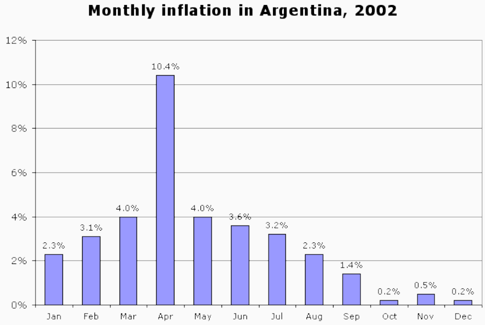
Credit Risk Analysis with Machine Learning

Viktor Sirakov, Youssef Nakhla, Hristiyan Daneshki and Nicolas Asseo

June 4, 2025

What? Why?

- **Credit:** An agreement where a borrower receives money or goods with the promise to repay later — typically with interest.
- **Credit Risk:** The risk that a borrower will fail to repay the loan as agreed.
- **Why it matters:** Argentinian debt crisis of 1998-2002.



Our goal

- **Interesting aspect:** seeing different worlds like banking/finance bridge with ML.
- **Our goal:** Use machine learning to better predict which borrowers are likely to default.

Index

Preliminary Steps

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt
0	22	59000	RENT	123.0	PERSONAL	D	35000
1	21	9600	OWN	5.0	EDUCATION	B	1000
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500
3	23	65500	RENT	4.0	MEDICAL	C	35000
4	24	54400	RENT	8.0	MEDICAL	C	35000
...	...	...	...	...	...	...	...
32576	57	53000	MORTGAGE	1.0	PERSONAL	C	5800
32577	54	120000	MORTGAGE	4.0	PERSONAL	A	17625
32578	65	76000	RENT	3.0	HOMEIMPROVEMENT	B	35000
32579	56	150000	MORTGAGE	5.0	PERSONAL	B	15000
32580	66	42000	RENT	2.0	MEDICAL	B	6475

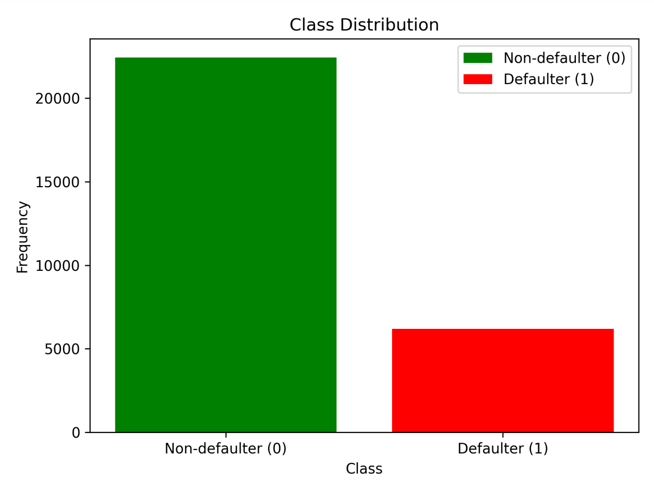
Our Data

Dataset Selection and Cleaning

- **Dataset:** Kaggle "Credit Risk Dataset" by laotse.
- Included both numerical (e.g. `loan_amnt`) and categorical (e.g. `loan_intent`) features.
- Visualized numerical columns using histograms with mean and median overlays.
- Bar plots were used to explore categorical feature distributions.
- Dropped rows with missing data to avoid complications with imputation.

Feature Engineering and Scaling

- Plotted a correlation matrix with `sns.heatmap` to identify dependencies between features.
- Standardized numeric features using `StandardScaler` to normalize ranges.
- Mapped ordinal features like `loan_grade` (A–G) to scores 1–7.



Histogram of loan status (reimbursed = 0). Shows data imbalance

Understanding the Trade-Offs

High Precision, Low Recall

- Few false alarms
- Many defaulters missed
- Risky borrowers go undetected

High Recall, Low Precision

- Most defaulters detected
- More good borrowers rejected
- Lost opportunities, customer frustration

F1-score helps balance both precision and recall in practical applications.

Missing Values Processing

- There are missing values only in two columns: employment length (895 values) and interest rate (3116 values)
- We want to find out whether the data is MCAR, MAR, MNAR and treat the values accordingly
- We proceed with statistical analysis techniques for both columns

Missing Values Analysis

- We use statistical tests to see if employment length missingness is dependent on any numerical features
- Our p-value threshold is 0.05
- We look at the mean of each numerical feature when employment rate is missing and when it isn't
- Our p-value tells us how likely we are to observe a given difference between the two means if they are identical

Employment Length

column	mean_missing	mean_not_missing	p_value
person_age	27.284916	27.747302	2.095927e-02
person_income	44229.924022	66691.878306	3.338956e-59
loan_amnt	7041.508380	9661.337815	9.815688e-45
loan_int_rate	10.036143	11.039867	7.195207e-16
loan_percent_income	0.191140	0.169612	1.712548e-07
cb_person_cred_hist_length	5.623464	5.809316	1.542460e-01

- Several p-values show statistical significance
- This is not a guarantee but an indication of possible dependence of missingness on other features
- For example, for the age column there is no practical difference between the two means

Employment Length

	precision	recall	f1-score	support
0	0.99	0.60	0.75	9501
1	0.05	0.70	0.09	274
accuracy			0.60	9775
macro avg	0.52	0.65	0.42	9775
weighted avg	0.96	0.60	0.73	9775

ROC AUC: 0.702591813232107

- Even though the ROC AUC suggests a very weak relationship between missingness and other features, the precision for missing values is very low
- Even though we identify 70% of the missing values, we classify not missing values as missing very often
- We can't strongly confirm that the values are MAR, so we conclude that they are MCAR

Interest Rate

	column	mean_missing	mean_not_missing	p_value
0	person_age	27.922657	27.714712	0.101437
1	person_income	66589.048460	66020.470490	0.630144
5	loan_amnt	9633.119384	9584.744612	0.686957
7	loan_percent_income	0.171088	0.170110	0.624219
9	cb_person_cred_hist_length	5.955071	5.788257	0.036934

p-values

	precision	recall	f1-score	support
0	0.90	0.63	0.74	8804
1	0.10	0.36	0.15	971
accuracy			0.60	9775
macro avg	0.50	0.50	0.45	9775
weighted avg	0.82	0.60	0.68	9775

ROC AUC: 0.49601523462558683

- In this case we hardly even observe statistically significant p-values
- For the sake of confirmation, we try to fit a logistic regression and get a ROC AUC of approximately 0.5
- This means we are just guessing, so the data from this column is MCAR

MNAR

- We never considered that the data could be MNAR
- We rely on intuition to rule it out
- Employment length is not a sensitive topic and it's usually a required question
- Interest rate should always be available since the bank itself imposes it
- Therefore, there is no reason to believe that the data would be missing because of its own values

Conclusion on Missing Values

- Since we concluded that in both columns the missing values are MCAR, we can either use basic imputing techniques or drop the problematic entries
- For logistic regression and knn we adopt a safe approach and drop the entries with missing values since these models are more sensitive to biased imputations of missing values
- For decision trees and random forests, we use mean imputation since these models are more robust in these scenarios

Model Implementation

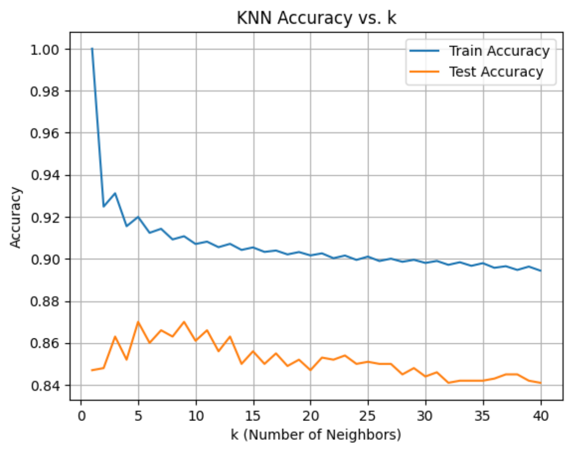
Model Implementation

KNN

K-Nearest Neighbors (KNN): Overview

- Instance-based learning — Did people with similar statistics repay the loan or default?
- How many similar cases do I look for?

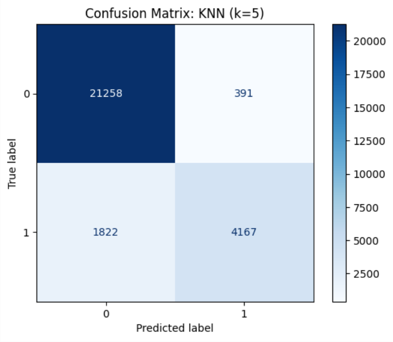
Choosing the Right k



Balance k to fight underfitting and overfitting.

KNN Performance ($k = 5$)

Confusion Matrix (Train)

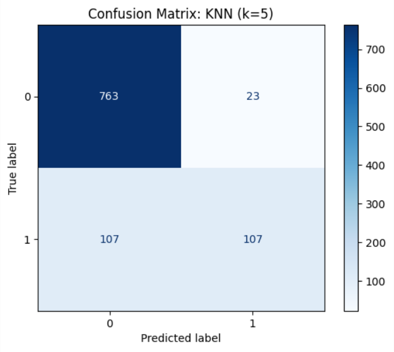


Precision: 0.914

Recall: 0.696

F1 Score: 0.791

Confusion Matrix (Test)



Precision: 0.823

Recall: 0.500

F1 Score: 0.621

KNN Observations

- Performance is skewed: predicts "good borrowers" well but misses many "bad borrowers". The data imbalance is hurting the algorithm.
- Massive improvement after data improvement.
- Good starting algorithm to compare the rest.

Model Implementation

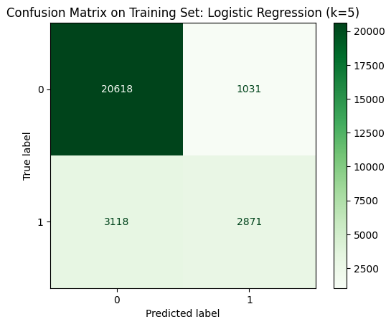
Logistic Regression

Logistic Regression: Overview

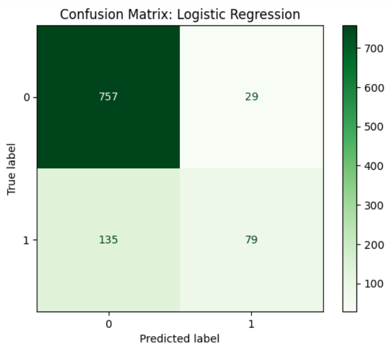
- A linear model for binary classification.
- Are you going to default(1) or not(0)?

Logistic Regression: Results

Confusion Matrix (Train)



Confusion Matrix (Test)

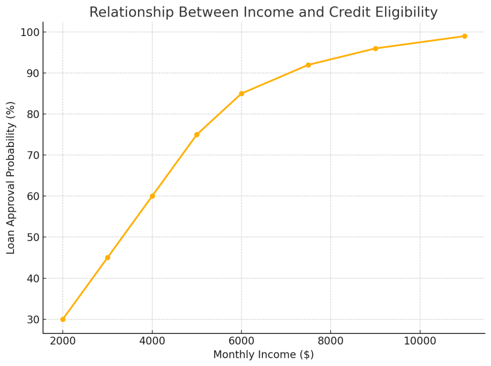


Logistic Regression: Metrics

Classification Report Test Dataset (Logistic Regression)

	precision	recall	f1-score	support
0	0.85	0.96	0.90	786
1	0.73	0.37	0.49	214
accuracy			0.84	1000
macro avg	0.79	0.67	0.70	1000
weighted avg	0.82	0.84	0.81	1000

Logistic Regression: Escape Linearity



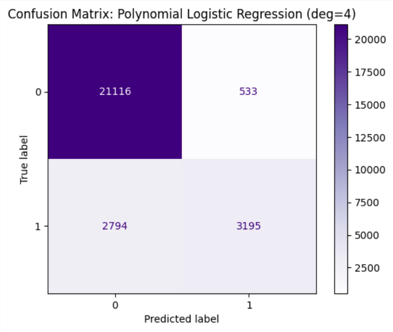
Logistic regression learns linear correlations — Is this enough?

Logistic Regression: 'New' Features

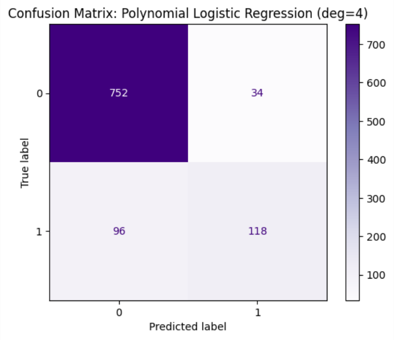
- Introducing polynomial features to introduce polynomial relations.
- Do we need polynomials of degree 10, 20 to express these relations? Where is the limit?

Logistic Regression with Polynomial Basis: Results

Confusion Matrix (Train)



Confusion Matrix (Test)



Logistic Regression: Metrics

Classification Report Test Dataset (Polynomial Logistic Regression):

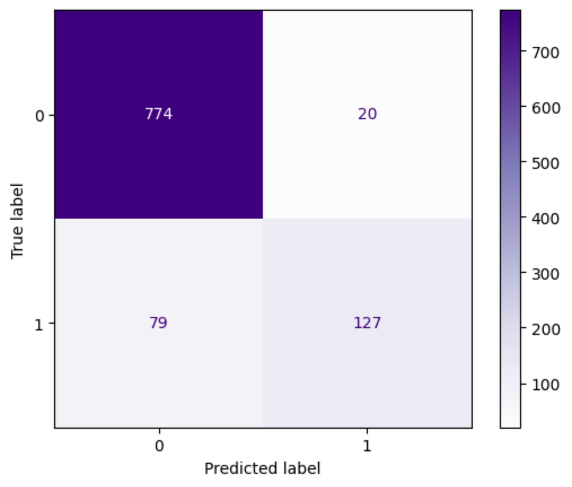
	precision	recall	f1-score	support
0	0.89	0.96	0.92	786
1	0.78	0.55	0.64	214
accuracy			0.87	1000
macro avg	0.83	0.75	0.78	1000
weighted avg	0.86	0.87	0.86	1000

Different Train Set

We chose to train the models on the first 27,000 data points, but is this the best choice for a set?

How can we determine which one is the best?

Potential Performance



Precision: 0.864 **Recall:** 0.616 **F1 Score:** 0.717

Model Implementation

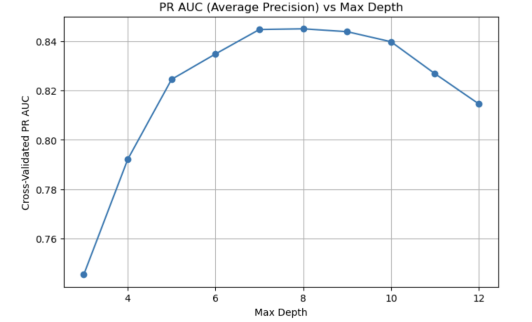
Tree-Based Models

Decision Trees

Hyperparameter Tuning

- We want to tune the max depth hyperparameter
- Since we have an imbalanced dataset, we will use the PR AUC metric
- This metric emphasizes defaulter detection
- Simply put, it works by averaging the precision across all levels of recall for a tree of given max depth

Hyperparameter Tuning



We choose max depth to be 8

Testing Performance

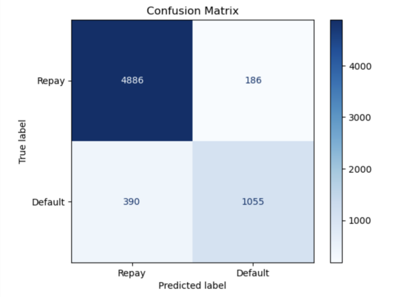
- In order to evaluate the testing performance, we look at the ROC AUC metric, the precision, the recall, the F1 score and the confusion matrix
- The ROC AUC metric gives a more balanced overview of the performance compared to the PR AUC, which concentrated on catching defaults
- In simple terms, it measures how likely the model is to rank a positive case higher than a negative case (give it a higher probability of being positive)
- The F1 score shows how balanced the precision and recall are

Testing Performance

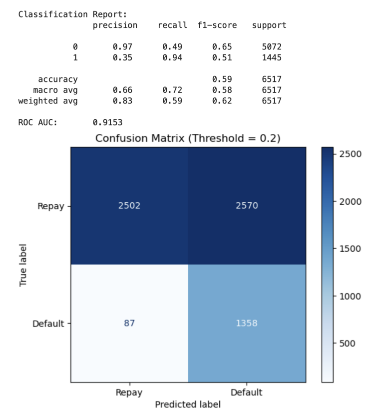
Classification Report:

	precision	recall	f1-score	support
0	0.93	0.96	0.94	5072
1	0.85	0.73	0.79	1445
accuracy			0.91	6517
macro avg	0.89	0.85	0.86	6517
weighted avg	0.91	0.91	0.91	6517

ROC AUC: 0.9153



Increasing Recall

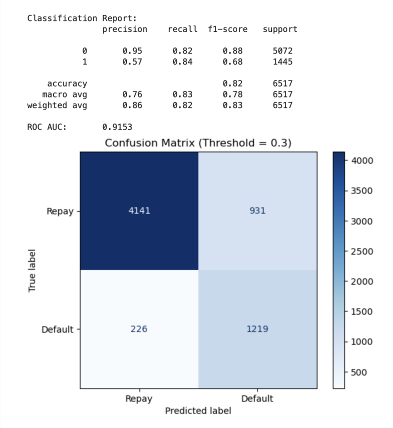


Classification Report:

	precision	recall	f1-score	support
0	0.97	0.49	0.65	5072
1	0.35	0.94	0.51	1445
accuracy			0.59	6517
macro avg	0.66	0.72	0.58	6517
weighted avg	0.83	0.59	0.62	6517

ROC AUC: 0.9153

Threshold 0.2



Classification Report:

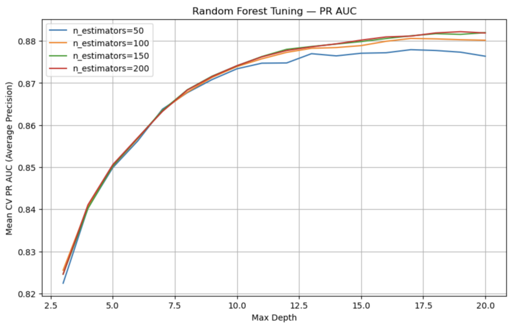
	precision	recall	f1-score	support
0	0.95	0.82	0.88	5072
1	0.57	0.84	0.68	1445
accuracy			0.82	6517
macro avg	0.76	0.83	0.78	6517
weighted avg	0.86	0.82	0.83	6517

ROC AUC: 0.9153

Threshold 0.3

Random Forests

Hyperparameter Tuning



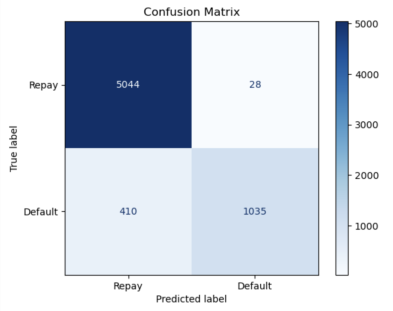
We choose max depth to be 18 with 150 estimators

Testing Performance

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.99	0.96	5072
1	0.97	0.72	0.83	1445
accuracy			0.93	6517
macro avg	0.95	0.86	0.89	6517
weighted avg	0.94	0.93	0.93	6517

ROC AUC: 0.9378

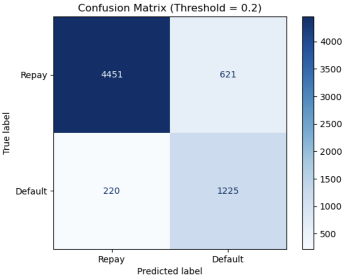


Increasing Recall

Classification Report:

	precision	recall	f1-score	support
0	0.95	0.88	0.91	5072
1	0.66	0.85	0.74	1445
accuracy			0.87	6517
macro avg	0.81	0.86	0.83	6517
weighted avg	0.89	0.87	0.88	6517

ROC AUC: 0.9378

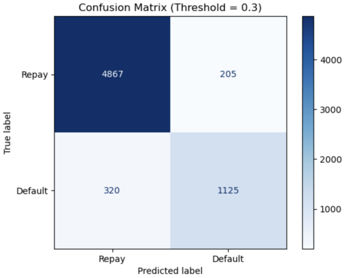


Threshold 0.2

Classification Report:

	precision	recall	f1-score	support
0	0.94	0.96	0.95	5072
1	0.85	0.78	0.81	1445
accuracy			0.92	6517
macro avg	0.89	0.87	0.88	6517
weighted avg	0.92	0.92	0.92	6517

ROC AUC: 0.9378



Threshold 0.3

Class Imbalance and SMOTE

Problem: Class Imbalance

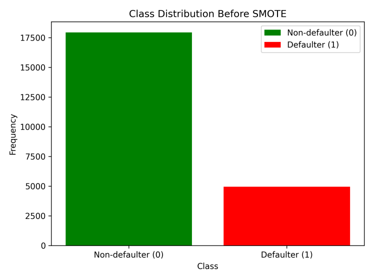
- Most borrowers in the dataset are **non-defaulters (class 0)**.
- This causes models to **ignore defaulters (class 1)** — leading to high accuracy but poor recall.

Solution: SMOTE (Synthetic Minority Over-sampling Technique)

- SMOTE generates **synthetic examples** of defaulters by interpolating between real ones.
- We applied SMOTE **only on the training data** to avoid test leakage.
- This balanced the class distribution and improved model recall on class 1.

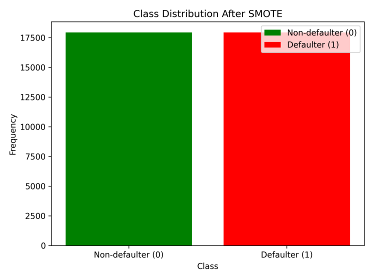
Effect of SMOTE on Class Distribution

SMOTE balanced the training data:



Class distribution **before** SMOTE

$\Rightarrow$



Class distribution **after** SMOTE

Defaulter class (red) is now equally represented in the training set.

What SMOTE Does in Practice

- SMOTE = *Synthetic Minority Oversampling Technique*
- It generates fake (but realistic) defaulter samples
- This pushes the model to pay attention to class 1

Effect on Model Behavior:

- $\uparrow$ **Recall**
- $\downarrow$ **Precision**
- $\uparrow$ **Balanced Accuracy**

Effect of SMOTE on Model Performance

Model	SMOTE	Precision (1)	Recall (1)	F1-score (1)	Balanced Acc.
KNN	No	0.82	0.50	0.62	0.74
KNN	Yes	0.49	0.67	0.56	0.73
Logistic Reg.	No	0.73	0.37	0.49	0.67
Logistic Reg.	Yes	0.49	0.75	0.59	0.77
Poly. Log Reg.	No	0.74	0.54	0.63	0.74
Poly. Log Reg.	Yes	0.66	0.73	0.69	0.81
Decision Tree	No	0.96	0.72	0.82	0.86
Decision Tree	Yes	0.76	0.74	0.75	0.84
Random Forest	No	0.97	0.70	0.81	0.85
Random Forest	Yes	0.77	0.75	0.76	0.84
Bagging Class.	No	0.96	0.60	0.74	0.80
Bagging Class.	Yes	0.72	0.75	0.74	0.83

SMOTE generally improves recall and balanced accuracy for defaulters (class 1), especially in logistic and ensemble models.

SMOTE Trade-Offs by Model

Model	Before SMOTE	After SMOTE	Trade-off Summary
KNN	High Precision, Low Recall	↑ Recall, ↓ Precision	More defaulters caught
Logistic Reg.	Very Low Recall	Major Recall Boost	Safer, more aggressive
Poly. Log Reg.	Balanced	Best overall gain	Top candidate w/ SMOTE
Decision Tree	Already strong	Slight recall gain	Minimal benefit
Random Forest	Balanced	Little change	SMOTE not needed
Bagging Class.	Strong but recall-limited	Small recall boost	Slightly better

Use SMOTE when recall is critical. Tree-based models may not benefit.

Conclusion

Insights and Takeaways

- Preprocessing is **key** — more effort than modeling even on an already pretty clean dataset.
- Random Forest was the best performing model - improving its accuracy was very costly.
- SMOTE helped tackle class imbalance effectively

What's Next?

- Explore more datasets or real-time data
  - Could merge datasets
  - Or use one where advanced data imputation is possible
- Deploy model using Flask or Streamlit
- Try advanced models like `XGBoost` or `CatBoost`

Questions?