

**QRM 2 Assignment:**

**Modern machine learning methods application to real-world data to predict probability of default**

Student: Mai Đình Thảo

ID: 11215386

**Table of Contents**

**I, Introduction 2**

**II, Theoretical background and methodology 3**

**III, Data exploration and modelling 13**

**IV, Results 17**

**V, Conclusion and discussion 23**

**I, Introduction**

**Background of the study:**

Probability of default (PD), one of the three big factors in calculating expected loss for a lender along side exposure at default (EAD) and loss given default (LGD), so far has been determined by traditional qualitative assessment or simple econometric models such as Linear Discriminant Analysis or Logistic Regression.

These models, while are reasonably accurate, rely heavily on certain assumptions as well as limited error handling regarding issues like imbalanced data (default cases are significantly fewer than non-default cases), missing values, and limited model tuning.

The goal of this study is to explore other methods to bypass/alleviate these issues by using modern machine learning methods. The machine learning algorithms chosen are:

**Imbalanced data**: Synthetic Minority Over-sampling Technique, Class Weighting

**Missing values**: K-nearest neighbors imputation

**Classification model**: XGBoost classifier

**Parameters tuning**: Optuna (Bayesian Optimization)

These will be implemented with Python, with the ‘hmeq’ dataset provided by our lecturer. Actual implementation can be found [here](https://github.com/Sawweer/QRM/blob/main/hmeq.ipynb).

**Goal of the study:**

Improving the reliability and fairness of credit risk models by tackling real-world data challenges like imbalanced datasets and missing values.

Enhancing the understanding of how XGBoost can be optimized for credit risk prediction, particularly in environments with challenging data.

Providing practical solutions to parameters tuning besides randomized trial-and-error.

Discuss potential conflicts regarding the use of these ML methods.

**II. Theoretical background and methodology**

**Imbalanced data:**

In many real-world datasets, the number of instances in one class (the minority class - defaults) is much lower than in another class (the majority class – non-defaults). This imbalance can lead to biased models that perform poorly on the minority class.

For a financial institution whose core business activities are borrowing/lending money, customers who are classified as good-creditor while they are in fact bad-creditor (False negatives) will incur the biggest loses to the institution’s loan portfolio. This is because the amount loss will be the entire outstanding loan balance adjusted by Loss Given Default (LGD). On the other hand, False positives (i.e customers classified as bad-creditors but are good-creditors in reality) has a less critical effect on losses on the loan portfolio. This is because the loss will only be the opportunity cost of the money set aside as reserve for such customers.

This paper will introduce two methods to resample so that ratio of positives-negatives are balanced, Synthetic Minority Over-sampling Technique and Class Weighting.

**Synthetic Minority Over-sampling Technique (SMOTE)**

SMOTE generates synthetic examples of the minority class rather than simply duplicating existing instances. Here’s how it works:

**Identify Minority Instances**: SMOTE first identifies all the instances of the minority class in the dataset.

**Select Neighbors**: For each instance in the minority class, SMOTE identifies its k-nearest neighbors (usually k=5-10).

**Generate Synthetic Examples**:

For each minority instance, SMOTE randomly selects one or more of its neighbors.

Synthetic samples are generated by interpolating between the feature values of the minority instance and its selected neighbor(s). Specifically, a synthetic instance is created by taking a weighted average of the features from the minority instance and the neighbor:

**Repeat**: This process continues until the desired balance between the classes is achieved.

Advantages of SMOTE

**Improved Performance**: By providing more training examples for the minority class, SMOTE can lead to better performance in classification tasks.

**Preservation of Information**: Unlike simple oversampling, which duplicates existing instances, SMOTE generates new instances that can help the model learn better decision boundaries.

Disadvantages of SMOTE

**Overfitting**: While SMOTE helps balance the classes, it can also lead to overfitting if too many synthetic instances are created. It's essential to tune the number of synthetic samples generated.

**Class Overlap**: If classes overlap significantly, SMOTE might create ambiguous samples that do not help in training a clear decision boundary.

**Class weighting**

Class weighting aims to adjust the learning process so that the model pays more attention to the minority class. By assigning a higher "weight" to the minority class during training, the model will be penalized more when it makes mistakes on this class. This way, the model learns to balance its predictions better and improves its ability to correctly classify the minority class.

There are many different variations of the formula used to calculate weights of each class, the one used in this paper is:

Where n is the total number of observations, k is the number of classes (2 in this case), and ni is the number of observations in a class. It can be understood that if n1 is small relative to n, the weighting w1 for the positive class will be higher than w0 for the negative class.

These weights can be applied to adjust the loss function during training. A common loss function used for a binary classification problem is log-loss.

Where wyi = w1 > 1 for the minority class, and wyi = w0 < 1 for the majority class. This will add more emphasis on classifying real positives than real negatives.

Advantages of Class Weights

**Improved Model Performance**: By adjusting the influence of different classes, class weights can help improve the model's performance on underrepresented classes, leading to better overall accuracy and F1 scores.

**Flexible**: Class weights can be adjusted easily, allowing practitioners to experiment with different weight configurations to find the optimal balance for their specific problem.

Disadvantages of Class Weights

**Complexity in Tuning**: Selecting the right weights can be challenging and may require extensive experimentation.

**Overfitting Risk**: Assigning high weights to minority classes might cause the model to overfit to those classes, especially if the training data for those classes is limited.

**Increased Training Time**: Models with adjusted class weights may take longer to train, especially if they need to learn more complex relationships due to the increased emphasis on minority classes.

**Interpretability Issues**: The introduction of class weights can make the model's behavior less interpretable, as the impact of each class is modified. This can complicate understanding how decisions are made.

**Missing data:**

Traditionally, the weight of evidence method is often used along side logistic regression. This method works with both categorical variables and continuous variables (when binned) and treats missing data as a separate category. It essentially replaces the original data with a log-odds ratio of how well it classifies the target variable. Mathematically it can be shown as follows:

Where:

pi​: Proportion of good (or positive) outcomes in category i

qi: Proportion of bad (or negative) outcomes in category i

The problem with this method is that it considers all missing values of a variable equally and makes no use of the observable data of the observation with missing value. Other issues arises when incoming new data are not within any of the existing bins/category, or when pi / qi is equal to 0.

With the chosen method of KNN imputation, it uses similar data points (neighbors) to provide information about the missing values. Here’s how the process goes:

**Identify Missing Values**: First, identify which entries in your dataset have missing values.

**Select a Distance Metric**: Choose a method to measure the similarity between data points. Common metrics include Euclidean distance, Manhattan distance, or other distance measures based on the type of data.

**Find Neighbors**: For each missing value, find the K nearest neighbors (data points) that are most similar to the data point with the missing value. The value of K is a parameter you can choose, often based on cross-validation or domain knowledge.

**Impute Missing Values**:

**Numerical Data**: For numerical features, the missing value can be imputed by averaging the values of the K nearest neighbors.

**Categorical Data**: For categorical features, the missing value can be filled by using the most common category among the K nearest neighbors (mode).

Note that these averages can be arithmetic averages or weighted averages where the weights are inversely related to their distance to the missing data point.

**Repeat**: This process is repeated for all missing values in the dataset.

Advantages of KNN Imputation

**Non-parametric**: It does not make strong assumptions about the distribution of data, making it flexible.

**Utilizes Local Information**: It leverages the local structure of the data, often leading to better imputations than simpler methods (like mean/mode imputation).

Disadvantages of KNN Imputation

* **Computationally expensive**: Finding neighbors can be slow for large datasets, especially with a high number of features.
* **Sensitive to Outliers**: Outliers can significantly affect the imputation results.

**PD estimation model:**

Traditionally, calculating PD has been by financial institutions via Logistic Regression, which relied on many assumptions and conditions to function well. Here are some significant ones:

**Linear Relationship**: There should be a linear relationship between the independent variables and the log-odds of the dependent variable. While the relationship between the independent variables and the outcome itself does not need to be linear, the log-odds should show a linear trend with respect to the predictors.

**No Multicollinearity**: The independent variables should not be too highly correlated with each other. High multicollinearity can lead to unstable estimates of coefficients and affect the model's interpretability.

**Absence of Outliers**: The presence of outliers can affect the performance of logistic regression models. Outliers may have a disproportionate effect on the estimated coefficients, leading to misleading conclusions.

**No Perfect Separation**: Logistic regression assumes that there is no perfect separation of the classes by the independent variables. If one class can be perfectly predicted by the independent variables, the model may fail to converge.

**No missing value:** Logistic regression requires all observations to not possess any missing values.

**Class balance (condition)**: The proportion of default cases to non-default cases should be balanced. However, this is not usually the case with real-world data as the number of default cases are much lower than non-default cases. This will make logistic regression very accurate to predict non-default cases but not effective at predicting default cases. False negative default cases are the ones that will incur heavy losses for financial institutions.

The benefits of this approach are:

**Economic interpretation:** Each of the coefficients binded to a variable in the log-odds formula above can have a meaning. The intercept represents a baseline log-odds of outcomes, while the other coefficients represent the change in the log-odds of outcomes for a one-unit increase in the corresponding variable.

**Easy to implement:** The underlying mathematics is not overly complex, which makes it easier to understand and implement. There are well-established algorithms, such as the maximum likelihood estimation (MLE), for estimating the coefficients of the logistic regression model. These algorithms have been thoroughly tested and optimized, making them reliable and efficient.

The drawbacks of this approach most notably are:

**Linearity assumption:** Logistic regression assumes a linear relationship between the independent variables and the log-odds of the dependent variable. If the true relationship is nonlinear, logistic regression may perform poorly unless transformations or interaction terms are included. Logistic regression is a linear model, which means it may struggle to capture complex relationships in the data, particularly in cases where the decision boundary is not linear.

**Outliers sensitivity:** Logistic regression can be sensitive to outliers in the dataset. Outliers can disproportionately influence the estimated coefficients, potentially leading to misleading results.

Due to these critical limitations of the traditional logistic regression model, this paper will explore a popular classification algorithm, XGBoost (Extreme Gradient Boosting).

This is an ensemble methods, which combines the predictions of several base models to improve overall performance. However unlike Random Forest, the “boosting” part here means that the algorithm creates a series of models where each new model will try to correct the errors of its predecessors. XGBoost uses gradient descent to minimize the loss function (the difference between the predicted and actual values). Each tree is added to reduce this loss further. The chosen objective function is often coincidentally the log loss function mentioned above, combined with a regularization term to control for overfitting. The term "gradient" refers to how the algorithm uses the gradients of the loss function to guide the construction of new trees, adjusting weights to emphasize harder-to-predict examples. Mathematically in general, it can be expressed as follows, further details are beyond the scope of this paper:

XGBoost minimizes an objective function, which consists of a loss function (log-loss in our case) and a regularization term R

y: The true labels.

: The predicted values.

f: The ensemble of decision trees.

λ: Regularization parameters.

The loss function often used is the log-loss function, adjusted for positive class weight in our case:

In the boosting process, XGBoost constructs the model iteratively by adding trees. The prediction at iteration t can be expressed as:

Where fk(x) is the k-th tree

For each iteration, XGBoost computes the gradients (first derivative) gi​ and Hessians (second derivative) hi​ of the loss function with respect to the predictions:

XGBoost constructs each tree by finding the optimal splits based on the gradients and Hessians. The gain from a potential split at feature jjj can be defined as:

GL​ and HL​: Sum of gradients and Hessians for the left child.

GR​ and HR​: Sum for the right child.

G and H: Sum for the parent node.

λ: Regularization term for complexity.

γ: Minimum loss reduction required to make a split.

After T iterations (or trees), the prediction for a given input x is:

A threshold is then chosen to decide whether the output is a positive (1) or a negative (0). This threshold is chosen by the user depending on their needs, however there are two popular methods: Youden’s index and Minimum distance.

Youden’s index method choses a threshold where the difference between the true positive rate and false positive rate is the highest on the ROC curve.

Minimum distance method choses a threshold on the ROC curve closest to the perfect model (true positive rate = 1, false positive rate = 0). This distance is computed as the square root of

**Parameters tuning algorithm:**

Traditionally, parameters tuning requires the user to validate and adjust their models manually to improve results on the validation set. This can be exhausting and may not yield the best results.

Optuna is an automatic hyperparameter optimization framework designed to efficiently search for optimal hyperparameter values. This paper uses its efficient search algorithm called Bayesian Optimization. Optuna is designed to minimize or maximize an objective function, which is typically a machine learning model's performance metric (such as accuracy, AUC, etc.). In our context, the performance metric of choice would obviously be **recall** (TP/TP+FN), however **accuracy** will also be used alongside as data will have been resampled with SMOTE. This metric will be validated across multiple fits using what is called **Stratified** **K-Fold cross validation.** It splits the training data into K parts while retaining the original target class distribution, fits using K – 1 parts, computes recall/accuracy on the remaining data, and does so K times.

Bayesian optimization builds a probabilistic model to predict which hyperparameter values might yield the best results and explores the search space accordingly, balancing **exploration** (trying new areas) and **exploitation** (refining known good areas). Optuna includes an important feature called **pruning**, where it stops unpromising trials early based on intermediate results. This helps save computational resources by not fully evaluating poor models.

One important feature of this framework to help with scalability is that it can run in **parallel** (i.e utilizing many CPU/GPU cores at the same time) to save time.

When using XGBoost, these are the parameters that needed to be tuned:

{

        'objective': 'binary:logistic',

        'eval\_metric': 'logloss',

        'booster': 'gbtree',

        'lambda': trial.suggest\_float('lambda', 1e-8, 1.0, log = True),

        'alpha': trial.suggest\_float('alpha', 1e-8, 1.0, log = True),

        'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.3),

        'n\_estimators': trial.suggest\_int('n\_estimators', 50, 500),

        'max\_depth': trial.suggest\_int('max\_depth', 3, 12),

        'min\_child\_weight': trial.suggest\_int('min\_child\_weight', 1, 10),

        'subsample': trial.suggest\_float('subsample', 0.5, 1.0),

        'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.5, 1.0),

        'gamma': trial.suggest\_float('gamma', 0, 1),

'scale\_pos\_weight': w0 / w1

    }

**objective: binary** - Specifies the learning task and objective. Here, binary:logistic means the model is for binary classification outputs predicted probabilities.

**eval\_metric: logloss** – Specifies the loss function to be used. In this case it is the log-loss function mentioned above.

**booster: gbtree** – Specifies the type of algorithm to be used to create trees, in this case we use gradient boosted trees.

**lambda**: lambda is the L2 regularization term on weights, which helps to prevent overfitting.

**alpha**: alpha is the L1 regularization term on weights. Similar to lambda, it prevents overfitting but adds sparsity to the model.

**learning\_rate**: This controls how much the model updates weights after each boosting step. A lower learning rate makes the model more conservative and might require more boosting steps.

**n\_estimators**: Specifies the number of boosting trees.

**max\_depth**: This parameter controls the maximum depth of each tree, where deeper trees can capture more complexity but are more prone to overfitting.

**min\_child\_weight**: This is the minimum sum of instance weights (or the number of samples) needed in a child node. Higher values make the model more conservative, preventing overfitting.

**subsample:** Denotes the fraction of samples used for training each tree. Values less than 1.0 prevent overfitting by adding randomness.

**colsample\_bytree**: This parameter specifies the fraction of features used when building each tree. A smaller value can help with overfitting by introducing more randomness.

**gamma**: Specifies the minimum loss reduction required to make a further partition on a leaf node of the tree. Higher values make the algorithm more conservative by preventing overly complex splits.

**scale\_pos\_weight:** Balancing of positive and negative weights. Usually computed as (# negatives / # positives) or just w0 / w1

Each time there’s a trial.suggest\_xxx, it means Optuna chooses a parameter value from the range listed by the user.

**III, Data exploration and modelling:**

**Exploration**:

“hmeq” contains 5960 rows of data, 1000 of which will be used for testing. The seed to be used is 1111 for replication. This sampling has ensured that the proportion of positives and negatives are reflected in the training set. That means 4960 rows of data will be used to train our model

The data contains the target variable ‘BAD’ (1 for default, 0 for paid) and the following predictor variables:

**MORTDUE**: Amount due on existing mortgage

**VALUE**: (Value of current property)

**REASON**: DebtCon = debt consolidation; Homelmp = home improvement

**JOB**: Occupational categories

**YOJ**: Years at present job

**DEROG**: Number of major derogatory reports

**DELINQ**: Number of delinquent credit lines

**CLAGE**: Age of oldest credit line in months

**NINQ**: Number of recent credit inquiries

**CLNO**: Number of credit lines

**DEBTINC**: Debt-to-income ratio

**gDEBTINC**: Debt-to-income ratio group

Here is the visualization of our variables:

A group of blue and white bars

Description automatically generated

Upon examining the plots, it is clear that the target variable ‘BAD’ is highly imbalanced, with roughly 4 times more negative observations compared to positive observations. Other insights to be made is that both ‘DEROG’ and ‘DELINQ’ are heavily skewed towards 0. However, XGBoost is not affected heavily by skewed predictors, while imbalanced target will pose a problem. SMOTE and class weight will both be used.

In terms of missing values, these are number of missing values for each variable, after mapping categorical variables via one-hot encoding. DEBTINC has more missing values compared to other variables by a wide margin. However, it will be kept and imputed with KNN imputer.

|  |  |
| --- | --- |
| LOAN | 0 |
| MORTDUE | 424 |
| VALUE | 92 |
| YOJ | 437 |
| DEROG | 596 |
| DELINQ | 491 |
| CLAGE | 258 |
| NINQ | 258 |
| CLNO | 192 |
| DEBTINC | 1038 |
| gDEBTINC | 0 |
| REASON\_DebtCon | 0 |
| REASON\_HomeImp | 0 |
| JOB\_Mgr | 0 |
| JOB\_Office | 0 |
| JOB\_Other | 0 |
| JOB\_ProfExe | 0 |
| JOB\_Sales | 0 |
| JOB\_Self | 0 |

**Modelling:** seed 1111

Here are the parameters for imputing with KNN imputer:

‘n\_neighbors’ : 10

‘weights’ : ‘distance’

‘metric’: euclidean distance

Here are the parameters for SMOTE sampling

‘k\_neighbors’ : 10

Here are the choices of parameters for modelling XGBoost

        'lambda': trial.suggest\_float('lambda', 1e-8, 1.0, log = True),

        'alpha': trial.suggest\_float('alpha', 1e-8, 1.0, log = True),

        'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.3),

        'n\_estimators': trial.suggest\_int('n\_estimators', 50, 500),

        'max\_depth': trial.suggest\_int('max\_depth', 3, 12),

        'min\_child\_weight': trial.suggest\_int('min\_child\_weight', 1, 10),

        'subsample': trial.suggest\_float('subsample', 0.5, 1.0),

        'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.5, 1.0),

        'gamma': trial.suggest\_float('gamma', 0, 1)

'scale\_pos\_weight': w1 / w0

n\_estimators and max\_depth are capped to prevent overfitting.

Here are the choices of parameters for setting up Optuna:

‘score’: accuracy, recall

‘K-folds’: 15

‘n\_trials’: 300 (save computational cost)

Here are the choices of computing optimal threshold:

‘Youden’s index’

‘Minimum distance’

**IV, Results**

**Results when using accuracy score:**

A graph of a function

Description automatically generated

A comparison of a classifier

Description automatically generated

**Gini Index**: 0.9390585888242711

Optimal Threshold (**Youden**): 0.07122978568077087

* Accuracy with custom threshold: 0.9
* Precision with custom threshold: 0.6761565836298933
* Recall with custom threshold: 0.9547738693467337

Optimal Threshold (**Min Distance**): 0.19009926915168762

* Accuracy with custom threshold: 0.924
* Precision with custom threshold: 0.7573221757322176
* Recall with custom threshold: 0.9095477386934674

**Optimal parameters:** (excluding fixed parameters)

'lambda': 0.0022976108052347117,

'alpha': 0.014396154321980216,

'learning\_rate': 0.062479492271966565,

'n\_estimators': 370,

'max\_depth': 12,

'min\_child\_weight': 1,

'subsample': 0.5190049705875315,

'colsample\_bytree': 0.7534941706554685,

'gamma': 0.046955934700650695

'scale\_pos\_weight': 4.01010101010101

**Youden’s index** is the better choice here. Only 4-6% of positive observations were incorrectly labeled across multiple runs with different seed values.

**Results when using recall score:**

A graph of a line

Description automatically generated with medium confidence

A comparison of a classifier

Description automatically generated

**Gini Index**: 0.8813794314895325

Optimal Threshold (**Youden**): 0.7075130343437195

* Accuracy with custom threshold: 0.902
* Precision with custom threshold: 0.7112970711297071
* Recall with custom threshold: 0.8542713567839196

Optimal Threshold (**Min Distance**): 0.7075130343437195

* Accuracy with custom threshold: 0.902
* Precision with custom threshold: 0.7112970711297071
* Recall with custom threshold: 0.8542713567839196

**Optimal parameters:** (excluding fixed parameters)

'lambda': 0.00019892250725941362,

'alpha': 9.5386838612203e-06,

'learning\_rate': 0.010866336310585303,

'n\_estimators': 51,

'max\_depth': 8,

'min\_child\_weight': 2,

'subsample': 0.9171624986579654,

'colsample\_bytree': 0.6676561960756537,

'gamma': 0.8244080097154964

Optimizing with recall and class weight seems to be **worse** than optimizing with accuracy and class weight. This paper will move on with **accuracy** as its optimization metric.

**Feature importance:**

While it’s not as clear as coefficients and hypothesis testing as it is the case with classical Logistic Regression, there are still ways to extract feature importance with XGBoost, most notably average information gain (gain-based feature importance) and number of trees a feature is present in (weight-based feature importance). However, it is unclear as to how individual features affect the splitting of nodes without examining tree-by-tree and observation-by-observation.

**Gain-based feature importance**: This measures the average gain of splits using a feature, i.e., the improvement in accuracy or performance brought by a feature. A higher value indicates a feature is important.

**A graph with numbers and a few black lines

Description automatically generated with medium confidence**

**Weight-based feature importance:** This counts the number of times a feature appears in the trees across all boosting rounds. A higher value indicates a feature is important.

**A graph with numbers and lines

Description automatically generated**

It seems that the highest contributing features are different by methods of feature importance, ‘gDEBTINC’ for information gain and ‘CLAGE for weight.

**V, Conclusion and discussion**

The modern machine learning methods mentioned in this paper (SMOTE, class weight, KNN imputer, XGBoost, Optuna) provide an alternative to classical econometric approaches such as Logistic Regression which rely heavily on assumptions and conditions. For complex, non-linear relationships between features and the target variable, these methods are often superior to classical models. However, there are some considerations to be made when using these modern methods, most notably computational cost and regulatory concerns.

**Computational cost:**

Many modern ML models, particularly deep learning architectures, require substantial computational power. This necessitates access to high-performance GPUs or TPUs, which can be expensive to procure and maintain. Training large ML models can take a significant amount of time, which can slow down the development cycle and delay deployment. This is particularly concerning in industries where timely insights are critical. The time needed to run Optuna using accuracy or recall as target metrics, with 300 trials and 15 folds each was around 45 minutes on an 8th generation Intel U series chip running on all 4 cores. This was very slow compared to logistic regression, which completed in less than 1 minute.

**Regulatory concerns:**

These issues stem from the need to balance innovation in predictive modeling with principles of transparency, fairness, and consumer protection. Regulators emphasize the need for financial institutions to use models that are explainable. Complex ML models, especially deep learning or ensemble methods, can act as "black boxes," making it difficult to understand how decisions are made. For credit risk, understanding the factors contributing to a borrower's PD is crucial for ensuring that decisions are fair, lawful, and non-discriminatory.

Regulatory Guidance:

* **Basel II/III** guidelines highlight that credit risk models must be transparent and validated by internal and external auditors.
* The **European Banking Authority (EBA)** emphasizes the need for "explainability" in its machine learning model risk management principles.

There is significant concern that ML models might unintentionally embed biases in their predictions. Machine learning models, when trained on historical data, can inherit biases related to race, gender, or socioeconomic status, potentially leading to discriminatory outcomes in PD calculations. This can violate regulations such as the **Equal Credit Opportunity Act (ECOA)** in the United States.

ML models used for calculating PD must undergo strict governance and validation to ensure their accuracy, stability, and appropriateness. This includes ensuring data quality, model robustness, and ongoing monitoring. Over-reliance on sophisticated models without strong governance increases the risk of model failure, which can undermine an institution's risk management capabilities.