied stratified sampling with respect to the target variable to ensure that the proportions of **Risk_Flag** were preserved in each sample.

3.1. Categorical Variables Transformations

From the exploratory analysis, it emerged that the target variable **Risk_Flag** is able to discriminate, even if slightly, among the different categories of the variables **Married/Single**, **Car_Ownership**, and **House_Ownership**. For this reason, we encoded these variables differently, in both the training and test sets. Specifically, we assigned to each category an incremental value, starting from 0, based on the mean of the **Risk_Flag** for that category.

In particular, for **Married/Single**, we encoded the category *married* as 0 and *single* as 1. Similarly, for **Car_Ownership**, *no* was assigned a value of 1, and *yes* a value of 0. Lastly, for **House_Ownership**, we encoded *owned* as 0, *norent_noown* as 1, and *rented* as 2.

3.2. Clustering

Given that the categorical variables **City**, **State**, and **Profession** have high cardinality, we decided to group them using the k-means algorithm [9]. We tested the algorithm with a varying number of clusters, ranging from 3 to 9. We also assessed the quality of the clustering structures using three different measures: the Silhouette Coefficient, the Dunn Index, computed using the R library *clValid* [5], and the Davies-Bouldin Index, computed using the Python library *scikit-learn* [10]. These measures are essential to determine the optimal number of clusters to use.
Three different line plots are shown, one for each metric.

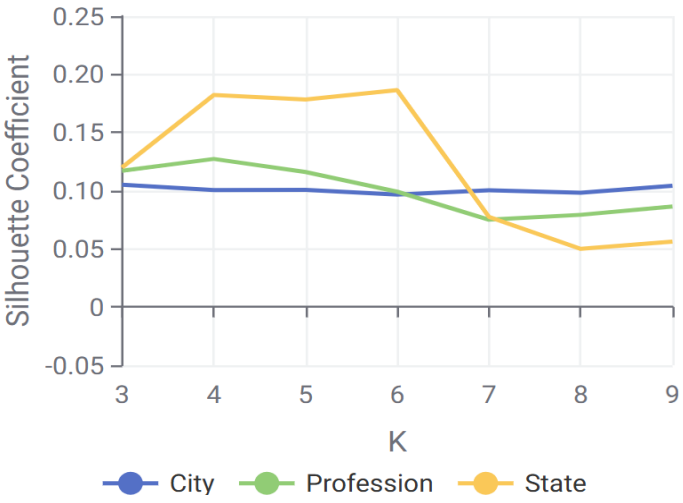

Figure 1. Silhouette Coefficient

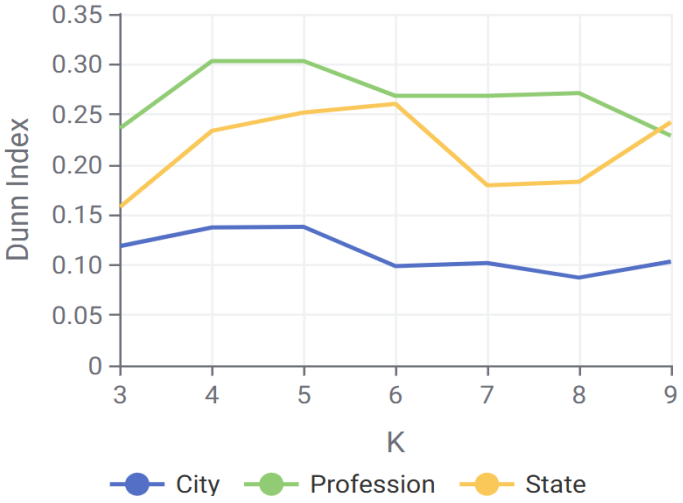

Figure 2. Dunn Index

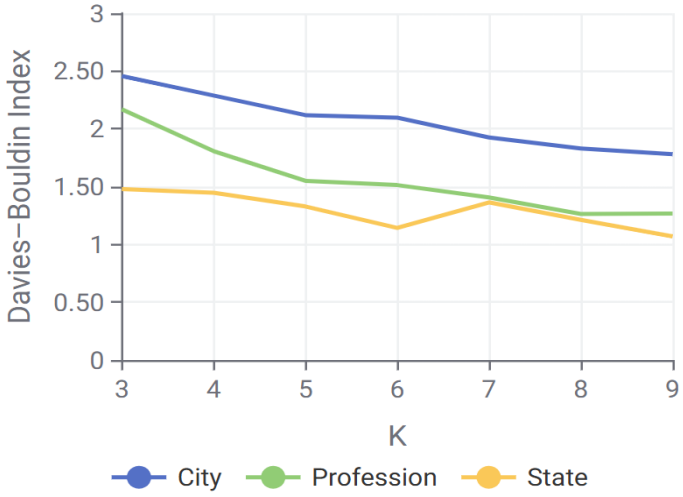

Figure 3. Davies-Bouldin Index

For clustering evaluation, the optimal number of clusters depends on the behavior of the chosen metrics. The Silhouette coefficient and the Dunn index identify the optimal number of clusters as the one that maximizes their respective values, indicating better cohesion and separation of clusters. Conversely, the Davies-Bouldin index defines the optimal number of clusters as the one that minimizes its value, as lower scores reflect well-defined and compact clusters with greater separation.
The following table shows the optimal number of clusters for each variable, considering the different metrics.

Variable	Silhouette	Dunn	Davies-Bouldin
City	3	5	9
Profession	4	5	8
State	6	6	9

Table 1. Optimal number of clusters for each metric

The results of the Cluster Analysis indicate that clustering based on the variable **State** generally performs better than clustering on the other two variables. This is particularly evident when considering the Silhouette Coefficient (Figure 1). For most values of K, the Silhouette Coefficient for **State** outperforms those of **City** and **Profession**, reflecting better-defined clusters with higher cohesion and separation.

These findings suggest that clustering based on **State** provides the most meaningful and interpretable grouping structure. We also tried to conduct the same classification models without applying clustering to **State** and the results obtained were worse, especially regarding the XGBoost model. For these reasons, we chose to perform clustering only on the variable **State**.

Regarding the number of clusters for **State**, as seen in Table 1, the Silhouette Coefficient and the Dunn Index identified K=6 as the optimal number of clusters , with Davies-Bouldin ranking it as the second-best choice, as shown by Figure 3. This consistency across metrics supports our selection of K=6 as a robust solution.

The following chart shows the result of the clustering algorithm:

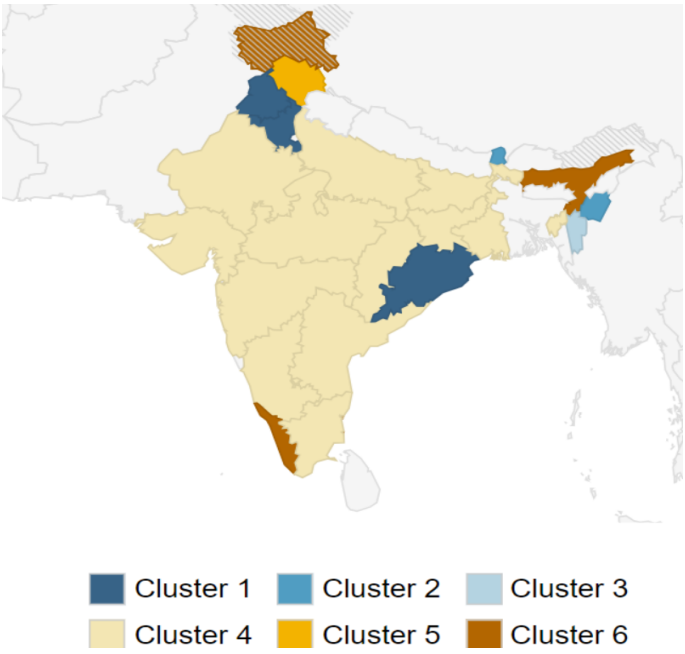

Figure 4. Cluster Assignment: Map of India

Each color represents a unique cluster, determined by analyzing the mean values of the target variable associated with each cluster. The sequence of the clusters, from Cluster 1 through Cluster 6, reflects the increasing average of the target variable.

The visualization reveals that clustering partially aligns with geographical patterns. Single-state clusters, such as those in the north and east, are near India's borders, suggesting that certain population characteristics, captured by the k-means algorithm, might be influenced by geographic, cultural, or economic factors associated with border regions.

In contrast, larger clusters are more centrally located, reflecting shared population traits among those hinterland states. This highlights how regional differences are captured by the k-means algorithm, even though it does not directly consider geography.

3.3. Target Encoding

Since some of the classification models used require numerical variables, and in order to avoid losing relevant information that could impact the loan approval process by banks, we decided to apply Target Encoding to the categorical variables **City** and **Profession**.

Target Encoding is a technique used to convert categorical variables into numerical values by replacing each category with the mean of the target variable for that category. This method helps preserve useful information while transforming the data for model compatibility. To do this, we used Python Script (legacy) nodes with an ad-hoc function. Then, we also utilized a Model Writer node to save the metrics calculated for both **City** and **Profession**. These metrics are essential, as they need to be applied to the test set to prevent data leakage.

Furthermore, we applied Target Encoding to the **State** variable, whose categories were grouped into clusters. However, in this case, we computed the mean of the **Risk_Flag** variable based on the obtained clusters, rather than the individual categories. Even in this case, we saved the calculated metrics.

3.4. Test Pre-processing

In the pre-processing phase, it is essential to ensure that the training and test datasets are processed in a consistent manner. This consistency is crucial because any discrepancies between the two sets can lead to unreliable model evaluations and biased performance results. When applying transformations, such as encoding, we must be sure that the same metrics calculated on the training are applied to the test set. This ensures that the test data is treated in the same way as the training data, preventing data leakage and ensuring that the model's evaluation reflects its true performance on unseen data.

4. Models

We employed different machine learning techniques to identify the model that best meets the objective of predicting whether a loan applicant should be considered risky or not. Here we list the models used [6][8]:

- GRADIENT BOOSTING: it is an ensemble technique that builds decision trees sequentially. Each new tree corrects the errors made by the previous one by focusing on the residuals. This method reduces bias and improves model accuracy by iteratively refining the predictions;
- XGBOOST: it is an optimized version of Gradient Boosting that includes regularization to prevent overfitting. It uses parallel processing for faster computation and employs a more efficient algorithm for constructing trees, making it highly scalable and accurate for complex tasks;
- RANDOM FOREST: it is an ensemble method that creates multiple decision trees using random subsets of data and features. The model averages the predictions from each tree to reduce overfitting and variance, resulting in a more stable and accurate model;
- CATBOOST: it is a gradient boosting algorithm specifically designed to handle categorical features effectively. It uses ordered boosting to reduce overfitting and improves prediction accuracy with minimal preprocessing, making it well-suited for datasets with a mix of numerical and categorical data;
- STACKED ENSEMBLE: it combines the predictions from multiple base models using a meta-model. The meta-model learns how to optimally combine the base models' predictions to improve overall performance, often leading to more accurate and robust results than any individual model. In order to build this model, we needed to obtain the predicted probabilities from the previous four model, which constitute our base models. In this case, the Logistic Regression was selected as meta-model.

For XGBoost, Gradient Boosting and Random Forest we pre-processed the training set used to learn these models using all the techniques described in Section 3 . On the other hand, for the CatBoost model, which supports natively categorical variables, we decided to use the training set with the original categories for **City**, **State**, and **Profession**.

4.1. Dealing with an Imbalanced Dataset

As seen in the section Section 2.1, the dataset exhibits a strong imbalance, with 87.70% of observations belonging to class 0 and 12.30% to class 1. If not appropriately addressed, this imbalance can lead to misleading results. A common issue in imbalanced datasets is that models may predominantly predict the majority class, leading to high accuracy but poor performance on the minority class.

To address this issue, we applied only to the training set a combined approach of undersampling the majority class and oversampling the minority class. We chose this strategy to create a nearly balanced training set.

To perform the undersampling of class 0, we used the Row Sampling node, which allows for random selection of a specified number of rows to retain. In this case, we applied an 80% random sampling, reducing the number of instances in the majority class. On the other hand, for oversampling, we employed the Bootstrap Sampling node, which performs a random sampling with replacement of the instances of class 1. This means that some observations from the minority class are selected multiple times, while others may not be selected at all. This process ensures that the minority class is adequately represented, providing the model with more data points to learn from.

Ultimately, the goal is to improve the model's ability to make accurate predictions for both classes, avoiding misleading results from relying solely on accuracy, which may not truly reflect the model's performance on imbalanced data.

5. Parameter Optimization

Parameter optimization is crucial in machine learning because it allows to fine-tune model hyperparameters, maximizing performance on the validation set and ensuring better generalization [4]. During the training phase, we carried out parameter optimization for all models to identify the best combination of hyperparameters in terms of AUC. This process involved two different approaches: 3-fold cross-validation and the Holdout method [1]. The choice of method depended on the specific characteristics of the models and the computational cost associated with their training.

5.1. Cross Validation

3-fold cross-validation divides the training dataset into 3 subsets or folds. During each fold, the model is trained on a subset of the data and validated on the remaining portion. This process is repeated across all folds, and the results are averaged to select the best parameter combination. We used this method for Gradient Boosting, XGBoost, Random Forest and Stacked Ensemble, since they present a manageable computational cost.

5.2. Holdout Method

The Holdout method involves splitting the training dataset into two disjoint subsets: a sub-training set and a validation set. The model is trained on the sub-training set and validated on the validation set to identify the optimal parameter configuration. Due to the computationally intensive training process of CatBoost, we selected the Holdout method for this model.

5.3. Parameter Optimization Description

For both methods, we paid special attention to address the imbalance in the dataset. As described in Section 4.1, we balanced each sub-training subset, before training it with the different parameter combinations. Meanwhile, we kept the validation subsets unbalanced to provide a realistic performance evaluation and reduce the risk of overfitting.

To perform the parameter optimization loop, we chose as a search function, which defines how the algorithm explores the parameter space to identify the best hyperparameter combination, the method Random Search, which randomly selects and evaluates combinations of parameters within a defined range, specified by start and stop values. An optional step size can be used to refine the possible parameter values. The loop runs for a set number of iterations, 100 in this case, or stops early if no improvement in the objective value is detected over a specified number of rounds. Parameters are sampled with replacement, meaning duplicate combinations may occur. While duplicates are processed only once, they still count toward the iteration limit, which may result in fewer unique evaluations than expected.

Here is a brief description of the parameters tested for each model and the resulting best combination.

5.3.1. Parameters Explanation

For all the models trained, with the exception of the Stacked Ensemble model, we tested the number of trees and maximum depth parameters with various values. The number of trees parameter controls how many trees are built in the model: more trees can improve performance but may lead to overfitting if excessive. The maximum depth parameter limits the maximum depth of each tree, influencing the model's complexity and ability to capture patterns without overfitting.

Additionally, we tested only for XGBoost, Gradient Boosting, and CatBoost the learning rate parameter, which determines the contribution of each tree to the final model's prediction. Smaller values help prevent overfitting by making more gradual updates, but they typically require more trees for effective learning, while larger values speed up training but may reduce the model's ability to generalize to unseen data.

5.3.2. Gradient Boosting and XGBoost

Parameter	Start value	Stop value	Step size
number of trees	100	500	200
max_depth	20	80	20
learning rate	0.01	0.1	0.05

Table 2. Gradient Boosting and Xgboost Tested Parameters

The best parameter combination for Gradient Boosting includes 500 trees, 40 as max_depth and 0.01 as learning rate, while for XGBoost these values are respectively 100, 60 and 0.06.

5.3.3. Random Forest

Parameter	Start value	Stop value	Step size
number of trees	20	100	20
max_depth	40	100	20

Table 3. Random Forest Tested Parameters

For the Random Forest model, the parameter optimization loop returned as the best combination 60 trees and 80 as max_depth.

5.3.4. CatBoost

Parameter	Start value	Stop value	Step size
number of trees	200	800	300
max_depth	8	12	2
learning rate	0.03	0.09	0.03

Table 4. CatBoost Tested Parameters

For the CatBoost model the best combination of parameters identified includes 800 as number of trees, 12 as max_depth and 0.03 as learning rate.

6. Focus on the Stacked Ensemble Model

To build the Stacked Ensemble model, we first divided the training set from the initial partition into a sub-training set and a validation set, allocating 75% of the observations to the sub-training set and 25% to the validation set. We then applied all the pre-processing steps described in Section 3 to the sub-training set, saving the transformations to ensure consistency when applying them to the validation set. We first balanced the subtraining set as explained in Section 4.1, then trained the base models on it, performing hyperparameter tuning through cross-validation as seen in Section 5.1. After training, we evaluated the base models on the validation set, using their predicted probabilities as training data for the Stacked Ensemble.

After recombining the sub-training and validation sets, we balanced this training set and retrained the base models on it using the best parameters found in the optimization phase. We then used these re-trained models to generate predictions on the test set, providing the input probabilities for the Stacked Ensemble's test evaluation. This approach effectively prevents data leakage, ensuring a clear separation between training and evaluation data. Before training the meta-model, we first searched for the best combination of parameters similar to what was done for the other models.

6.1. Parameters for the Meta-Model

For the Stacked Ensemble model, since we decided to use the Logistic Regression as meta-model, we tested different parameters compared to the previous models. Regularization techniques, including Laplace, Gauss, and Uniform, control the model's complexity. In particular, Laplace regularization encourages sparsity by pushing some coefficients to zero, Gauss penalizes large coefficients to prevent overfitting, and Uniform applies a constant penalty to all the coefficients, ensuring uniform regularization. The variance parameter reflects the model's sensitivity to changes in the training data: handling this parameter is important to avoid overfitting. The step parameter adjusts the learning rate, influencing how quickly the model converges. The epoch parameter refers to how many times the entire dataset is passed through the model: more epochs can improve fitting, but without proper regularization, they may lead to overfitting. Finally, epsilon sets the stopping criteria for optimization, ceasing when changes become minimal to ensure efficient convergence.

Parameter	Tested Values		
regularization	Uniform	Gauss	Laplace
variance	0.1	1	10
step	0.01	0.1	1
epoch	50	100	500
epsilon	1e-4	1e-3	1e-2

Table 5. Stacked Ensemble Tested Parameters

In this case, the best combination of parameters identified includes Laplace as the regularization parameter, 0.1 for variance, 0.01 for step, 500 for epochs and 1e-2 for epsilon.

7. Results of the Analysis

Once we identified the optimal parameters for each model, we balanced the entire training set and retrained the models using these parameters. Finally, we evaluated the models' performance on the independent test set, ensuring unbiased performance metrics.

Since Accuracy is not the most reliable method for comparing the results of different classifiers in presence of imbalanced data, we chose to use in addition other metrics, such as Precision, Recall, F1-score and AUC.

Accuracy is the fraction of instances which are correctly predicted by the classification model:

$$\text{Accuracy} = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}$$

Precision is the proportion of true positive predictions out of all the positive predictions made by the model:

$$\text{Precision} = \frac{TP}{TP+FP}$$

Recall is the proportion of true positive predictions out of all actual positive instances in the dataset.

$$\text{Recall} = \frac{TP}{TP+FN}$$

F1-score is the harmonic mean of Precision and Recall:

$$\text{F1} = \frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision+Recall}}$$

AUC, Area Under the Curve, is the area under the Receiver Operating Characteristic (ROC) curve. It represents the model's ability to distinguish between classes. AUC ranges from 0 to 1, where a value of 1 indicates perfect classification and a value of 0.5 indicates random guessing.

The ROC curve is a graphical representation of a classifier's performance. It plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings. The

curve helps to visualize how well the model can discriminate between positive and negative classes.

The following table presents the values of the metrics mentioned above for each model, offering a comprehensive overview of the achieved performance.

Classifier	Accuracy	Precision	Recall	F1
Gradient Boosting	0.887	0.528	0.784	0.631
XGBoost	0.887	0.529	0.778	0.630
Random Forest	0.896	0.559	0.744	0.638
CatBoost	0.890	0.535	0.776	0.633
Stacked Ensemble	0.862	0.467	0.830	0.598

Table 6. Estimated Performance Metrics

As seen in Table 6, the overall performance of each metrich is quite similar for all the models considered. In particular, the relatively low values in Precision compared to Recall are an indication that the models are more sensitive to risky loan applicant, with the risk of committing more false positive predictions.

However, a deeper analysis allowed us to state that Random Forest shows a solid overall performance, ranking first in Accuracy and Precision. Although its Recall is slightly lower compared to other models, its robust F1-score and higher Precision indicate that it is well balanced in terms of minimizing false positives while still capturing a substantial number of positive cases. Gradient Boosting and XGBoost show similar performance, with good Accuracy and Recall values. However, these models fall behind Random Forest in Accuracy, Precision and F1-score. CatBoost follows closely behind, with competitive performance in terms of Accuracy and Recall. However, its slightly lower Precision makes it less favorable than Random Forest, while Stacked Ensemble, although excelling in Recall, shows a notable drop in Precision and overall performance.

The following figure can be used to compare the classifiers based on their ROC curve and AUC.

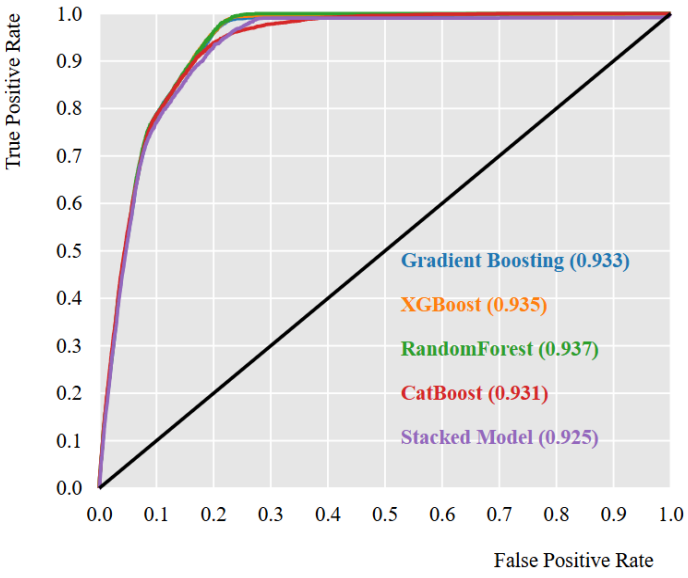

Figure 5. ROC Curves

The ROC curves for the different classifiers are nearly identical, reflecting very similar performance across models. Examining the AUC values reveals that the best-performing model is Random Forest, achieving an AUC of 0.937, closely followed by XGBoost at 0.935. The Stacked Ensemble model shows the lowest performance, with an AUC of 0.925. These results suggest that all classifiers are highly effective at distinguishing between positive and negative classes, as evidenced by the overlap of their ROC curves. The small differences in AUC values highlight the models' strong discriminatory power, with only marginal variations in performance.

8. Conclusions

This project explored the application of machine learning techniques to predict the loan risk of applicants based on socioeconomic data. By addressing critical challenges, such as class imbalance and the high diversity of categorical variables, we developed a robust framework for model evaluation and selection.

The results highlight Random Forest as the most effective model, achieving an AUC of 0.937 and demonstrating quite a strong balance between Precision and Recall. While XGBoost and Gradient Boosting performed comparably, Random Forest provided superior reliability, making it the preferred choice for tasks requiring accurate identification of risky loan applicants. Other approaches, such as the CatBoost and the Stacked Ensemble, showed potential but were less consistent due to lower AUC scores.

Our analysis demonstrates the importance of rigorous preprocessing, such as balancing techniques and encoding methods, to ensure model effectiveness in real-world scenarios. These findings provide a solid foundation for future efforts aimed at improving credit risk assessment.

Moving forward, future research could expand the scope by exploring additional features or employing more advanced ensemble techniques. Such developments could enhance the accuracy and interpretability of the models, providing financial institutions with even more reliable tools for decision-making.

References

[1] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection", *Statistics Surveys*, vol. 4, no. none, 2010, ISSN: 1935-7516.

[2] M. R. Berthold, N. Cebron, F. Dill, *et al.*, "KNIME: The Konstanz Information Miner", in *Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007)*, Springer, 2007, ISBN: 978-3-540-78239-1.

[3] C. Dormann, J. Elith, S. Bacher, *et al.*, "Collinearity: A review of methods to deal with it and a simulation study evaluating their performance", *Ecography*, vol. 36, no. 1, pp. 27–46, 2013.

[4] C. Erden, H. I. Demir, and A. H. Kökçam, *Enhancing machine learning model performance with hyper parameter optimization: A comparative study*, 2023. arXiv: 2302.11406 [cs.LG].

[5] B. Guy, P. Vasyl, D. Susmita, and D. Somnath, *clValid: Validation of clustering results*, R package version 0.7, 2021.

[6] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R*, 2nd. New York: Springer, 2023, pp. 327–352.

[7] Kaggle. “Loan approval dataset”. (2021), [Online]. Available: https://www.kaggle.com/datasets/rohit265/loan-approval-dataset.

[8] I. D. Mienye and Y. Sun, “A survey of ensemble learning: Concepts, algorithms, applications, and prospects”, *IEEE Access*, vol. 10, pp. 99 129–99 149, 2022.

[9] K. Modarresi and A. Munir, “Generalized variable conversion using k-means clustering and web scraping”, in *Computational Science – ICCS 2018*, Y. Shi, H. Fu, Y. Tian, *et al.*, Eds., Springer International Publishing, 2018, pp. 247–258.

[10] 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.

[11] B. Schloerke, D. Cook, J. Larmarange, *et al.*, *Ggally: Extension to 'ggplot2'*, R package version 2.2.1, 2024.