Artificial Intelligence Engineering Program

Project 4 – Build a scoring model

Patrice Preudhomme
06/2024

Table of Contents

1. Context

The financial company 'Prêt à dépenser' offers consumer loans to individuals with little or no credit history.

To grant consumer credit, the company aims to implement a 'credit scoring' tool that calculates the likelihood of a client repaying the loan or not, and then classifies the application: credit granted or denied.

What is expected:

Train a classification model to help decide whether a loan can be granted to a client.

Analyze the factors that justify the model's decision.

2. Dataset Presentation

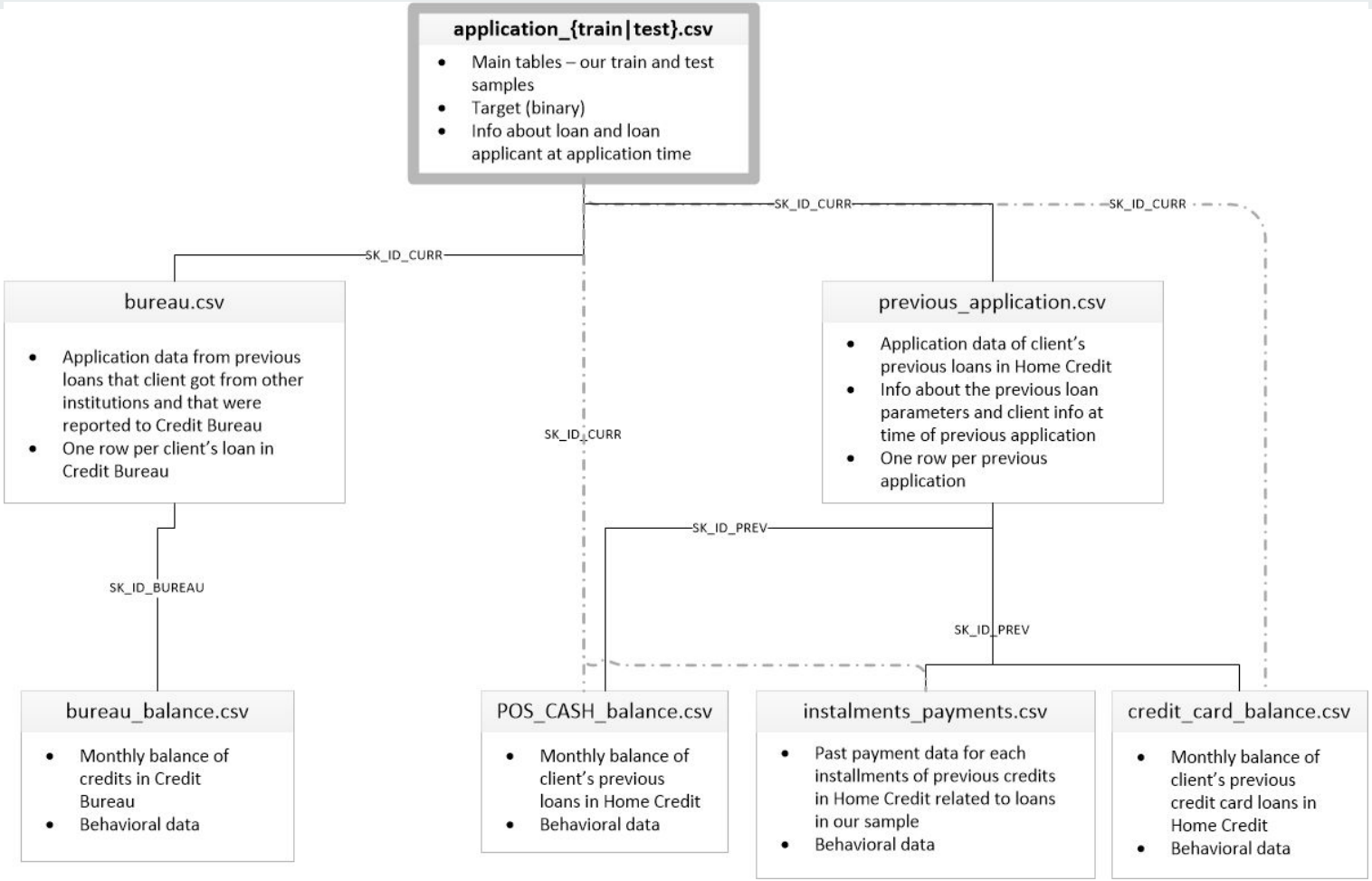
We have at our disposal a dataset that contains:

- A loan history
- A financial information history
- Information on borrower behavior (whether they defaulted or not)

There are several CSV files, and here is a diagram explaining how they are interconnected:

For the `application_train` file: There are a total of 307,511 loans reported, along with 122 features including the 'TARGET' column indicating whether the client has not repaid the loan (TARGET=1) or if they have (TARGET=0).

The `application_test` file contains 48,744 loans, but does not have a 'TARGET' column.


application_{train|test}.csv
Main tables – our train and test samples
Target (binary)
Info about loan and loan applicant at application time
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
bureau.csv
Application data from previous loans that client got from other institutions and that were reported to Credit Bureau
One row per client's loan in Credit Bureau
previous_application.csv
Application data of client's previous loans in Home Credit
Info about the previous loan parameters and client info at time of previous application
One row per previous application
SK_ID_CURR
SK_ID_PREV
SK_ID_BUREAU
SK_ID_PREV
bureau_balance.csv
Monthly balance of credits in Credit Bureau
Behavioral data
POS_CASH_balance.csv
Monthly balance of client's previous loans in Home Credit
Behavioral data
instalments_payments.csv
Past payment data for each installments of previous credits in Home Credit related to loans in our sample
Behavioral data
credit_card_balance.csv
Monthly balance of client's previous credit card loans in Home Credit
Behavioral data

2.Dataset Presentation - Regulatory Requirements in France for Credit Approval - Some features to understand

Regulatory Requirements in France for Credit Approval

Credit Ceilings:

- **AMT_CREDIT**: Monitor that credit requests stay within legal limits.

Repayment Capacity:

- **AMT_INCOME_TOTAL**: Evaluate client's income.
- **AMT_ANNUITY**: Calculate monthly payments of the loan.
- **CNT_CHILDREN & CNT_FAM_MEMBERS**: Include family expenses in repayment capacity assessment.
- **AMT_GOODS_PRICE**: Compare loan amount to the price of financed goods.

Debt-to-Income Ratio:

- **Maintain below 33%.**
- **Use the formula: Debt-to-Income Ratio=AMT_ANNUITY× 12/ AMT_INCOME_TOTAL**

2.Dataset Presentation - Data Availability and Model Readiness

Variable Identification and Categorization:

- Relevant variables for complying with credit ceilings and repayment capacities have been identified and categorized. These are essential for crafting a dependable credit scoring model.

Data Availability:

- All necessary variables are present in the **application_train.csv** file provided by Home Credit.
- Comprehensive data allows for accurate assessment of customer default risk and adherence to regulatory standards.

Project Efficiency:

- Availability of all required data in a single source simplifies the development process.
- No need to merge data from other sources for the **Minimum Viable Product (MVP),** aligning with our project timeline.

3. Exploratory Data Analysis

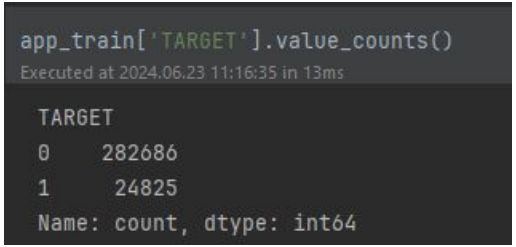
We will focus on the `application_train` file.

The 'TARGET' column: TARGET=1 corresponds to an unpaid loan TARGET=0 corresponds to a repaid loan

This is the value we want to predict with our Machine Learning model, and we can use this file to train it.

This is a supervised classification problem.

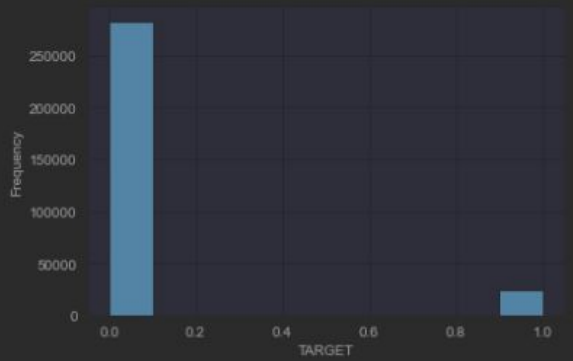
The 'TARGET' variable is imbalanced (about 92% of TARGET=0), and we will need to rebalance the dataset to avoid introducing bias during the training of our model.



3.Exploratory Data Analysis - Imbalance Class Problem

```
app_train['TARGET'].astype(int).plot.hist();
plt.xlabel('TARGET');
```

Executed at 2024.06.23 11:16:35 in 291ms



From this information, we see this is an *imbalanced class problem*. There are far more loans that were repaid on time than loans that were not repaid. Once we get into more sophisticated machine learning models, we can weight the classes by their representation in the data to reflect this imbalance.

3. Exploratory Data Analysis - continued

Missing Values

There are 122 columns:

- 67 columns have missing values
- We will keep all the columns
- However, we will need to impute the missing values before training our supervised models.

Column Types

There are 16 categorical variables, 41 variables are integers, and 65 are floating numbers

```
# Number of each type of column
app_train.dtypes.value_counts()
```

Executed at 2024.06.23 11:16:39 in 14ms

```
float64    65
int64      41
object     16
Name: count, dtype: int64
```

3. Exploratory Data Analysis - continued

Given the analysis, we can derive the following business conclusions regarding the best source for credit scoring predictions:

- **EXT_SOURCE_3** appears to be the most robust indicator, as it shows the greatest difference in median scores between TARGET = 0 and TARGET = 1 and has a broader distribution for TARGET = 0. This suggests that EXT_SOURCE_3 is highly effective at distinguishing between good and bad credit risks.

- **EXT_SOURCE_2** also shows a significant difference in median scores and high variability for TARGET = 0, making it another strong predictor for credit scoring.

- **EXT_SOURCE_1** follows the same trend but with slightly less variability compared to EXT_SOURCE_3 and EXT_SOURCE_2. It is still a valuable predictor, but perhaps not as strong as the other two sources.

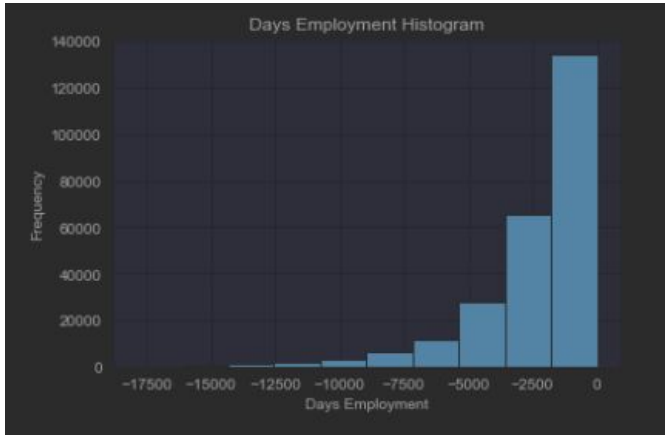
- **DAYS_BIRTH** indicates that older individuals tend to have a TARGET of 0, suggesting that age might also be a useful predictor. However, it might be more effective as a supplementary variable rather than a primary indicator.

4. Data Manipulation

Anomalies :

We analyze anomalies using the describe method. For example, values in DAYS_BIRTH are negative, representing age in days relative to the application; to convert to years, multiply by -1 and divide by 365.

DAYS_EMPLOYED column is analyzed. Anomalies, identified as values exactly equal to 365243, are marked and replaced with NaN to prevent biases in machine learning models. A boolean column is created to track these anomalies, reflecting a systematic approach to data cleaning for improved model accuracy



4. Data Manipulation - Encoding

Categorical Variable Encoding

We will encode categorical variables to convert the data into numerical values, making them interpretable by our Machine Learning models.

Label Encoding

For variables with only two possible values, we will use Label Encoding, converting them into numerical values.

One-Hot Encoding

For variables with more than two possible values, we will employ One-Hot Encoding to avoid creating an "order" among values. This method splits the column into multiple columns (one for each possible value) and assigns a 1 or 0 in these columns, depending on the value each column represents.

5. Feature engineering

We will do a lot of feature engineering when we start using the other data sources, but in this notebook we will try only two simple feature construction methods:

* Polynomial features :

The transformation has expanded the original 4 features into 35 polynomial features. It increases the complexity of the model, so it's important to monitor for overfitting and consider regularization techniques if necessary.

* Domain knowledge features :

To make our dataset more interpretable, we will create new features from the current features :

app_train_domain['CREDIT_INCOME_PERCENT'] = app_train_domain['AMT_CREDIT'] / app_train_domain['AMT_INCOME_TOTAL'] # Create the new feature

app_train_domain['ANNUITY_INCOME_PERCENT'] = app_train_domain['AMT_ANNUITY'] / app_train_domain['AMT_INCOME_TOTAL'] # Create the new feature

app_train_domain['CREDIT_TERM'] = app_train_domain['AMT_ANNUITY'] / app_train_domain['AMT_CREDIT'] # Create the new feature

app_train_domain['DAYS_EMPLOYED_PERCENT'] = app_train_domain['DAYS_EMPLOYED'] / app_train_domain['DAYS_BIRTH'] # Create the new feature

No need to apply label encoding or one-hot encoding for these features as they are already numerical.

6. Data Preparation

Splitting DataFrames into Train/Test Sets on app_train, app_train_poly, and app_train_domain for determining the best model with class imbalance techniques=> app_train_domain

```python
from sklearn.model_selection import train_test_split # Import train_test_split function used to split data

# Function to split data
def split_data(df):
    X = df.drop(columns=['TARGET']) # Drop the target variable
    y = df['TARGET'] # Set the target variable
    return train_test_split(X, y, test_size=0.2, random_state=42, stratify=y) # Split the data into train and test sets with stratification; stratification is used to ensure that the proportion of the target variable is the same in both sets; test_size is used to set the proportion of the test set i.e. 0.2 means 20% of the data is used for testing. random_state is used to set the seed for the random number generator to ensure reproducibility and stratify is used to ensure that the proportion of the target variable is the same in both sets

# Split each DataFrame
X_train_app, X_test_app, y_train_app, y_test_app = split_data(app_train) # Split the original data
X_train_poly, X_test_poly, y_train_poly, y_test_poly = split_data(app_train_poly) # Split the polynomial features data
X_train_domain, X_test_domain, y_train_domain, y_test_domain = split_data(app_train_domain) # Split the domain features data

Executed at 2024.06.23 11:22:56 in 4s 723ms
```

7. Implementing and Evaluating Business-Centric Models for Credit Scoring with Class Imbalance Techniques and Custom Metrics

Evaluating Class Imbalance Techniques with Cost Metric 10 * False Negatives + False Positives and 3 techniques: Under-sampling, Over-sampling, and Class Weighting

The cost metric `10 * False Negatives + False Positives` is a custom evaluation metric designed to prioritize the correct prediction of one class over another, specifically in the context of imbalanced datasets like credit scoring. In this metric:

- False Negatives (FN)**: Instances where the model incorrectly predicts a defaulter as a non-defaulter.

- False Positives (FP)**: Instances where the model incorrectly predicts a non-defaulter as a defaulter.

Why This Metric?

In credit scoring, incorrectly predicting a defaulter as a non-defaulter (false negative) is more costly because it means giving credit to someone who may not repay. This metric penalizes false negatives more heavily to reflect their higher cost.

7. Implementing and Evaluating Business-Centric Models for Credit Scoring with Class Imbalance Techniques and Custom Metrics

```
Evaluating app_train:
Under-sampling Cost Metric: 43877
Over-sampling Cost Metric: 43886
Logistic Regression with Class Weight Cost Metric: 43850
Decision Tree with Class Weight Cost Metric: 46273

Evaluating app_train_poly:
Under-sampling Cost Metric: 49650
Over-sampling Cost Metric: 38452
Logistic Regression with Class Weight Cost Metric: 38455
Decision Tree with Class Weight Cost Metric: 46195

Evaluating app_train_domain:
Under-sampling Cost Metric: 43852
Over-sampling Cost Metric: 43914
Logistic Regression with Class Weight Cost Metric: 43884
Decision Tree with Class Weight Cost Metric: 46008
```

7. Implementing and Evaluating Business-Centric Models for Credit Scoring with Class Imbalance Techniques and Custom Metrics

Choosing the Train Dataset for Credit Scoring

- The `app_domain` dataset has a good performance with the cost metric evaluation for class imbalance techniques.

- **DAYS_BIRTH**, **EXT_SOURCE_1**, **EXT_SOURCE_2**, and **EXT_SOURCE_3** have a significant im**Overall Analysis:**

- The ROC curve and the AUC score of 0.75 indicate that the model has a reasonable ability to distinguish between the positive and negative classes, but there is room for improvement.

- The Precision-Recall curve shows that the model has good precision at lower recall levels, but its performance drops as it attempts to increase recall.

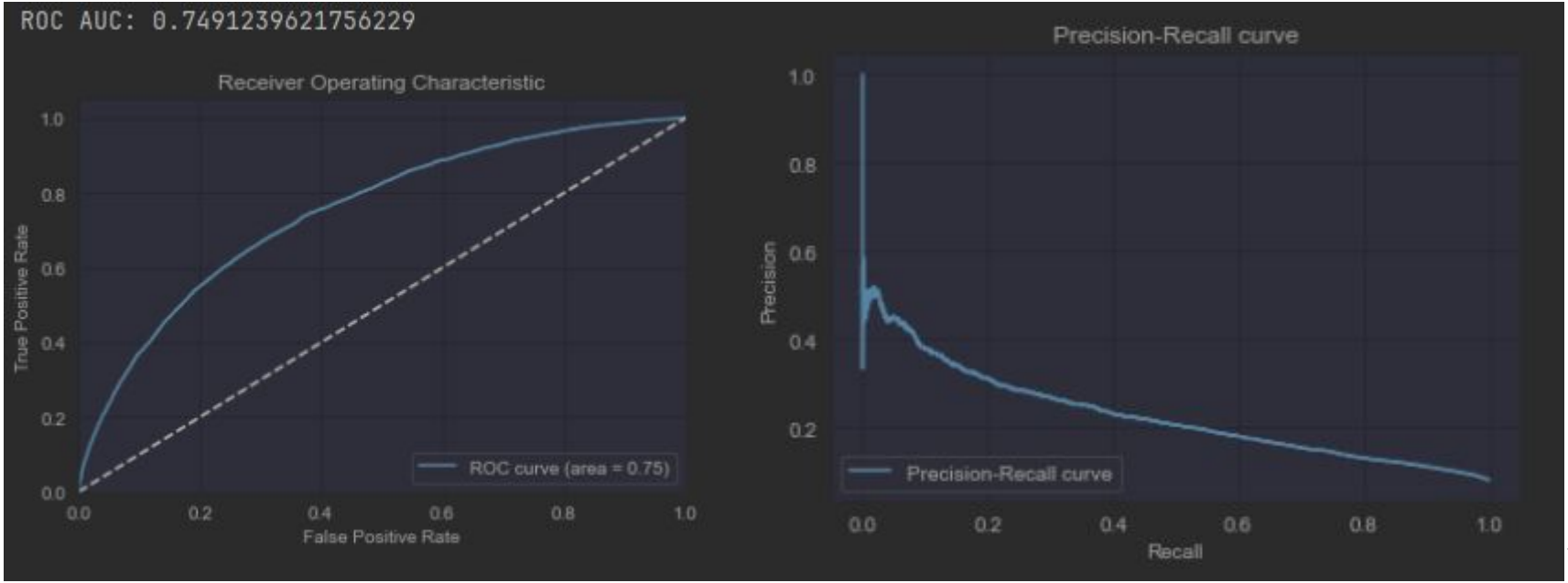
- For imbalanced datasets or cases where positive class identification is critical, focusing on improving both recall and precision while keeping an eye on the ROC AUC can lead to better model performance.

pact on whether a client will repay a loan (the TARGET variable) and are included in the application dataset.

- **CREDIT_INCOME_PERCENT**, **ANNUITY_INCOME_PERCENT**, **CREDIT_TERM**, and **DAYS_EMPLOYED_PERCENT** are domain-specific features that may help predict loan repayment (it is assumed that these features are important based on domain knowledge).

- Therefore, we will use the `app_domain` dataset to train a model to predict whether a client will repay a loan.

8. Baseline Model Training - Logistic regression



8. Baseline Model Training - Logistic Regression

Baseline Model Creation with Logistic Regression :

- Add the custom cost metric to the model evaluation
- Visualize the ROC Curve and Precision-Recall Curve
- Overall Analysis:

  - The ROC curve and the AUC score of 0.75 indicate that the model has a reasonable ability to distinguish between the positive and negative classes, but there is room for improvement.

  - The Precision-Recall curve shows that the model has good precision at lower recall levels, but its performance drops as it attempts to increase recall.

  - For imbalanced datasets or cases where positive class identification is critical, focusing on improving both recall and precision while keeping an eye on the ROC AUC can lead to better model performance.

9. Hyperparameter Tuning via GridSearchCV and Imbalanced-learn Pipeline with make_scorer for Custom Cost Metric with RandomForestClassifier and RidgeClassifier

Custom Cost Metric Function: Implements a custom function to calculate costs based on false negatives and positives, optimizing for minimal financial loss.

Scoring with Custom Metric: Uses this custom cost function as the scoring mechanism in model evaluations, focusing on cost minimization.

Data Preparation: Loads and prepares features (X) and target (y) from app_train_domain.csv for model training.

Resampling Techniques: Integrates SMOTE for oversampling and RandomUnderSampler for undersampling to handle class imbalance in the data.

Model Definition: Employs two types of classifiers, RandomForestClassifier and RidgeClassifier, for robust comparisons.

Robust Scaling: Incorporates RobustScaler within pipelines to standardize features, reducing the impact of outliers.

Hyperparameter Tuning: Applies GridSearchCV to optimize model parameters using the custom cost metric across different pipelines.

Model Evaluation: Uses cross-validation predictions to calculate ROC AUC scores, aiding in selecting the best model based on performance.

Validation Set Assessment: Splits the data to assess model performance on an unseen validation set, providing metrics like ROC AUC, confusion matrix, and a detailed classification report.

Threshold Optimization: Calculates and applies an optimal threshold to maximize the F1 score, enhancing model decision-making and evaluating its impact with the custom cost metric.

9. Hyperparameter Tuning via GridSearchCV and Imbalanced-learn Pipeline with make_scorer for Custom Cost Metric with RandomForestClassifier and RidgeClassifier

Conclusion :

-Random Forest performs best with under-sampling, suggesting that reducing the majority class samples improves model performance.

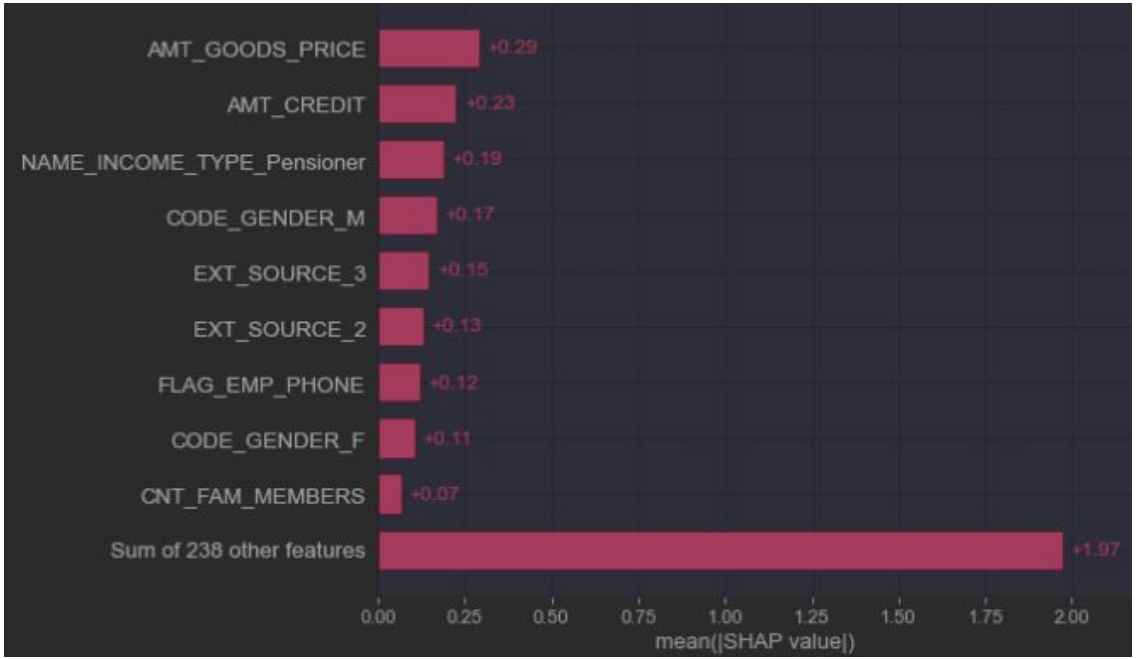
- Ridge Classifier achieves the best results with class weight adjustment, effectively managing class imbalance.

- Prediction Threshold Optimization** improved the handling of the minority class, enhancing precision and recall metrics.

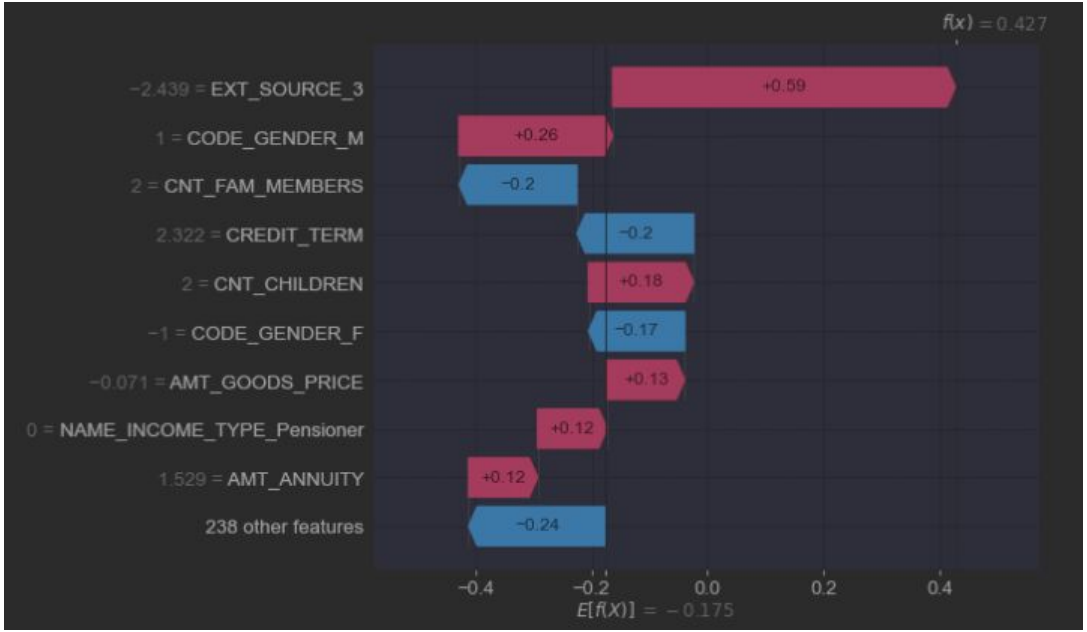
- Custom Cost Metric: The implementation of the custom cost metric function in GridSearchCV has allowed for more targeted optimization, focusing on minimizing the cost of misclassification. However, further improvements are needed to balance the precision and recall for defaulting clients.

- Overall: The Ridge Classifier with class weight adjustment and an optimized prediction threshold is recommended for this dataset due to its superior ROC AUC and balanced performance metrics. Further exploration of hyperparameters and additional models could provide incremental improvements.

10. SHAP Analysis for Feature Importance



10. SHAP Analysis for Feature Importance



11. Global conclusions

- Process in the first notebook is as follows and the objective is to choose the best techniques for preprocessing with cost metric :

Data Loading and Initial Analysis: Load the data, perform initial analysis, and identify areas for improvement.

Outlier Detection and Normality Test: Detect outliers and test the normality of features to guide preprocessing steps.

Cost Metric Definition: Define a custom cost metric to minimize the combined cost of false negatives and false positives.

Choose the best techniques for preprocessing with cost metric.

11. Global conclusions

- Process in the second notebook is as follows:

Data Handling: Loaded data and conducted initial analysis to identify areas for enhancement.

Model Benchmarking: Established a baseline using Logistic Regression and explored further with RandomForestClassifier and RidgeClassifier to set comparative performance benchmarks.

Optimal Model Selection: Applied GridSearchCV for rigorous hyperparameter tuning combined with resampling strategies to select the best models.

Model Evaluation: Evaluated model performance using confusion matrices, providing clear metrics on prediction accuracy.

Predictive Insights: Employed SHAP analysis to understand feature influence on the model's predictions.

Future Directions: Prioritize addressing data imbalance to enhance model performance. Iterative analysis and model refinement recommended for developing an optimal MVP.