UMBC

Loan Default Risk Prediction Using Machine Learning

Course: DATA 602 – Introduction to Data Analysis and Machine Learning

Team Members: Sai Bhargav K, Giridhar Sriram J, Jayanth R.

Term: Spring 2025

Business Problem

- Credit default poses a major challenge to financial institutions
- Goal: Predict whether an applicant is likely to default on a loan.
- Why it matters: Helps lenders manage credit risk and avoid financial losses.

UMBC

Dataset Overview

The dataset used originates from the Home Credit Default Risk competition and comprises detailed application records of over 307,000 customers. It includes 122 features, ranging from numerical to categorical data, encompassing:

- Demographic details (age, gender, family size)
- Financial metrics (income, credit amount, annuity, goods price)
- Credit history (external credit scores EXT_SOURCE_1/2/3)
- Employment information (days employed, occupation type)
- Housing and ownership status

Data Loading

- Loads the training and testing datasets.
- Add a TARGET column to the test set (placeholder) to combine both datasets.
- Combines appl1 and appl2 datasets into one full_df for uniform preprocessing.

```python
# Load application training and test datasets
import pandas as pd

train = pd.read_csv('/content/application_train.csv')
test = pd.read_csv('/content/application_test.csv')

# Confirm structure
print("Train shape:", train.shape)
print("Test shape:", test.shape)

# Add placeholder TARGET column in test to allow merging
test['TARGET'] = np.nan

# Combine for uniform preprocessing
full_df = pd.concat([train, test], axis=0).reset_index(drop=True)

# Preview the top rows
full_df.head()
```

```
Train shape: (307511, 122)
Test shape: (48744, 121)
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	NAME_TYPE_SUITE	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE
0	100002	1.0	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	351000.0	Unaccompanied	Working	Secondary / secondary special
1	100003	0.0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5	1129500.0	Family	State servant	Higher education
2	100004	0.0	Revolving loans	M	Y	Y	0	67500.0	135000.0	6750.0	135000.0	Unaccompanied	Working	Secondary / secondary special
3	100006	0.0	Cash loans	F	N	Y	0	135000.0	312682.5	29686.5	297000.0	Unaccompanied	Working	Secondary / secondary special
4	100007	0.0	Cash loans	M	N	Y	0	121500.0	513000.0	21865.5	513000.0	Unaccompanied	Working	Secondary / secondary special

UMBC

Data Cleaning & Merging

- Flagged unrealistic employment values (DAYS_EMPLOYED = 365243)
- Created EMPLOYED_FLAG binary feature for valid employment status
- Combined EXT_SOURCE_1, 2, and 3 into EXT_SOURCE_MEAN
- Applied Label Encoding and One Hot Encoding to all categorical columns
- Filled missing values using median imputation for robustness

```python
full_df['EMPLOYED_FLAG'] = (full_df['DAYS_EMPLOYED'] != 365243).astype(int)

# Create a new feature combining EXT_SOURCE columns
full_df['EXT_SOURCE_MEAN'] = full_df[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3']].mean(axis=1)

# === B. Encode Categorical Features ===
le = LabelEncoder()
for col in full_df.select_dtypes(include='object').columns:
    full_df[col] = le.fit_transform(full_df[col].astype(str))

# === C. Handle Missing Values ===
imputer = SimpleImputer(strategy='median')
full_df = pd.DataFrame(imputer.fit_transform(full_df), columns=full_df.columns)
```

```python
# === 2. One-hot encode categorical features ===
categorical_cols = X.select_dtypes(include=['object', 'category']).columns.tolist()
# Include 'category' in select_dtypes

if categorical_cols:  # Check if there are any categorical columns
    encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')  # sparse=False
    combined_cats = pd.concat([X[categorical_cols], X_test_final[categorical_cols]], axis=0)
    encoded_combined = pd.DataFrame(encoder.fit_transform(combined_cats),
                                    columns=encoder.get_feature_names_out(categorical_cols))
    encoded_combined.reset_index(drop=True, inplace=True)

    X = X.drop(columns=categorical_cols).reset_index(drop=True)
    X_test_final = X_test_final.drop(columns=categorical_cols).reset_index(drop=True)

    X_combined = pd.concat([X, encoded_combined.iloc[:len(X), :]], axis=1)
    X_test_combined = pd.concat([X_test_final, encoded_combined.iloc[len(X):, :]], axis=1)
else:
    X_combined = X
    X_test_combined = X_test_final
```

Feature Engineering

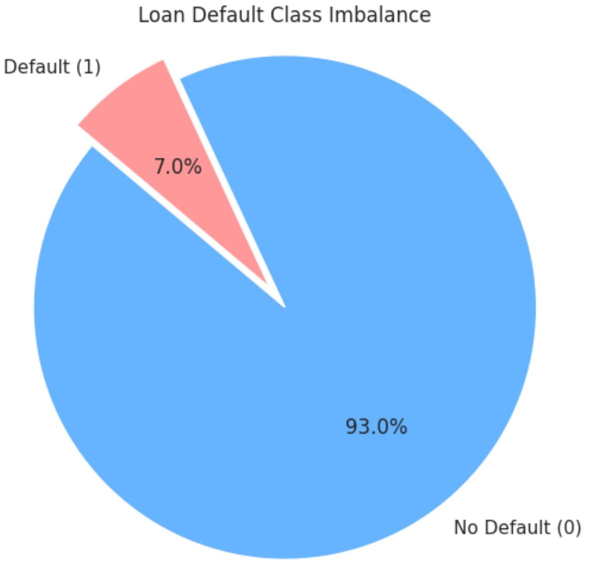
- Dropped low-variance, sparse columns (e.g., FLAG_DOCUMENT_*)
- Combined credit bureau features
- Log-transformed skewed features
- Reduced dimensionality for model input

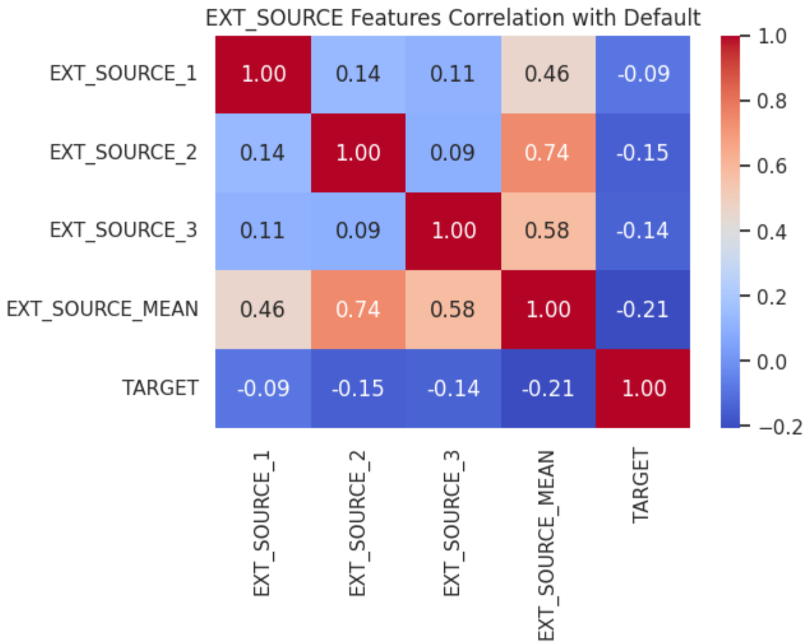
```python
#  Drop sparse FLAG_DOCUMENT_* columns ===
drop_doc_flags = [col for col in full_df.columns if 'FLAG_DOCUMENT' in col]
print(f"Dropping {len(drop_doc_flags)} sparse document flag columns.")
full_df.drop(columns=drop_doc_flags, inplace=True)

# Log-transform AMT_INCOME_TOTAL to handle extreme skew ===
full_df['AMT_INCOME_TOTAL_LOG'] = np.log1p(full_df['AMT_INCOME_TOTAL'])


# capture signal from sparse variable AMT_REQ_CREDIT_BUREAU_QRT
if 'AMT_REQ_CREDIT_BUREAU_QRT' in full_df.columns:
    full_df['CREDIT_BUREAU_FLAG'] = (full_df['AMT_REQ_CREDIT_BUREAU_QRT'] > 0).astype(int)
```

```
Dropping 20 sparse document flag columns.
```

Exploratory Data Analysis

- Identified ~7% default rate

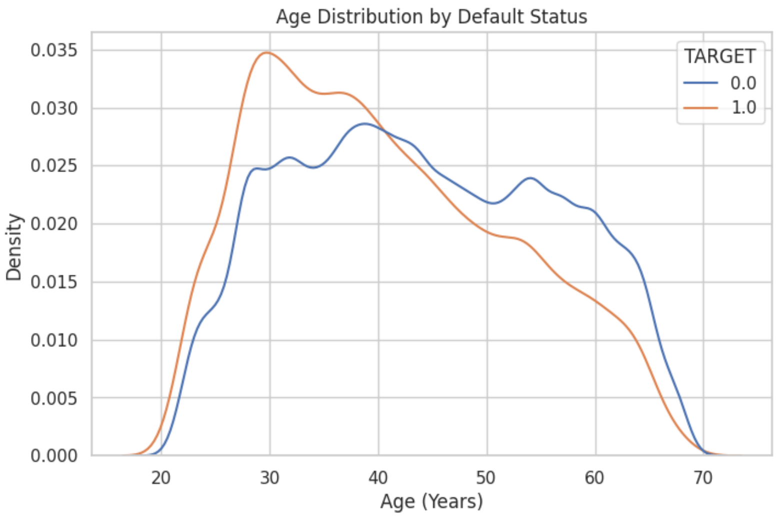


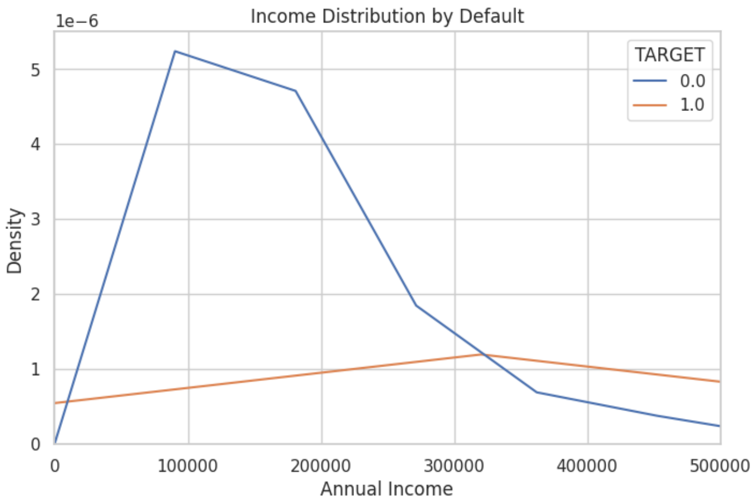


- Used correlation heatmaps and KDE plots

Exploratory Data Analysis

- Age and income affect default risk
- Young and low-income applicants more likely to default





UMBC

Step 5: Skewness Handling

- Detected highly skewed features
- Applied log-transform to income and social circle counts
- Dropped sparse FLAG_DOCUMENT features

```
Top 10 Most Skewed Features:
FLAG_DOCUMENT_12              422.049760
AMT_INCOME_TOTAL              403.649665
FLAG_DOCUMENT_10              225.590337
FLAG_DOCUMENT_2               165.533868
AMT_REQ_CREDIT_BUREAU_QRT     124.313825
FLAG_DOCUMENT_4               108.959932
FLAG_DOCUMENT_7                76.402194
FLAG_DOCUMENT_17               65.890894
FLAG_DOCUMENT_21               58.786187
FLAG_DOCUMENT_20               47.756733
```

```
Top 10 Remaining Skewed Features (Post-Cleaning):
COMMONAREA_MODE                 10.386159
COMMONAREA_AVG                  10.119309
COMMONAREA_MEDI                 10.056945
NONLIVINGAREA_AVG                9.619701
NONLIVINGAREA_MEDI               9.547321
NONLIVINGAREA_MODE               9.537339
DEF_30_CNT_SOCIAL_CIRCLE         8.122912
REG_REGION_NOT_LIVE_REGION       7.805040
LANDAREA_AVG                     7.244071
DEF_60_CNT_SOCIAL_CIRCLE         7.203470
```

Preprocessing Pipeline

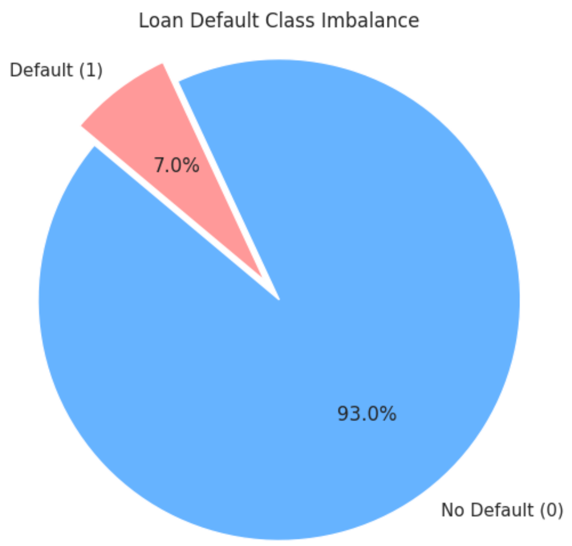
- Split dataset into training and testing sets (X, y)
- Encoded categorical columns using LabelEncoder
- Dropped non-numeric/binned features (e.g., text bins)
- Filled missing values using median imputation
- Scaled features using StandardScaler for uniformity

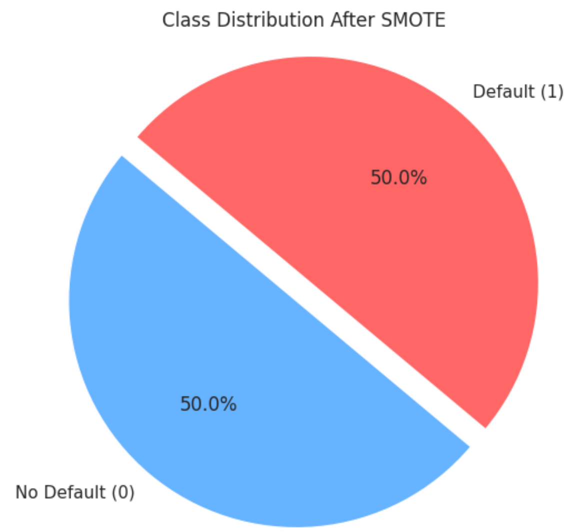
```
# 1. Split full_df into train and test sets based on the original train and test DataFrames
train_df = full_df[full_df['SK_ID_CURR'].isin(train['SK_ID_CURR'])]
test_df = full_df[full_df['SK_ID_CURR'].isin(test['SK_ID_CURR'])].drop(columns=['TARGET'])

#  2. Separate features and target variable ===
X = train_df.drop(columns=['TARGET', 'SK_ID_CURR'])
y = train_df['TARGET']
X_test_final = test_df.drop(columns=['SK_ID_CURR'])

#  3. Encode all remaining object columns (label encoding) ===
object_cols = X.select_dtypes(include=['object']).columns.tolist()
print(f"Encoding {len(object_cols)} categorical columns:", object_cols)

le = LabelEncoder()
for col in object_cols:
    X[col] = le.fit_transform(X[col].astype(str))
    X_test_final[col] = le.transform(X_test_final[col].astype(str))

# 4. Combine X and X_test_final to ensure identical structure ===
X_combined = pd.concat([X, X_test_final], axis=0)
```

```
# 5. Drop non-numeric columns (e.g., binned strings like '20s') ===
X_combined = X_combined.select_dtypes(include=[np.number])

# 6. Re-split the combined data into train and test sets ===
X = X_combined.iloc[:len(X), :].reset_index(drop=True)
X_test_final = X_combined.iloc[len(X):, :].reset_index(drop=True)

#  7. Impute missing values using median strategy ===
imputer = SimpleImputer(strategy='median')
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)
X_test_final = pd.DataFrame(imputer.transform(X_test_final), columns=X_test_final.columns)

# 8. Scale features for models like SVM, LR, KNN ===
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_test_scaled = scaler.transform(X_test_final)
```

Handling Class Imbalance

- Used SMOTE to address class imbalance in the training data.
- Achieved a balanced 50/50 split in the training set.





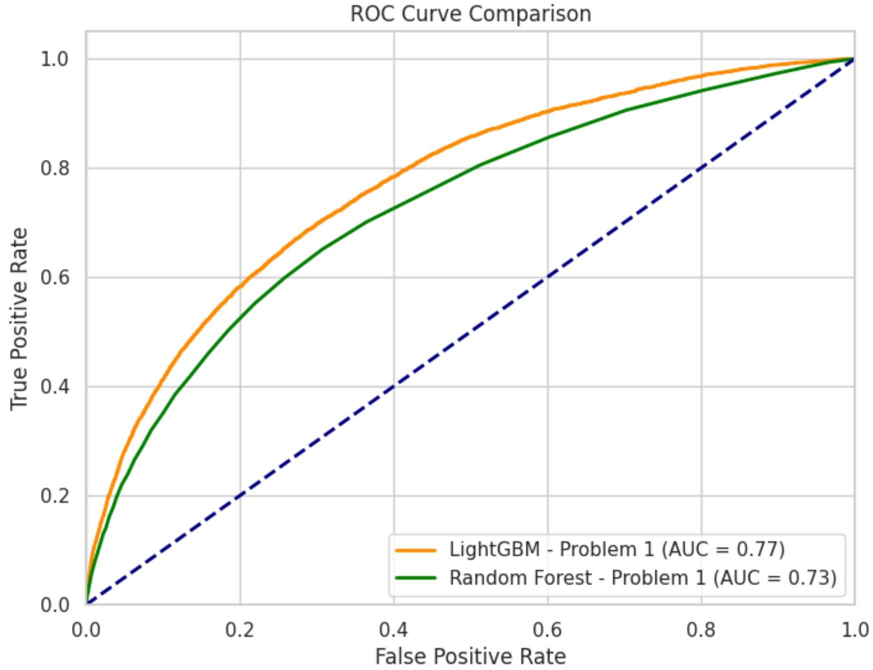
Modeling Strategy

PROBLEM 1- Binary Classification

> Will a customer default or not?

LightGBM with SMOTE Data

LightGBM is trained on SMOTE-balanced data, providing strong AUC performance (0.7707) when evaluating true validation splits, demonstrating effective generalization.



Random Forest on Imbalanced Data

Random Forest is trained using class_weight='balanced' on the original imbalanced data. It achieves an AUC of 0.7236.

Both approaches are compared to highlight the balance versus weighting strategies.

Problem 2: Probability Ranking

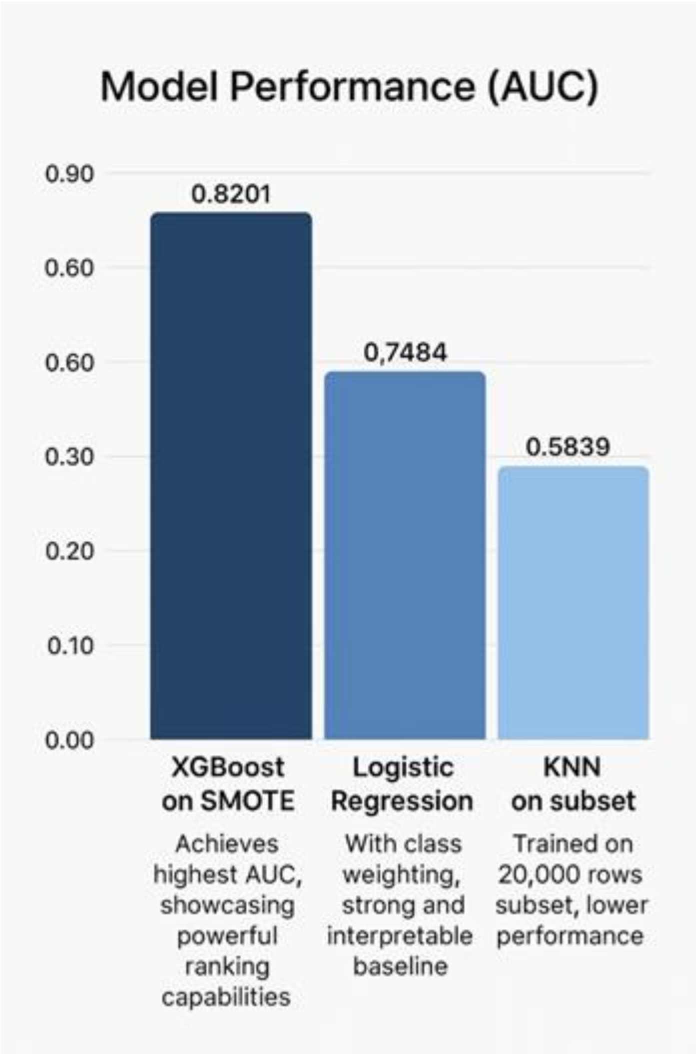
Rank customers by risk to set interest rates and limits

XGBoost and Logistic Regression

XGBoost trained on SMOTE data achieves the highest AUC (0.8201), showcasing its powerful ranking capabilities. Logistic Regression, trained with class weighting and standardization, achieves an AUC of 0.7484, offering a strong and interpretable baseline.

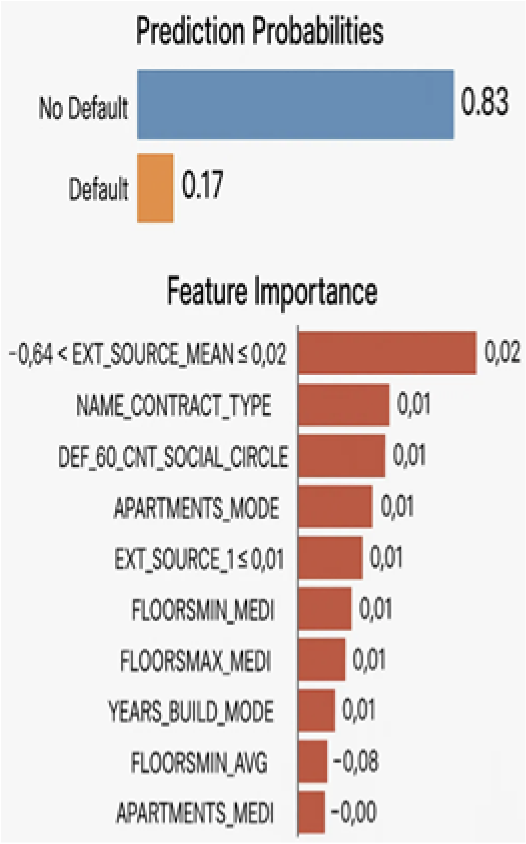
KNN as a Baseline

KNN is only trained on a 20,000-row subset due to computational constraints, achieving a lower AUC (0.5839). This highlights its limitations on large, high-dimensional datasets compared to tree-based models.



PROBLEM 3- MODEL EXPLAINABILITY

Justify loan decisions, comply with Fair Lending laws



Decision Tree & Logistic Regression

A shallow Decision Tree (maxdepth=4) is trained to provide human-interpretable decision paths, and Logistic Regression coefficients can be inspected for feature impact, both achieving reasonable AUCs.

Random Forest with LIME

LIME (Local Interpretable Model-agnostic Explanations) enables local, instance-level feature attribution for Random Forest predictions. LIME visualizes the top 10 features influencing a given result, aiding trust and insight for stakeholders.

UMBC

SUMMARY OF MODEL PERFORMANCE

A cross-model summary table displays AUC scores for all tasks:

Model Comparison Summary

```
import pandas as pd

summary_df = pd.DataFrame({
    "Binary Classification (P1)": results_p1,
    "Probability Ranking (P2)": results_p2,
    "Explainability (P3)": results_p3
}).T

display(summary_df)
```

	LightGBM	Random Forest	XGBoost	Logistic Regression	KNN (subset)	Decision Tree	Random Forest (LIME)
Binary Classification (P1)	0.977835	0.978544	NaN	NaN	NaN	NaN	NaN
Probability Ranking (P2)	NaN	NaN	0.977052	0.765599	0.770714	NaN	NaN
Explainability (P3)	NaN	NaN	NaN	0.765599	NaN	0.885132	0.978544

- Tree-boosting models consistently outperform others for both classification and ranking, but simpler models support greater interpretability.
- Techniques like SMOTE and LIME enhance both model balance and explainability.

Recommendations

- Deploy LightGBM or XGBoost in production pipelines to identify and flag high-risk applicants.
- Integrate explainable models (e.g., Logistic Regression or Decision Trees) for transparency in regulatory environments.
- Consider enriching the dataset with time-based transactional data or credit repayment history to improve accuracy.
- Implement real-time scoring and batch prediction systems for scalability.
- Routinely retrain and monitor models to adapt to evolving customer behavior and economic trends.

THANK YOU