

VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

OeNB ILAB

OeNB ILAB:
Methodology Summary

Author:

Tristan LEITER Anastasia PRYSHCHEPA

Submission Date:

January 14, 2026

Contents

1	Data Processing	2
1.1	Preprocessing and data splitting	2
1.1.1	Preprocessing	2
1.1.2	Data splitting	2
1.2	Feature engineering	3
1.2.1	Standardization (Size Normalization)	3
1.2.2	Quantile Transformation (Probability Integral Transform)	3
2	Modelling	4
2.1	Model Selection	4
2.2	Hyperparameter Tuning	4
2.2.1	Discrete Grid Search	4
2.2.2	Random Grid Search	4
2.2.3	Bayesian Optimization	4
2.3	GLMs	4
2.3.1	Logistic Regression	4
2.3.2	Regularized GLMs	4
2.4	Decision Trees	4
2.4.1	Random Forest	4
2.4.2	AdaBoost	6
2.4.3	Comments on code	7
2.4.4	XGBoost	7
2.4.5	CatBoost	8
2.5	Neural Networks	8
3	Model Assessment	9
3.1	Final Model	9
3.2	Model Evaluation	9
4	Task Distribution	10

Chapter 1

Data Processing

1.1 Preprocessing and data splitting

1.1.1 Preprocessing

Based on the instructions from the client, we filter the balance sheet data based on the following constraints to remove noisy observations. To account for floating-point inconsistencies, we add the condition that the difference must be greater than 2000 (as per client instructions).

1. $f_{10} \geq 0$
2. $(f_2 + f_3) \leq f_1$
3. $(f_4 + f_5) \leq f_3$
4. $(f_6 + f_{11}) \leq f_1$

1.1.2 Data splitting

We utilize a custom function, `MVstratifiedsampling`, to perform a stratified split at the firm level rather than the observation level. This ensures that all records for a specific firm ID reside in the same partition.

1. **Aggregate to Firm Level**

The dataset is first grouped by unique identifier (`id`). We summarize the data to create a single profile per firm, extracting the maximum default status (`y`) and the business sector.

2. **Create Stratification Key**

We generate a combined key for each unique firm by interacting the stratification variables (e.g., `Sector` and `Target`).

- This creates composite levels (e.g., "Energy.1", "Retail.0") used to balance the distribution of the target variable across sectors.

3. **Partition Unique IDs**

Using the `createDataPartition` algorithm from the *caret* package, we split the unique firm IDs based on the stratification key.

- The default split creates a training set containing 70% of the firms and a test set containing the remaining 30%.

4. **Retrieve Observation Data**

Finally, we filter the original dataset to reconstruct the Training and Test sets based on the selected lists of IDs. This preserves the original data structure without requiring column removal.

1.2 Feature engineering

1.2.1 Standardization (Size Normalization)

Financial data often exhibits a "size effect," where absolute magnitude (e.g., total sales or debt in Euros) overshadows financial performance. To ensure comparability between large and small firms while preserving the risk signal associated with company size, we apply a two-step logic:

1. **Ratio Creation:** We scale absolute financial figures (e.g., Debt, Cash, EBIT) against **Total Assets**. This converts raw figures into structural ratios (e.g., $\text{EBIT} \rightarrow \text{ROA}$), isolating efficiency from magnitude.
2. **Size Retention:** We explicitly retain the **Total Assets** feature. In credit risk, size itself is often a predictor of stability (e.g., larger firms having better capital access). By keeping it as a feature, we allow the model to learn these non-linear interactions between "Efficiency" (ratios) and "Scale" (assets).

1.2.2 Quantile Transformation (Probability Integral Transform)

After size normalization, both the financial ratios and the size feature typically remain highly skewed with heavy tails (non-Gaussian). We apply a **Quantile Transformation** (Rank-Gauss) using the Probability Integral Transform (PIT) to all continuous features.

To prevent data leakage and preserve macro-economic signals (e.g., a global downturn increasing leverage ratios across the board), we utilize a **Frozen Reference Approach**:

1. **Training Phase:** We compute the ranks and Empirical Cumulative Distribution Function (ECDF) on the training set to establish a "Through-the-Cycle" reference distribution.
2. **Testing Phase (Walking Forward):** We map future observations (Test set) onto this fixed training ECDF. This ensures that if the economy deteriorates (shifting the test distribution right), the transformed Z-scores reflect this increased risk relative to the training baseline, rather than normalizing it away.

The transformation chain for any continuous variable x (whether a calculated ratio or raw Total Assets) is:

$$x_{final} = \Phi^{-1} (ECDF_{train}(x_{input})) \quad \text{where} \quad x_{final} \sim N(0, 1)$$

Where x_{input} is either the calculated ratio ($\frac{x_{raw}}{\text{Assets}}$) or the raw Size variable itself.

Chapter 2

Modelling

2.1 Model Selection

Selection is done by the AuC-ROC measure as per client instructions.

2.2 Hyperparameter Tuning

2.2.1 Discrete Grid Search

2.2.2 Random Grid Search

2.2.3 Bayesian Optimization

2.3 GLMs

2.3.1 Logistic Regression

2.3.2 Regularized GLMs

2.4 Decision Trees

2.4.1 Random Forest

We use a Random Forest classifier to predict firm default events. The Random Forest combines a large number of classification trees, each trained on a different bootstrap sample of the training data. At each split in a tree, only a random subset of predictors is considered. This randomness reduces correlation across trees and improves predictive performance. Each tree produces a predicted default probability, and final predictions are obtained by aggregating the results across all trees.

Hyperparameter Specification

The Random Forest is estimated with the following hyperparameter settings:

- **Number of trees (B):** The number of trees is set to $B = 500$. A large number of trees stabilizes the ensemble predictions and reduces Monte Carlo noise. Increasing B does not lead to overfitting.
- **Number of predictors per split (m):** At each split, the number of candidate predictors is set to $m = \lfloor \sqrt{p} \rfloor$, where p denotes the total number of predictors. This is the standard choice for classification tasks and helps decorrelate trees.
- **Minimum terminal node size:** The minimum node size is set to 1, allowing trees to grow fully. Individual trees are therefore highly flexible (low bias), while variance is reduced through aggregation across the forest.

Algorithm

Let the training sample be

$$\mathcal{Z} = \{(x_i, y_i)\}_{i=1}^N,$$

where x_i is a vector of firm-level predictors and $y_i \in \{0, 1\}$ indicates default status.

The Random Forest algorithm proceeds as follows:

1. **Bootstrap sampling**

For each tree $b = 1, \dots, B$, a bootstrap sample of size N is drawn with replacement from the training data.

2. **Tree growing**

A classification tree is grown on each bootstrap sample. At each split:

- a random subset of m predictors is selected,
- the best split among these predictors is chosen using the Gini impurity criterion,
- the node is split into two child nodes.

Trees are grown without pruning until the minimum node size is reached.

3. **Prediction aggregation**

For a new observation x , each tree produces a predicted class probability. These probabilities are averaged across trees to obtain the Random Forest predicted default probability.

Model Evaluation

Model performance is evaluated on a separate hold-out test sample obtained via firm-level stratified sampling. For each observation in the test set, the Random Forest produces a predicted default probability.

To obtain class labels, predicted probabilities are converted into default and non-default outcomes using a fixed threshold of 0.5.

Because default events are rare, model performance is primarily assessed using the area under the receiver operating characteristic curve (AUC). The AUC measures the model's ability to correctly rank defaulting firms above non-defaulting firms and is independent of the chosen classification threshold. In addition, confusion matrices are reported to summarize classification outcomes at the 0.5 threshold.

Variable Importance

To assess the contribution of individual predictors, variable importance measures computed internally by the Random Forest algorithm are employed. In particular, we rely primarily on permutation-based importance measures following Breiman (2001), which quantify how strongly predictive performance deteriorates when the information contained in a given predictor is destroyed.

For each predictor X_j , its values are randomly permuted among the out-of-bag (OOB) observations, and the resulting decrease in predictive accuracy is recorded. Let Acc_{OOB} denote the original out-of-bag prediction accuracy and $\text{Acc}_{\text{OOB}}^{(j, \text{perm})}$ the accuracy obtained after permuting X_j . The permutation importance reported in the analysis corresponds to the absolute decrease in accuracy,

$$\text{VI}_j^{\text{abs}} = \text{Acc}_{\text{OOB}} - \text{Acc}_{\text{OOB}}^{(j, \text{perm})}.$$

Larger values of VI_j^{abs} indicate that permuting predictor X_j leads to a substantial deterioration in out-of-bag predictive performance, implying a higher contribution of the predictor to the Random Forest model. Values close to zero suggest that the predictor has little influence on predictive accuracy.

In addition to permutation-based importance, the Random Forest algorithm also reports an impurity-based importance measure, the mean decrease in Gini impurity. This measure aggregates the total reduction in Gini impurity attributable to each predictor across all splits and trees. While the mean decrease in Gini provides insight into how often a variable is used to create purer nodes during tree construction, it is known to be biased toward continuous predictors and predictors with many potential split points.

Breiman (2001) originally expresses permutation importance as a percentage increase in prediction error. The absolute formulation employed here is equivalent up to scaling and preserves the same ranking of predictors. Consequently, permutation-based importance is used as the primary criterion for ranking variables, while Gini-based importance is reported for completeness.

2.4.2 AdaBoost

Model description

Empirical evidence on the performance of AdaBoost in credit-risk applications is mixed. While boosting methods are theoretically appealing, studies such as Zhou et al. (2009) show that standard AdaBoost does not consistently outperform traditional models in highly imbalanced credit-scoring datasets. These findings suggest that AdaBoost can be sensitive to class imbalance and noise, both of which are characteristic features of default prediction data.

Nevertheless, AdaBoost remains a relevant benchmark in credit-risk modelling for several reasons. First, it is a well-established ensemble method that improves predictive performance by sequentially focusing on hard-to-classify observations, which is a central challenge in default prediction. Second, large-scale benchmarking studies (Lessmann et al., 2015) emphasize that no single classifier dominates across datasets, making the inclusion of diverse model classes essential. Third, AdaBoost provides a useful contrast to bagging-based methods such as Random Forests, allowing the analysis to distinguish between bias-reduction mechanisms (boosting) and variance-reduction mechanisms (bagging).

Consistent with the data-splitting strategy described in Section 1.1.2, AdaBoost is trained exclusively on the firm-level training set. The adaptive reweighting of observations is therefore applied only within the training data and does not introduce information leakage across firms.

Algorithm

1. Initialization

All observations receive equal weights:

$$w_i^{(1)} = \frac{1}{N}, \quad i = 1, \dots, N.$$

2. Iterative Boosting

For boosting iteration $m = 1, \dots, M$:

- A weak classifier $h_m(x)$ is fitted using the current observation weights.
- The weighted classification error is computed as

$$\varepsilon_m = \sum_{i=1}^N w_i^{(m)} \mathbf{1}\{h_m(x_i) \neq y_i\}.$$

- The classifier weight is given by

$$\alpha_m = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_m}{\varepsilon_m} \right).$$

3. Weight Update

Observation weights are increased for misclassified observations:

$$w_i^{(m+1)} = w_i^{(m)} \exp(\alpha_m \mathbf{1}\{h_m(x_i) \neq y_i\}),$$

and normalized such that

$$\sum_{i=1}^N w_i^{(m+1)} = 1.$$

4. Final Prediction

The AdaBoost classifier is a weighted combination of all weak learners:

$$H(x) = \text{sign} \left(\sum_{m=1}^M \alpha_m h_m(x) \right).$$

The resulting score is mapped into a default probability, which is used for ranking firms by credit risk.

Logic

In the presence of strong class imbalance, AdaBoost improves default prediction by sequentially increasing the importance of misclassified firms in the training set. Since default events are rare, they tend to be misclassified in early iterations and therefore receive higher weights in subsequent boosting rounds. This mechanism forces later weak learners to focus on economically fragile and borderline firms. Combined with firm-level stratified sampling, AdaBoost concentrates model capacity on difficult default cases while preserving a clean out-of-sample evaluation on unseen firms.

2.4.3 Comments on code

Firm-Level Data Leakage Prevention

Given the panel structure of the data, multiple observations may exist for the same firm across time. To ensure a valid out-of-sample evaluation, the data are split at the firm level, such that all observations of a given firm appear exclusively in either the training or the test set.

A leakage check is performed prior to model estimation to verify that no firm identifier appears in both sample. Formally, letting $\mathcal{I}_{\text{train}}$ and $\mathcal{I}_{\text{test}}$ denote the sets of firm identifiers in the training and test data, respectively, the following condition is enforced:

$$\mathcal{I}_{\text{train}} \cap \mathcal{I}_{\text{test}} = \emptyset.$$

Handling via Observation Weight

The training data exhibits a strong class imbalance, with default events being rare relative to non-default observations. To address class imbalance, we assign observation-level weights based on inverse class frequencies:

Let N denote the total number of training observations, and let N_+ and N_- denote the number of default and non-default observations, respectively. Initial observation weights are defined as

$$w_i = \begin{cases} \frac{N}{2N_+}, & \text{if } y_i = 1, \\ \frac{N}{2N_-}, & \text{if } y_i = 0. \end{cases}$$

This weighting scheme ensures that defaults and non-defaults contribute equally to the estimation objective, preventing the boosting algorithm from being dominated by the majority class in early iterations. Within the AdaBoost framework, initial observation weights enter directly into the weighted exponential loss function, implying that reweighting observations is equivalent to introducing asymmetric misclassification costs. This approach is well established in the literature on cost-sensitive and imbalanced classification (Friedman et al., 2000; Elkan, 2001; He and Garcia, 2009) and is commonly applied in credit-risk modeling where default events are rare but economically critical (Baesens et al., 2003; Lessmann et al. (2015)).

2.4.4 XGBoost

```
# Start date for data
start_date <- as.Date("2024-12-31")
```

Model Description

XGBoost (Extreme Gradient Boosting) is a scalable and highly efficient implementation of gradient boosted decision trees (GBDT). Unlike traditional Gradient Boosting Machines (GBM) that use first-order derivatives, XGBoost minimizes a regularized objective function using a second-order Taylor expansion, incorporating both the gradient (first derivative) and the hessian (second derivative). This allows for a more precise estimation of the loss reduction and faster convergence.

Algorithm

The XGBoost algorithm builds an ensemble of classification trees in an additive manner. At each iteration t , a new tree f_t is added to the model to minimize the objective function.

Objective Function The objective function at iteration t consists of a loss term and a regularization term:

$$\mathcal{L}^{(t)} = \sum_{i=1}^n l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) \quad (2.1)$$

where l is the internal optimization metric, typically ****Binary Logistic Loss (LogLoss)****, which measures the difference between the predicted probability and the true label y_i .

Taylor Expansion To efficiently optimize this objective, XGBoost employs a second-order Taylor expansion of the loss function:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^n \left[l(y_i, \hat{y}^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \quad (2.2)$$

where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ is the gradient (first derivative) and $h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$ is the hessian (second derivative).

Regularization The regularization term $\Omega(f_t)$ controls the complexity of the tree to prevent overfitting:

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \|w\|^2 \quad (2.3)$$

Here, T is the number of leaves, and w represents the leaf weights. γ penalizes the addition of new leaves (pruning), while λ provides L2 smoothing of the leaf weights.

Hyperparameter Specifications

We optimize the following hyperparameters to control the bias-variance trade-off:

- **Learning Rate (η):** Controls the step size shrinkage to prevent overfitting. Lower values require more trees but improve generalization.
- **Max Depth:** The maximum depth of a tree. Deeper trees capture more complex interactions but risk overfitting specific samples.
- **Subsample:** The fraction of observations randomly sampled for each tree. Values < 1.0 introduce bagging-like randomness.
- **Colsample_bytree:** The fraction of features randomly sampled for each tree. This decorrelates trees, similar to Random Forest.

Tuning Strategies

To maximize the **AUC (Area Under the Curve)** on the validation set, we compare three hyperparameter optimization strategies:

1. **Discrete Grid Search:** A baseline that evaluates every combination of a fixed set of values. It is computationally expensive and limited by the pre-defined grid.
2. **Random Search:** Samples hyperparameters from defined distributions. It is often more efficient than grid search for high-dimensional spaces as it explores a wider variety of values for continuous parameters like η .
3. **Bayesian Optimization (TPE):** We employ the Tree-structured Parzen Estimator (TPE). Unlike random search, TPE builds a probabilistic model of the objective function (AUC) based on past evaluations to intelligently select the next hyperparameters. ? demonstrate that TPE significantly improves stability and performance in credit scoring.

Potential Improvements

A critical challenge in credit risk modeling is the severe class imbalance (defaults are rare). To address this, we propose an enhancement to the standard objective function.

Following the methodology of Chang et al. (2018), we can calculate the *imbalance ratio* (the ratio of non-default to default observations) and assign it to the `scale_pos_weight` parameter. This modifies the gradient calculation to penalize false negatives (missed defaults) more heavily than false positives. While our primary model uses standard LogLoss, implementing this weighted loss function is expected to improve the AUC by correcting the model's bias towards the majority class.

2.4.5 CatBoost

2.5 Neural Networks

Chapter 3

Model Assessment

3.1 Final Model

3.2 Model Evaluation

Chapter 4

Task Distribution

The following matrix outlines the distribution of project responsibilities among the four team members. Primary ownership is denoted by an **X**.

Table 4.1: Team Task Distribution Matrix

Task	Implemented	Tristan	Nastia	Leonid	Martin
Data Preprocessing	Yes	X	X		
Feature Engineering (Standardization & PIT)	Yes	X			
Stratified Sampling within Train Set	No				X
GLMs and regularized GLMs	No				
Random Forest and Boosting	No	X	X		
Neural Networks	No				

Bibliography

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Yung-Chia Chang, Kuei-Hu Chang, and Guan-Jhih Wu. Application of extreme gradient boosting trees in the construction of credit risk assessment models for financial institutions. *Applied Soft Computing*, 73:914–920, 2018.

Stefan Lessmann, Bart Baesens, Hsin-Vonn Seow, and Lyn C Thomas. Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research. *European Journal of Operational Research*, 247(1):124–136, 2015.

Ligang Zhou, Kin Keung Lai, and Jerome Yen. Credit scoring models with auc maximization based on weighted svm. *International Journal of Information Technology & Decision Making*, 8(04):677–696, 2009.