Vienna University of Economics and Business

OeNB ILAB

OeNB ILAB:
Methodology Summary

Author:
Tristan Leiter

Submission Date:
December 26, 2025

Contents

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., EBIT $\rightarrow$ 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:

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

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 $b = 1, \ldots, B$, 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

2.4.3 XGBoost

2.4.4 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				