Machine Learning for Mortgage Defaults

Can we succeed where banks failed?

Gabriele Cialdea Rebecca Florea Simone Gallo Giulio Lanza

Abstract

We evaluate the strengths and weaknesses of several classifiers in the context of home loan applications, aiming to identify borrowers at risk of delinquency. Since loan defaults are relatively rare—largely due to rigorous bank screening processes—real-world datasets are typically highly skewed towards the non-default class. To mitigate this imbalance, we applied dimensionality reduction using an autoencoder and experimented with rebalancing techniques.

Keywords: mortgage risk prediction, autoencoder, rebalancing.

1 Introduction

We analyzed a home loan dataset provided by Freddie Mac, a U.S. federal agency that securitizes residential mortgages. Our analysis integrates both loan origination and loan performance data over the two-year period from 2020 to 2022, which we further enriched by matching with HMDA (Home Mortgage Disclosure Act) data. In total, we gather 86,000 instances and 14 variables, which are summarized in Table 1.

In terms of data preprocessing, we converted `applicant_age` and `debt_to_income`, originally reported in interval formats, into continuous values by taking the midpoint of each interval. Additionally, we constructed a dummy variable for `applicant_race`.

Variable	Description
`credit_score`	Numerical value representing applicant's creditworthiness
`total_units`	Total number of housing units involved in the loan
`occupancy_type`	Type of occupancy: primary, secondary, investment
`debt_to_income`	Ratio of applicant's debt payments to income
`loan_amount`	Amount of money approved for the loan
`interest_rate`	Annual interest rate applied to the loan
`loan_purpose`	Purpose of the loan: purchase, refinance, home improvement
`loan_term`	Duration of the loan in months
`defaulted`	Whether the loan was defaulted (1 = yes, 0 = no)
`loan_to_value`	Ratio of the loan amount to the appraised property value
`income`	Annual income of the loan applicant
`applicant_race`	Race of the loan applicant
`applicant_sex`	Sex of the loan applicant
`applicant_age`	Age of the loan applicant

Table 1: List of variables with descriptions used in the loan dataset

Regarding collinearity, the strongest observed correlation is between `income` and `loan_amount`, with a moderate coefficient of 0.40. Similarly, the variance inflation factors (VIFs) for `debt_to_income`, `income`, and `loan_amount`—a set of variables one might expect to be highly collinear—are all relatively low, with values below 2.

An overarching finding from our analysis is that `credit_score` emerges as the most informative variable. Based on exploratory analysis, we observe a strong rank correlation between `credit_score` and the realized default probability, defined as the proportion of applicants who defaulted within each credit score range. Figure 1 illustrates this relationship using a second-order polynomial approximation of the observed scatterplot.

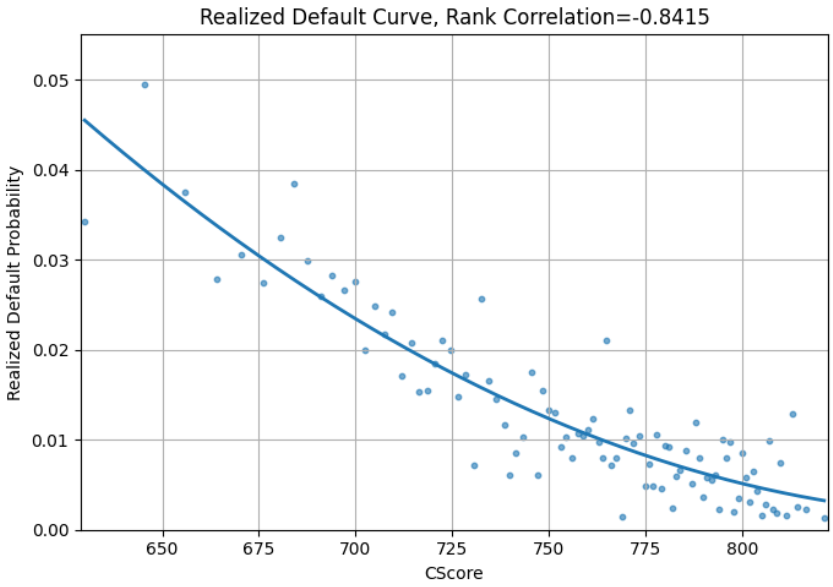

Figure 1: Second-order approximation

2 PCA and clustering

As a dimensionality reduction technique, we chose to apply Principal Component Analysis (PCA). After running PCA on the dataset, the elbow method—based on the explained variance—suggested selecting between 4 and 7 components (see Figure 2). Since 6 components explained at least 50% of the total variance, we decided to reduce the dataset to these 6 principal components for our analysis.

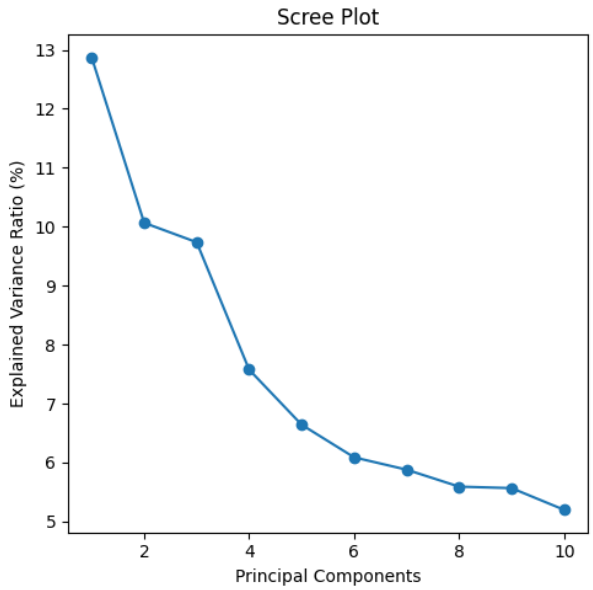

Figure 2: Scree plot on the first 10 principal components

For the Exploratory Data Analysis, we performed unsupervised clustering without predefined expectations. Using the K-Means algorithm, and optimizing both the Silhou-

ette Score and the elbow method, we identified 7 as the optimal number of clusters. Unfortunately, the Adjusted Rand Index is very poor, suggesting that clusters did not match well with the classes.

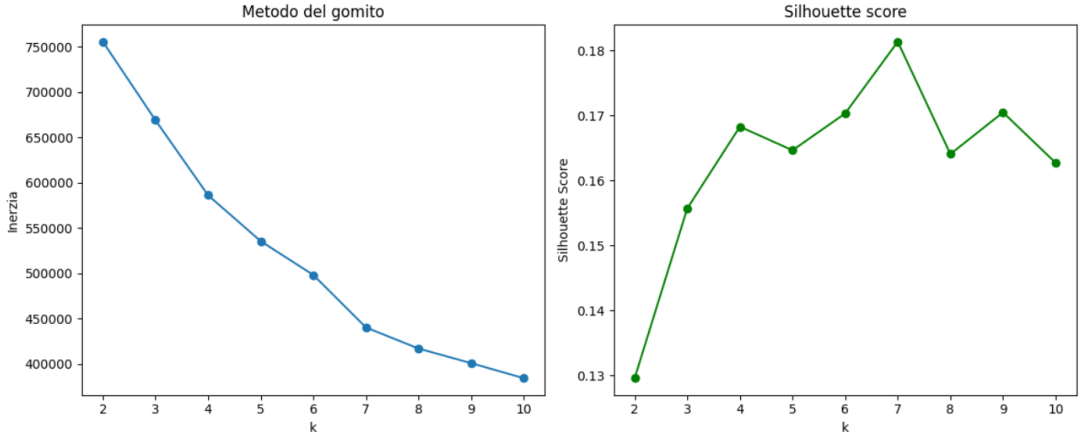

Figure 3: K-Means

3 Logistic Regression with PCA

3.1 Training

We decided to train a logistic regression classifier using the first six principal components derived from our dataset in order to extract meaningful information, valuable pattern or even crucial insights from our data.

During our initial training attempt, the logistic regression classifier failed to produce meaningful results, as it consistently predicted every data point as belonging to the non-default class. This outcome indicated that the model was not learning the distinction between the classes, a behavior commonly associated with a significantly imbalanced dataset. To mitigate this issue and ensure the model considered both classes appropriately, we introduced class weighting into the training process. By assigning different weights to each class, we encouraged the classifier to penalize misclassifications of the minority (default) class more heavily, thereby making it more sensitive to those cases.

We explored a range of possible class weight combinations using grid search, systematically evaluating model performance for each configuration. The effectiveness of each set of weights was assessed by examining the corresponding confusion matrix, which allowed us to visually and quantitatively evaluate the trade-off between true positives, false positives, and false negatives. Through this approach, we were able to identify a set of class weights that led to a more balanced and meaningful classification performance (see Table 2 and Figure 4).

Actual \ Predicted	Positive	Negative
Positive	9684	6881
Negative	64	150

Table 2: Confusion Matrix with adjusted Weights

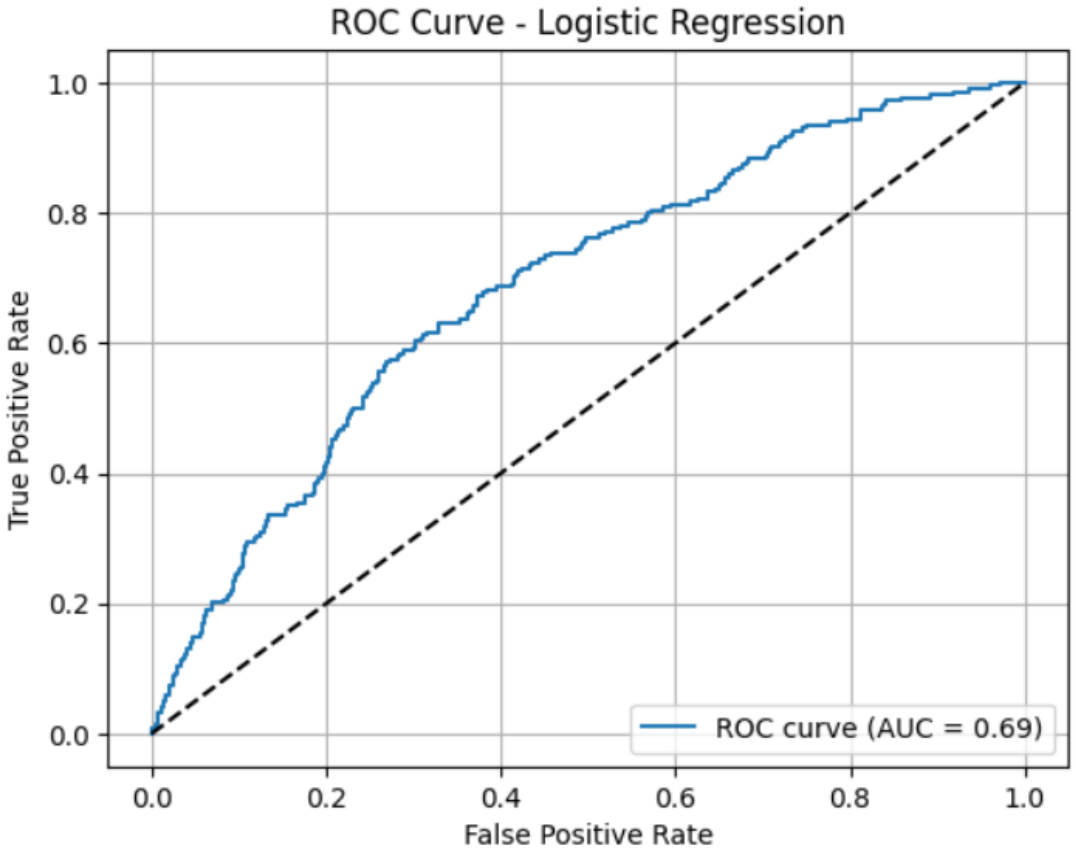

Figure 4: ROC curve for the Logistic Regression on Six Principal Components

3.2 Explanation

Since our model was trained exclusively on the first six principal components obtained through Principal Component Analysis (PCA), we did not have direct access to the weights associated with each original feature. Instead, the logistic regression model produced weights corresponding only to the selected principal components, which are linear combinations of the original variables. To recover an interpretable representation of the model in terms of the original feature space, we used the PCA loadings matrix, which contains the coefficients that define how each principal component is constructed from the original features. Specifically, we took the matrix composed of the loadings for the first six principal components and premultiplied it by the vector of logistic regression weights. This operation effectively projected the model coefficients back into the original feature space, allowing us to estimate the contribution or "weight" of each original feature

in the final classification model.

$$w_{\text{orig}} = w^T \cdot L_6$$

$$L_6 \in \mathbb{R}^{6\times 18}, \quad w \in \mathbb{R}^6, \quad w_{orig} \in \mathbb{R}^{18}$$

This approach provided a way to interpret the influence of the original features despite the dimensionality reduction step, and helped us gain insight into which features were most relevant in distinguishing between the classes.

Feature	Weight
credit_score	-0.16363239
total_units	-0.04573444
occupancy_type	0.1220353
debt_to_income	0.31075724
loan_amount	0.05898812
interest_rate	0.18942094
loan_purpose	0.19113466
loan_term	0.27526974
loan_to_value	0.30016855
income	-0.2518229
applicant_sex	0.08009453
applicant_age	-0.15974871
race_1	0.00868074
race_2	-0.00174289
race_3	0.07159245
race_4	0.00964971
race_5	-0.10770837
race_6	0.09741807

Table 3: Computed Feature Weights

4 Ridge and Lasso Regularization

In this section, we perform Ridge and Lasso regularization in the Logistic regression on the full dataset (with all the features). Despite logistic regression not performing very well even with the full dataset, these regularizations can be interesting for analyzing which features are the most important. Starting with Ridge regularization, in Figure 5 we can see how parameters behave as regularization strength increases. We can note for instance that some parameters stay close to zero for each regularization strength c, like "occupancy type", suggesting a low predictive power. Other parameters, the last to go to

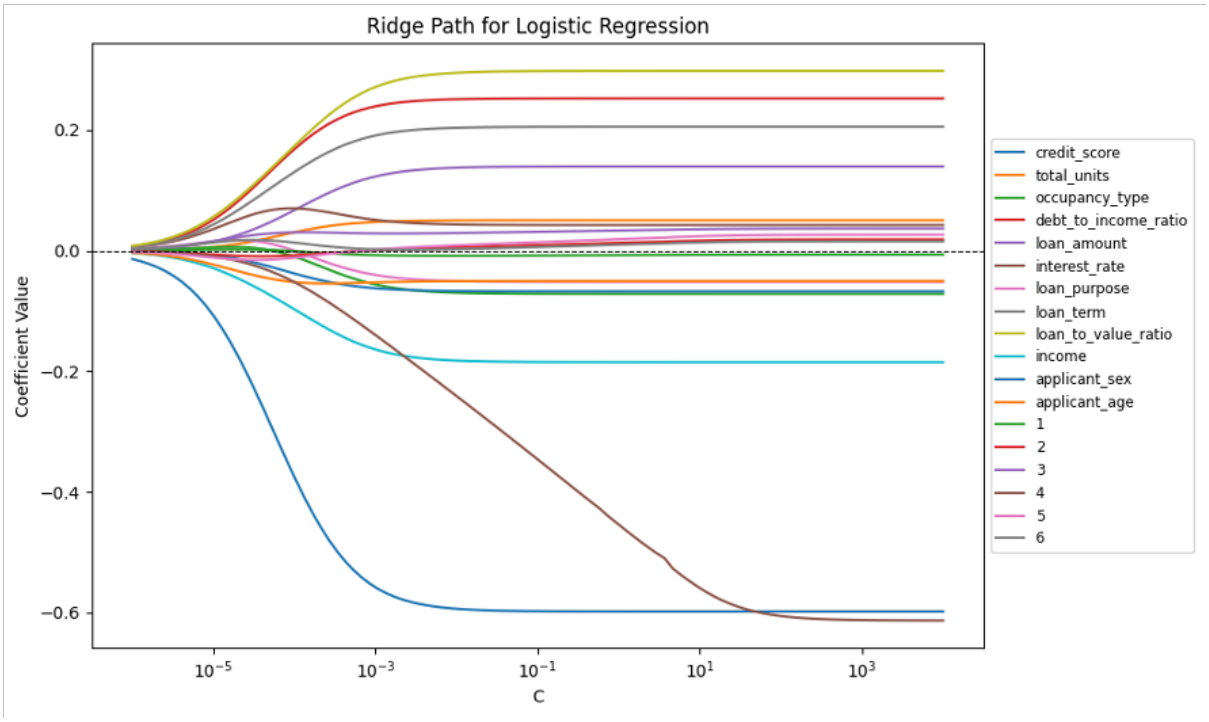

Figure 5: Logistic regression, Ridge regularization: parameters when regularization strength varies.

zero, suggest a higher predictive power, like "debt to income ratio" and "loan to value ratio" with a positive impact and "credit score", with a negative impact (which means that a higher credit score leads to a lower probability of default. In Figure 6, we can see how the test error varies with the regularization strength, which is decreasing of the parameter c and is fairly high even for low regularization strength (around 40%).

Moving to Lasso regularization, we can see similar figures for how parameters vary when test error varies (Figure 7). Similarly to Ridge regularization, the last parameters to shrink to zero are: "credit score", "loan to value ratio", "debt to income ratio" and also "loan term", that shrinks to zero just before the other. We can also look at how test error varies with c in Figure 8, which, apparently strangely, falls to zero for very low c (high regularization strength). It actually falls to 1.27%, which is exactly the percentage of defaulted values in our dataset. This value should be read together with Figure 7: we note that, for low c (below $\sim 10^{-3}$, every coefficients goes to zero, so that the model always predicts the same value: not defaulted, which explains the odd behavior of the test error. We double-checked this by performing a random undersampling on the "not defaulted" population, so that they were equal in number to the "defaulted population". The new logistic regression test error can be seen in Figure 9: in this case, when, with a low c, all the parameters go to zero, the prediction error becomes 0.5, which is coherent with the fact that the model always predict the same value and the two populations ("defaulted" and "not defaulted") have the same number of components.

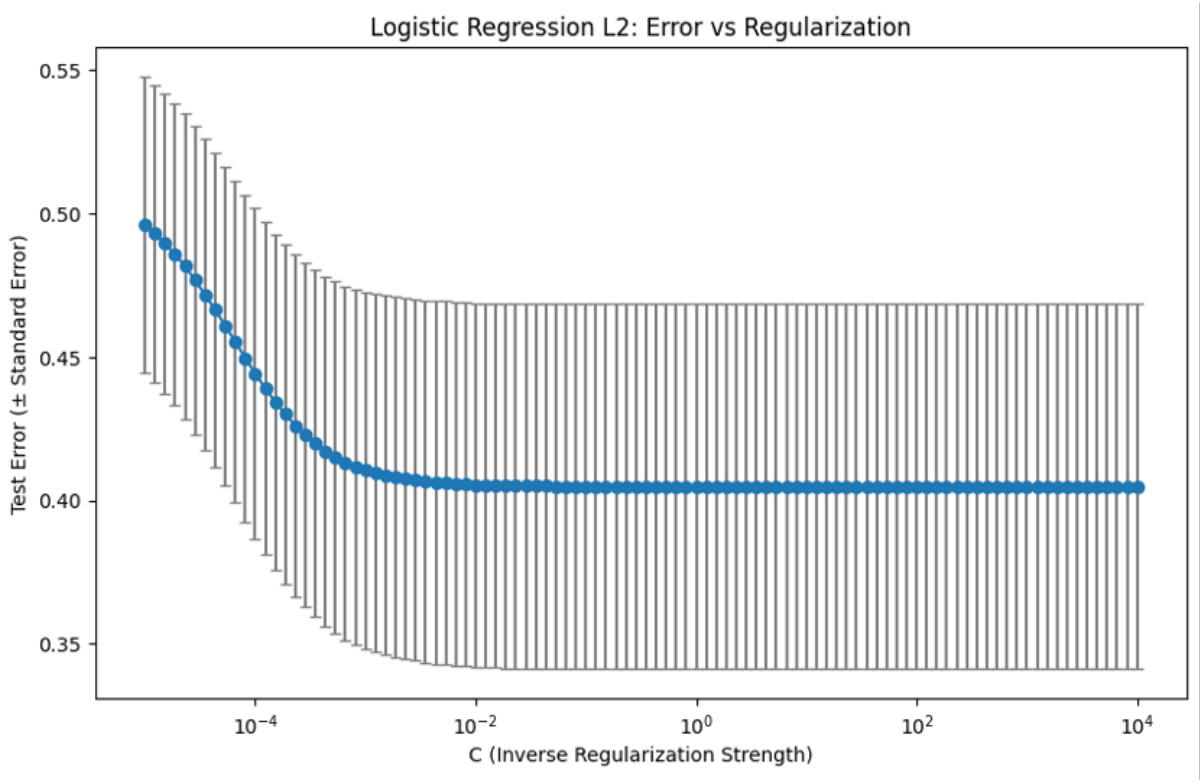

Figure 6: Logistic regression, Ridge regularization: test error when regularization strength varies.

5 Linear Discrimant Analysis

In this section we try to predict default using a Linear Discriminant analysis (LDA). We divide our dataset in a train set (70%) and test set (30%). As expected, LDA performs poorly on the whole dataset given the huge imbalance in the dataset, the confusion matrix is the following (AP=actually positive, AN= actually negative, PP=predicted positive, PN=predicted negative):

	PN	PP
AN	16553	0
AP	226	0

If we plot our data after performing the discriminant analysis, we can actually see that there is a difference in distribution between "defaulted" and "not defaulted" (see Figure 10.

If we perform random undersampling on the "not defaulted" class, the new distribution can be seen in Figure 11. Now, the new confusion matrix is the following:

	PN	PP
AN	140	64
AP	77	147

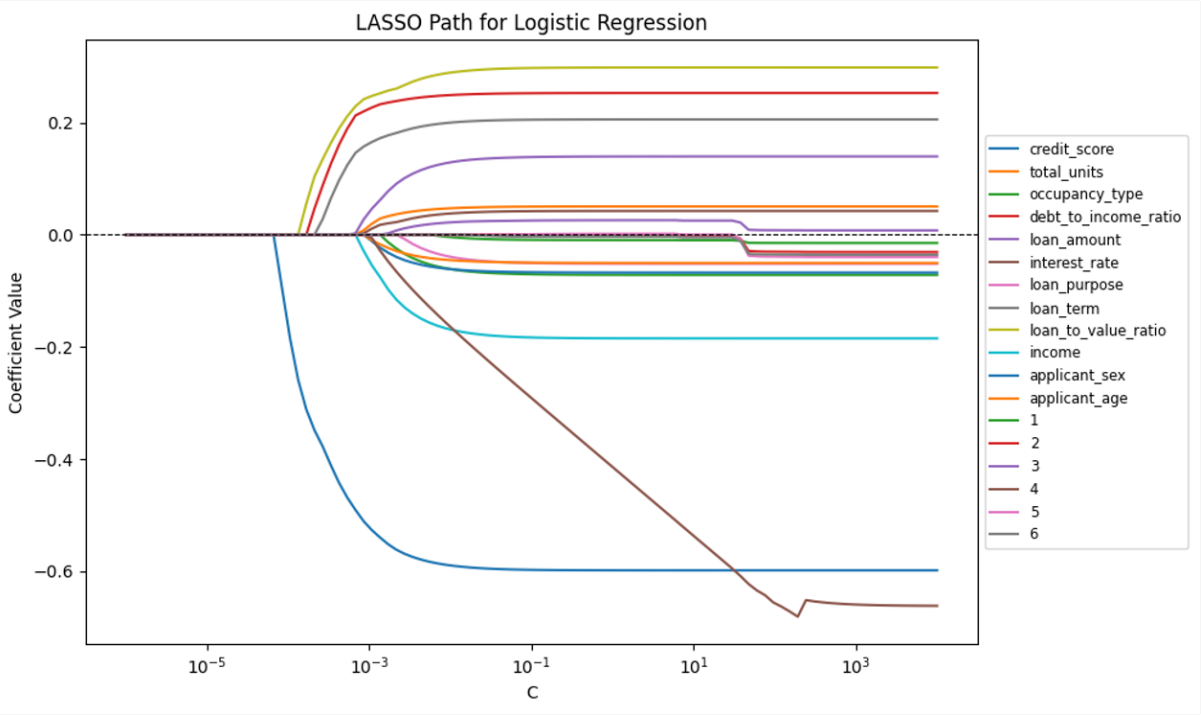

Figure 7: Logistic regression, Lasso regularization: test error when regularization strength varies

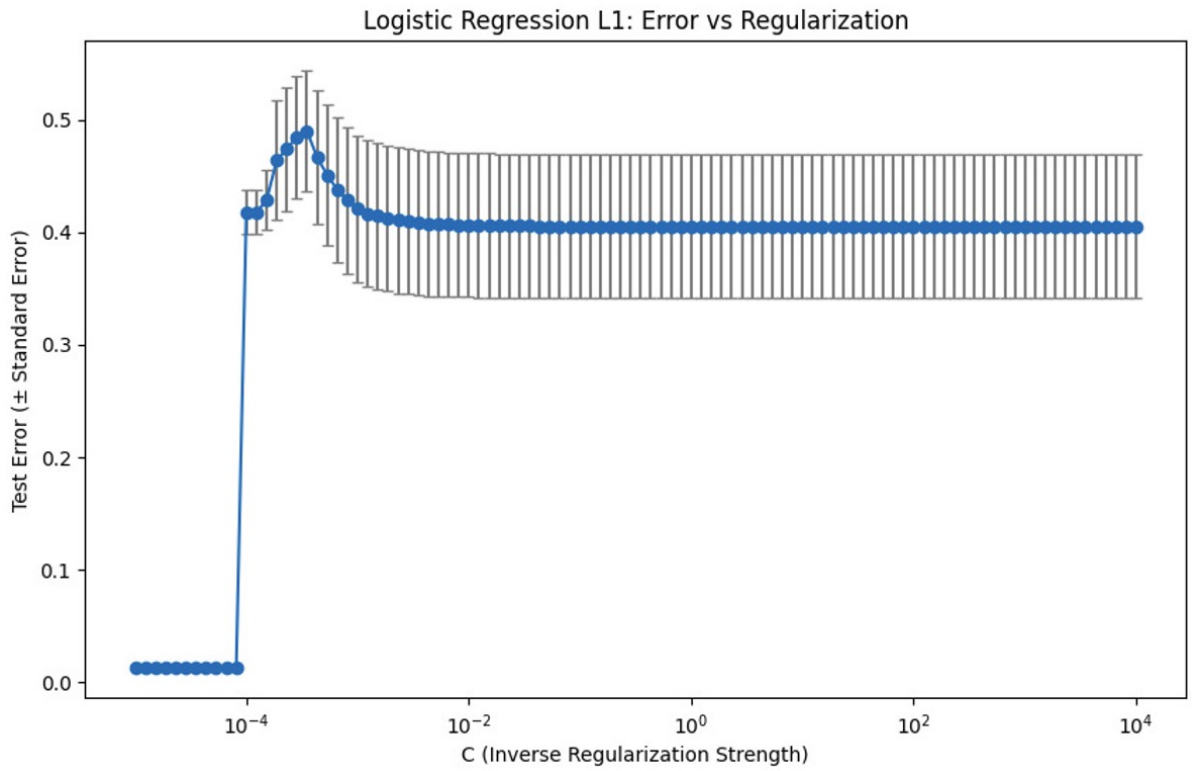

Figure 8: Logistic regression, Lasso regularization: test error when regularization strength varies.

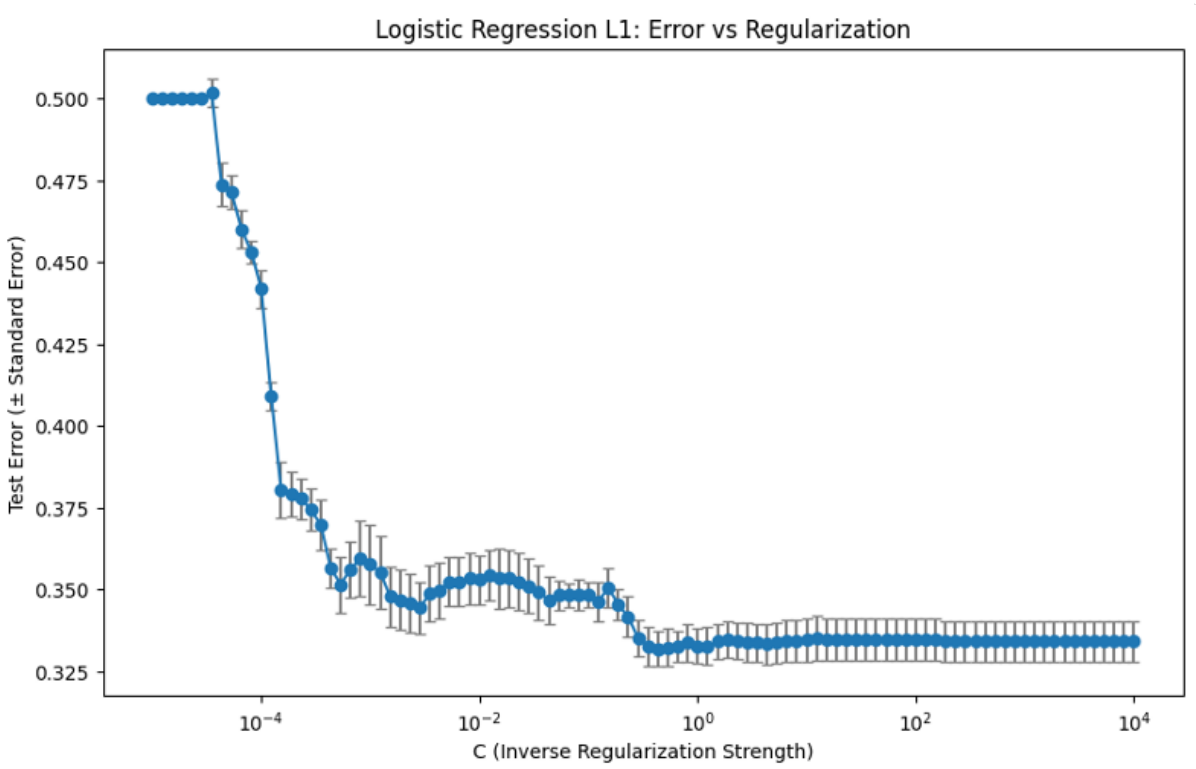

Figure 9: Logistic regression, Lasso regularization with random undersampling: test error when regularization strength varies.

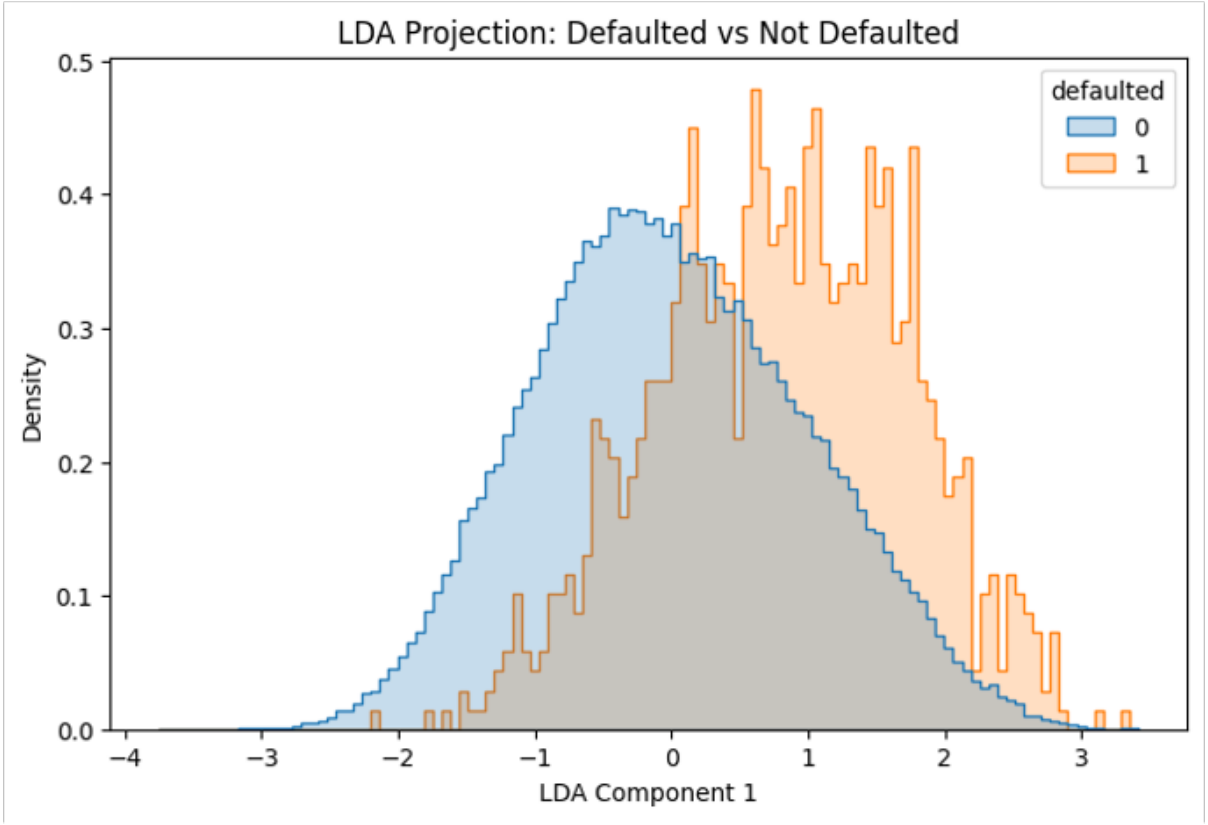

Figure 10: Distribution of defaulted (orange) and not defaulted (blue) after performing LDA. The default distribution is rescaled.

Intuitively, the performance is improved from the previous case. We can confirm it by computing some relevant indices:

	Precision	Recall	F1 score
0	0.65	0.69	0.67
1	0.70	0.66	0.68

Both precision and recall fall within the range of 0.65 to 0.70, indicating moderate performance that, while not outstanding, can be considered acceptable.

5.1 Quadratic Discriminant Analysis

A similar reasoning can be done with a Quadratic discriminant analysis: as in the LDA case, without undersampling it always predicts "not defaulted". If we perform random undersampling, the new confusion matrix becomes:

	PN	PP
AN	96	108
AP	48	176

and the indices are:

	Precision	Recall	F1 score
0	0.67	0.47	0.55
1	0.62	0.79	0.69

with a performance comparable to LDA (maybe slightly worse, considering that Recall for "1" is higher but for "0" is much worse).

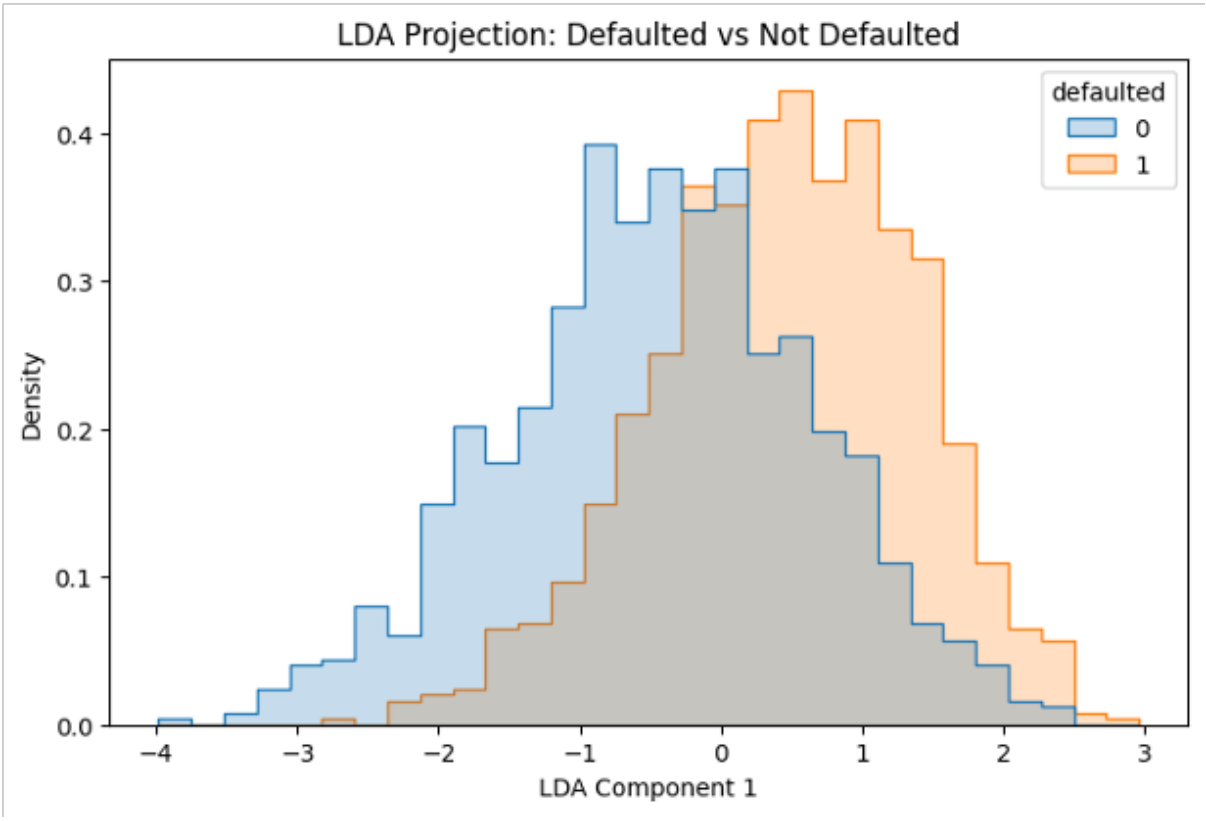

Figure 11: Distribution of defaulted (orange) and not defaulted (blue) after performing LDA and undersampling (no rescaling).

6 Neural Nets for Dimensionality Reduction

We have decided to explore a deep learning approach for unsupervised dimensionality reduction, leveraging the representational power of neural networks to uncover complex relationships within the dataset. Once the model is trained, we will compare its performance and the quality of the learned representations with those obtained from previously used methods.

6.1 Autoencoders

Autoencoders are a type of neural network used for unsupervised learning, primarily for dimensionality reduction and feature extraction [1]. They work by learning to compress input data into a lower-dimensional representation (the encoder) and then reconstructing the original input from this compressed form (the decoder). By minimizing the difference between the input and its reconstruction, autoencoders learn to capture the most important features of the data. They are widely used in applications such as anomaly detection, image denoising, and data compression [2]. We have chosen to employ autoencoders as a method for dimensionality reduction, aiming to compress the original high-dimensional data into a more compact representation with only six

dimensions. Once the dimensionality has been reduced using the encoder part of the autoencoder, we use the resulting six-dimensional representation as input to train a logistic regression classifier. Our objective is to evaluate the performance of this approach and compare it directly with the results obtained from training the same classifier on the first six principal components derived via Principal Component Analysis (PCA).

6.2 Training

We have decided to train three different autoencoder architectures in order to explore how varying the network structure affects the quality of the learned representations. Figure 12 displays the architecture of the second model.

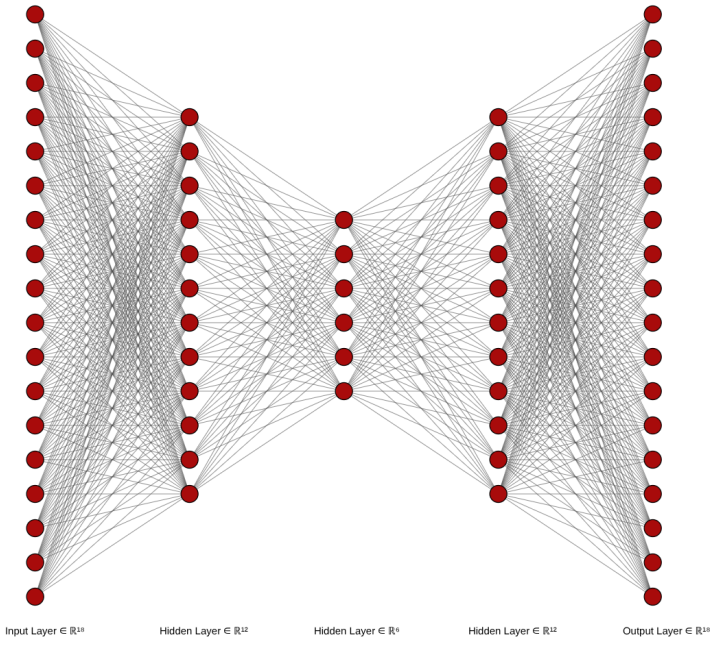

Figure 12: Autoencoder 2, with five layers

The first model is a simple autoencoder consisting of three layers: an input layer with 18 nodes, a bottleneck (or latent) layer with 6 nodes, and an output layer that mirrors the input with 18 nodes. All the layers are fully connected.

The second model extends this architecture by adding more depth and complexity. Specifically, it introduces two additional hidden layers of 12 nodes each—one before the bottleneck layer and one after it—resulting in the following architecture: $18 \to 12 \to 6 \to 12 \to 18$.

The third model is a variation of the second, where the two hidden layers of 12 nodes are replaced with layers of 14 and 10 nodes. Thus, its structure becomes: $18 \to 14 \to 10 \to 6 \to 10 \to 14 \to 18$.

These different configurations allow us to evaluate how the depth and width of the network influence the performance of the autoencoder and the usefulness of the extracted

features.

6.3 Results

The results obtained from the first and third autoencoder models were rather disappointing, as the compressed representations failed to preserve meaningful information from the original data.

This became evident when we evaluated the performance of the logistic classifiers trained on these representations, which performed only marginally better than random guessing. This observation is further supported by the confusion matrices, which show that the models struggled to distinguish between the classes effectively, indicating that the compression process did not capture relevant features for the classification task.

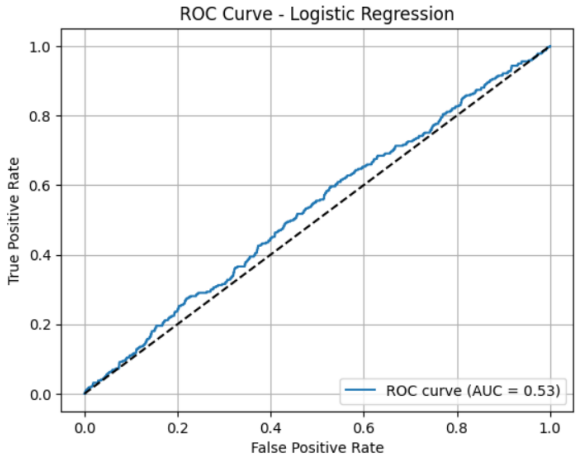

Figure 13: LC with Autoencoder 1

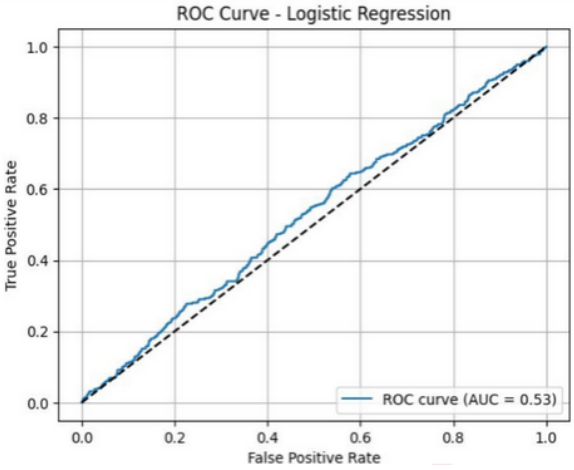

Figure 14: LC with Autoencoder 3

The results obtained from the second model were encouraging. In fact, we observed a noticeable improvement in performance compared to the classifier trained on the first six principal components. This suggests that the features extracted by the second autoencoder captured more relevant and informative patterns in the data, leading to better classification outcomes.

In conclusion, we have observed that autoencoder-based models can outperform more traditional methods in certain cases, demonstrating their potential for capturing complex data representations. However, they often require careful tuning of their architecture and training parameters to achieve good performance.

Moreover, a significant drawback of autoencoders is their lack of interpretability, as the learned features are typically abstract and not easily explained in terms of the original input variables.

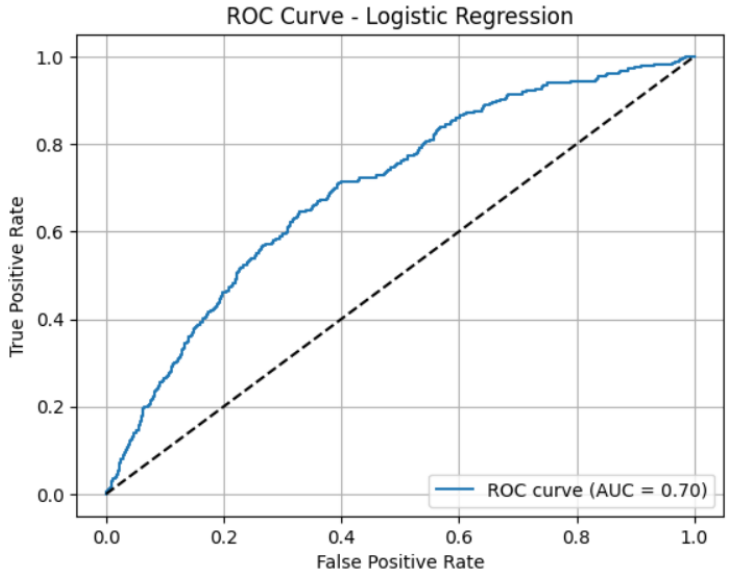

Figure 15: Logistic Classifier with second Autoencoder

7 Rebalancing

In this section, we discuss several methods for rebalancing that are implemented in the imbalanced-learn Python library [3].

We first apply random undersampling to reduce the majority class to 500% of the original minority class size, followed by SMOTE (Synthetic Minority Oversampling Technique) to augment the minority class. This combined approach often outperforms the use of SMOTE alone, particularly in highly imbalanced settings [4]. In our case, it effectively limits the amount of synthetic data generated, as the majority of observations—nearly 99%—correspond to non-defaulted loans. Figure 16 displays ROC and Precision-Recall curves of our preferred classifier.

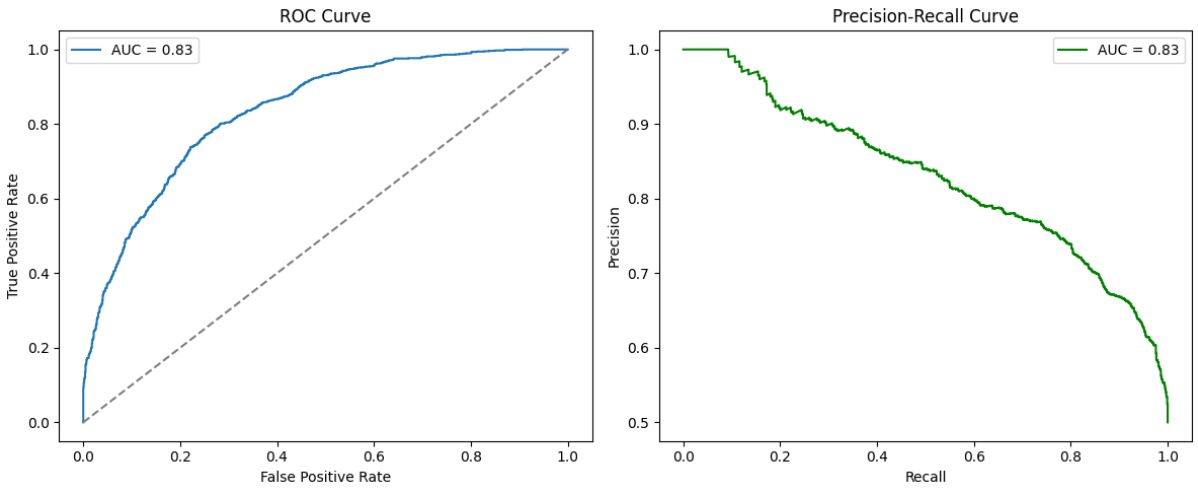

Figure 16: Random forest on Undersampled+SMOTEed dataset

It is worth noting that, since default instances are surrounded by a dense cluster of

non-default instances—a key insight consistently observed from k-Nearest Neighbors regardless of the number of neighbors—SMOTE tends to reinforce regions of the feature space predominantly occupied by non-default cases. Therefore, we judge the performance of this classifier to be encouraging.

To our delight, we explored alternative undersampling algorithms that proved to be more robust than the standard random approach. Specifically, we played with NearMiss (which includes two variants) and ClusterCentroids.

The first version of NearMiss selects majority class samples that are closest to the minority class, thereby creating a challenging decision boundary that emphasizes overlap between classes. In contrast, the second version selects majority class samples that are farthest from the minority class, as it seeks to promote a clearer separation between the classes.

ClusterCentroids, on the other hand, aims to retain the distributional structure of the majority class. It does so by applying K-means clustering (with K set to the size of the minority class) and replacing majority samples with the resulting cluster centroids [5]. A snapshot of the algorithm is provided at Figure 17.

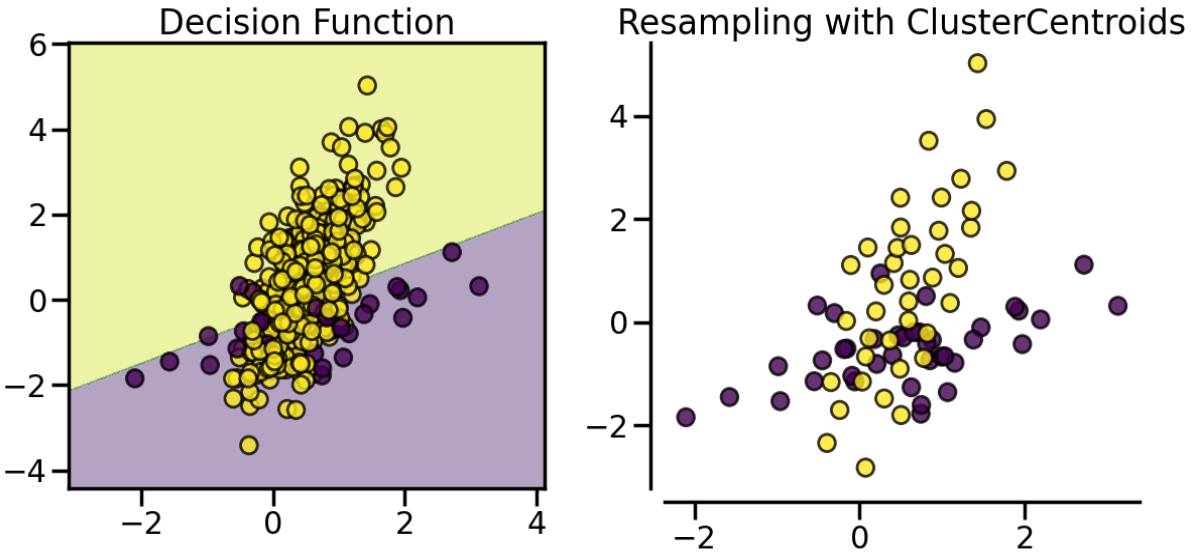

Figure 17: Illustration of the ClusterCentroids algorithm in 2D space

In our dataset, training Random Forest classifiers with NearMiss-1 and NearMiss-2 sampling strategies consistently yielded high recall for default cases. However, this performance comes at the expense of specificity.

In contrast, the ClusterCentroids method achieves a more balanced outcome: it correctly classifies non-default cases 82% of the time while also performing exceptionally well on the challenging default class, with an impressive AUC-PR of 0.95.

8 Conclusions

In our project, we investigated the determinants of loan default using a comprehensive dataset that merges Freddie Mac loan records with HMDA disclosures over the period 2020–2022.

Exploratory analysis revealed that credit score is the most informative predictor, showing a strong inverse relationship with realized default probability.

To address class imbalance inherent in default prediction, we evaluated several resampling techniques in conjunction with Random Forest classifiers. NearMiss-1 and NearMiss-2 yielded high recall for default cases, effectively flagging risky loans, but did so at the expense of specificity. In contrast, the ClusterCentroids method demonstrated a more balanced performance, achieving 82% specificity while also attaining an AUC-PR of 0.95 on the minority (default) class.

References

[1] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pages 353–374, 2023.

[2] Michael Tschannen, Olivier Bachem, and Mario Lucic. Recent advances in autoencoder-based representation learning. *arXiv preprint arXiv:1812.05069*, 2018.

[3] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017. URL http://jmlr.org/papers/v18/16-365.html.

[4] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[5] Qian Zhou and Bo Sun. Adaptive k-means clustering based under-sampling methods to solve the class imbalance problem. *Data and Information Management*, 8(3): 100064, 2024.

[6] Amir E Khandani, Adlar J Kim, and Andrew W Lo. Consumer credit-risk models via machine-learning algorithms. *Journal of Banking & Finance*, 34(11):2767–2787, 2010.