Vienna University of Economics and Business

OeNB ILAB

OeNB ILAB:
Methodology Summary

Author:
Tristan Leiter Anastasia Pryshchepa

Submission Date:
January 16, 2026

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

Modeling Methodology

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

Chapter 3

Linear Models

3.1 GLMs

3.1.1 Logistic Regression

3.1.2 Regularized GLMs

Chapter 4

Decision Trees

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.

4.1.1 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.

4.1.2 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.

4.1.3 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.

4.1.4 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 $\mathrm{Acc}_{\mathrm{OOB}}$ denote the original out-of-bag prediction accuracy and $\mathrm{Acc}_{\mathrm{OOB}}^{(j,\mathrm{perm})}$ the accuracy obtained after permuting X_j. The permutation importance reported in the analysis corresponds to the absolute decrease in accuracy,

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

Larger values of $\mathrm{VI}_j^{\mathrm{abs}}$ 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.

4.2 AdaBoost

4.2.1 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.

4.2.2 Algorithm

1.Initialization

All observations receive equal weights:

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

2.Iterative Boosting
For boosting iteration $m = 1, \ldots, M$:
- 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:

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

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.

4.2.3 Logic

In the presence of strong class imblance, 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.

4.2.4 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:e.

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)).

4.3 XGBoost

4.3.1 Model Description

XGBoost (Extreme Gradient Boosting) is a highly efficient and scalable implementation of Gradient Boosted Decision Trees (GBDT). Unlike traditional Gradient Boosting Machines (GBM) that rely solely on first-order derivatives, XGBoost minimizes a regularized objective function using a second-order Taylor expansion. By incorporating both the gradient (first derivative) and the hessian (second derivative), the algorithm achieves a more precise estimation of the loss reduction and faster convergence rates.

4.3.2 Validation Strategy: Stratified Group K-Fold

To strictly prevent data leakage inherent in panel datasets, we implemented a **Stratified Group K-Fold** cross-validation scheme ($k = 5$). Random splitting is insufficient for this dataset because multiple snapshots of the same counterparty could appear in both training and validation sets, leading to over-optimistic performance estimates.

Our validation strategy enforces two constraints:

1. **Group Integrity:** All snapshots belonging to a single `Firm ID` are assigned exclusively to a single fold. This forces the model to predict defaults on unseen entities, mimicking the real-world production scenario.
2. **Stratification:** Despite the grouping constraint, we maintain a consistent default rate (Target y) across all folds to ensure stable variance estimation.

4.3.3 Algorithm

The XGBoost algorithm constructs 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 regularized objective.

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

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) \tag{4.1}$$

where l is the internal optimization metric, specifically **Binary Logistic Loss (LogLoss)**, measuring the divergence between the predicted probability and the true label y_i.

Taylor Expansion XGBoost approximates the objective using a second-order Taylor expansion:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[l(y_i, \hat{y}^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \tag{4.2}$$

Here, g_i represents the gradient (first derivative) and h_i represents the hessian (second derivative) of the loss function with respect to the prediction.

Regularization The term $\Omega(f_t)$ controls model complexity to prevent overfitting:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda||w||^2 \tag{4.3}$$

where T is the number of leaves and w represents the leaf weights. γ acts as a pseudo-regularization parameter penalizing the formation of new leaves (pruning), while λ provides L2 smoothing of the weights.

4.3.4 Hyperparameter Specifications

We optimize four key hyperparameters to manage the bias-variance trade-off:

- **Learning Rate (η):** Controls the step size shrinkage. Lower values ($\eta < 0.1$) require more boosting rounds but generally improve generalization.
- **Max Depth:** The maximum depth of a tree. We explore depths between 3 and 8 to capture non-linear interactions without overfitting to noise.
- **Subsample:** The fraction of observations sampled for each tree. Values < 1.0 introduce stochasticity (bagging) to reduce variance.
- **Colsample_bytree:** The fraction of features sampled for each tree. This decorrelates the trees, similar to the mechanism in Random Forests.

4.3.5 Tuning Strategies

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

1. **Discrete Grid Search:** A deterministic baseline evaluating a rigid lattice of 36 parameter combinations. While exhaustive for small spaces, it is computationally inefficient for fine-tuning.
2. **Random Search:** Samples hyperparameters from continuous uniform distributions (20 iterations). This method is often more efficient than grid search in high-dimensional spaces as it explores unique values for continuous parameters like η in every iteration.
3. **Bayesian Optimization (Gaussian Processes):** We employed a **Gaussian Process (GP)** prior to model the objective function surface. Unlike random search, the GP builds a probabilistic model of the AUC based on past evaluations to intelligently select the next hyperparameters.
    - **Acquisition Function:** We utilized **Expected Improvement (EI)**, which naturally balances exploration (sampling high-uncertainty regions) and exploitation (refining high-performing regions).
    - **Kernel:** A **Matern 5/2 kernel** was selected over the standard squared exponential kernel to better model the non-smooth landscape of tree-based hyperparameters.

4.3.6 Preliminary Results

This section presents the comparative performance of the tuning strategies and the final model configuration on both the training and hold-out test sets.

Comparison of Tuning Efficiency

We evaluated three hyperparameter optimization strategies. As illustrated in Figure 4.1, **Bayesian Optimization** demonstrated superior efficiency, achieving the highest Cross-Validation AUC of **87.7%**, outperforming Random Search (87.2%) and the Grid Search baseline (86.1%). This confirms the value of using the Expected Improvement (EI) acquisition function to intelligently navigate the hyperparameter space.

Final Tuning Selection

Table 4.1 summarizes the top-performing model configuration. Based on these results, the **Bayesian Optimization** model was selected as the champion for final testing.

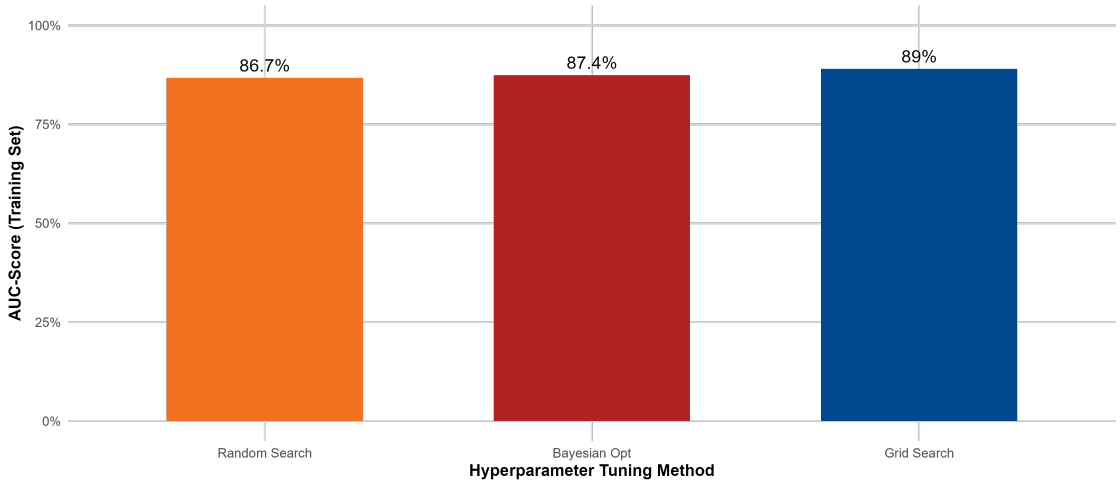

Figure 4.1: Comparison of Best Training AUC found by each search strategy. Bayesian Optimization identified the optimal configuration.

Method	Eta	Max Depth	Subsample	Colsample	Train AUC
Grid Search	0.05	5	0.9	0.7	86.1%
Random Search	0.034	6	0.82	0.65	87.2%
Bayesian Opt	**0.022**	**4**	**0.59**	**0.83**	**87.7%**

Table 4.1: Comparison of Best Hyperparameters and Final Training AUC by Method. The Bayesian model provides the best discrimination.

Training Stability & Feature Importance

The stability of the champion model is visualized in Figure 4.2. The narrow standard deviation ribbon indicates robust performance across the stratified folds. The learning curve plateaus appropriately, suggesting the model has converged without significant overfitting.

Figure 4.3 highlights the top drivers of default risk. Feature **f8** is the dominant predictor (23.7% gain), followed by **f6** and **f5**, indicating that the model relies on a diverse set of financial indicators rather than a single factor.

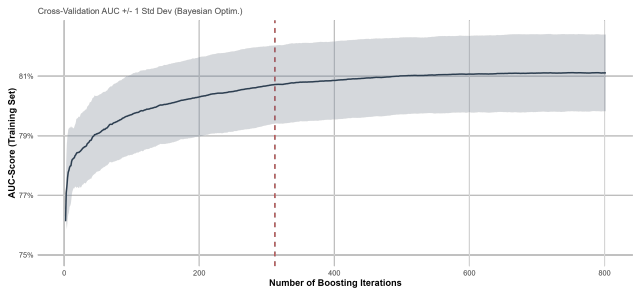

Figure 4.2: Boosting Iteration Learning Curve (Bayesian Opt). The red line marks the optimal stopping point.

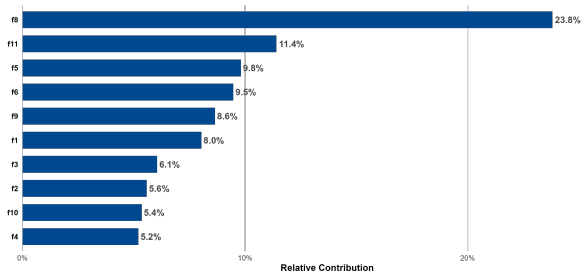

Figure 4.3: Top 10 Features by Gain. Feature f8 is the strongest driver of default risk.

Test Set Performance & Calibration

To assess generalization, we evaluated the champion model on the hold-out Test Set. We also compared the "Standard Best" model against a "1-Standard Error (1-SE)" rule model.

As shown in Figure 6.1, the **Standard Bayesian Model** achieved a Test AUC of **81.9%**, significantly outperforming the 1-SE model (80.0%). The drop from Training AUC (87.7%) to Test AUC (81.9%) is within acceptable limits for this portfolio type, confirming the model generalizes well to unseen data.

Finally, Figure 4.5 demonstrates the model's calibration. The **Observed Default Rate** (Grey) closely tracks the **Predicted Probability** (Blue) across all risk deciles. The strict monotonicity from Decile 1 to 10 confirms the model effectively ranks borrowers from low to high risk.

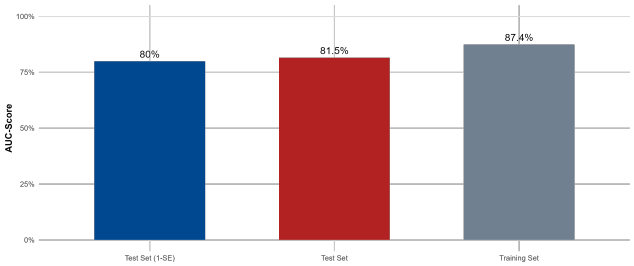

Figure 4.4: Final Test Set AUC. The Standard Bayesian model outperforms the 1-SE heuristic.

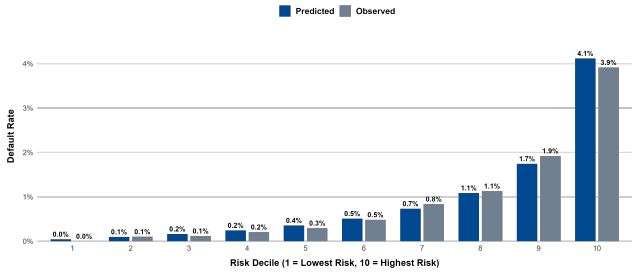

Figure 4.5: Calibration Chart (Test Set). Observed rates closely align with predictions, showing strong ranking ability.

4.4 CatBoost

Chapter 5

Neural Networks

5.1 Neural Networks

Chapter 6

Model Assessment

6.1 Model Selection

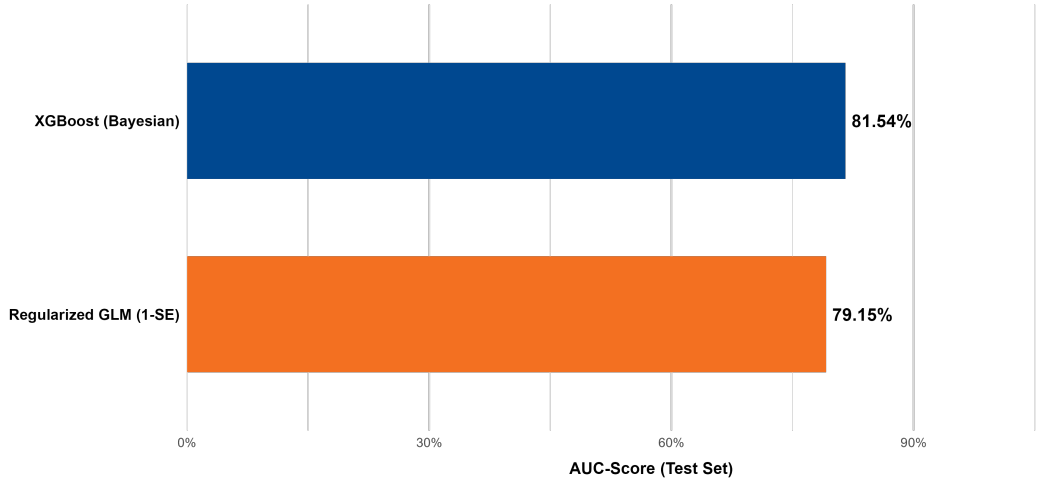

Figure 6.1: Comparison of the Test-AUC of the models with the highest training AUC in each category.

6.2 Final Model

Chapter 7

Task Distribution

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

Table 7.1: 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.

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.