Final Report Rubric

Team Tritons
Akshat Muir, Michael Garcia-Perez, Jasmine Lo, Asif Mahdin, Aritra Das

Problem Statement

Every year, multiple people apply for loans to be able to afford a home. Given this necessity, banks must find a way to decipher which people are more or less likely to default on a loan if their application is approved. Traditionally, lenders look at credit score to determine an applicant's decision, but what happens when someone doesn't have a credit score? We would have to look at other factors of their spending, income, and saving. Therefore, a model must be made, and an accurate one, to be able to help companies determine who is eligible for a home loan or not. The dataset we were working with was very large, with around 32 gigabytes of data, making the training process very long. Furthermore, this data consisted of various financial information regarding people's past. Some examples include, but are not limited to, contracts, loans, failed payments, amount of late payments, account balance, etc. These types of data give insight to a person's past financial behaviors and could be used to predict whether they will default on their home loans or not. One way to evaluate how well our model is performing is to split our data up into train and validation sets in order to ensure that our results will be similar in our test set. Another direct way of measuring the model's performance is through a Gini score. A gini score estimates a model's performance through taking the ratio of the area under the ROC curve plus that of a perfectly linear line through the middle of the graph between the axes of the True positive rate and the False positive rate. The reason there is a line in the middle is to represent how a model would perform if it were to spit out random guesses at classifying, making it a 50-50 chance. The ROC curve, if it bends above the linear line, will reflect a Gini score that is above 0.5. If it is below the linear line, that will reflect a score that is below 0.5. If the model is a 100% perfect classifier then the Gini score will be a 1. It is through this information that we're able to build various models that utilize past financial transactions to estimate who would be a good fit for a home loan and who wouldn't. In the future, loaning agencies can use such models to help predict whether a customer would fail to pay his home loan or not, instead of relying on something as seemingly arbitrary as a credit score.

Potential Strategies and Approach

1. **Gradient Boosting Models**:
    - **XGBoost**: Already used with good results. Known for handling missing values and capturing complex patterns.
    - **LightGBM**: Another gradient boosting framework that is faster and more efficient with large datasets.
    - **CatBoost**: Specifically designed to handle categorical features, which could be beneficial if we decide to include categorical data in the future.
2. **Ensemble Methods**:
    - **Stacking**: Combining multiple models (e.g., XGBoost, LightGBM, CatBoost) to leverage their individual strengths.
    - **Blending**: A simpler approach than stacking, where the final predictions are a weighted average of multiple models.
3. **Regularization Techniques**:

    - **Elastic Net Regularization**: A combination of L1 and L2 regularization that could be applied to linear models to handle multicollinearity and feature selection.
    - **Feature Selection**: Using techniques like Recursive Feature Elimination (RFE) or Feature Importance from tree-based models to select the most relevant features.
4. **Feature Engineering**:
    - **Interaction Features**: Creating new features that capture interactions between existing features.
    - **Aggregation of Historical Data**: Summarizing historical data into meaningful statistics (e.g., mean, median, standard deviation) to capture trends over time.
    - **Missing Value Indicators**: Creating binary indicators for missing values, allowing models to learn patterns associated with missingness.

Adaptation for the Specific Problem

1. **Handling Missing Values**:
    - Using models like XGBoost, LightGBM, and CatBoost, which inherently handle missing values, allows for the inclusion of more features without extensive preprocessing.
2. **Focus on Stability**:
    - Incorporating stability directly into the training process by using cross-validation strategies that respect the temporal nature of the data and using L1 and L2 regularization to avoid overfitting.

Key Findings in Model Development

Initial Approach: Logistic Regression

What We Tried:

- **Model**: Logistic Regression
- **Features**: 20 manually selected columns deemed important
- **Handling Nulls**: Imputed missing values with 0 for numerical columns and the mode for categorical columns

Why We Tried It:

- Logistic regression is a simple, interpretable model and often serves as a good baseline.

Outcome:

- **Gini Score**: 0.08 (private score)
- The model performed poorly, likely due to its inability to handle the complexity of the data and the suboptimal handling of missing values.

Evidence:

- The low Gini score indicates poor discrimination between defaulters and non-defaulters.

Experiment: Logistic Regression Column Inputs

What We Tried:

- Tested different subsets of columns to see which were important and could increase our stability and which would actually decrease stability.

Why We Tried It:

- Because the overhead of training the logistic regression model is so high, we could only train a subset of columns at once. Due to this limitation, we wanted to choose the best subset possible to optimize our score.

Outcome:

- **Gini Score**: 0.22 on test subset created from training dataset
- The model performed slightly better than our original baseline, with scores oscillating between 0.18 (for testing which columns were irrelevant) and 0.22 (to see which columns, when added, improved stability).

Evidence:

- The range in Gini scores we saw showed that there were indeed columns that could make the model more stable, and on the other hand, columns that could make the model less stable.

Transition to XGBoost

What We Tried:

- **Model**: XGBoost
- **Features**: 400 continuous columns
- **Handling Nulls**: Utilized XGBoost's inherent ability to handle missing values

Why We Tried It:

- XGBoost is known for its ability to handle large datasets with many features, including those with missing values. It also captures complex, non-linear relationships.

Outcome:

- **Gini Score**: 0.54 (private score)
- Significant improvement in performance due to the model's ability to utilize more features and effectively handle missing values.

Evidence:

- The substantial increase in Gini score from 0.08 to 0.54 demonstrates the model's enhanced capability in predicting loan defaults.

Hyperparameter Tuning

What We Tried:

- **Hyperparameters**: Adjusted parameters like `max_depth`, `learning_rate`, `n_estimators`, `colsample_bytree`, `colsample_bynode`, `alpha`, and `lambda`.

Why We Tried It:

- To optimize the model's performance by fine-tuning its complexity and regularization.

Outcome:

- Improved stability and slightly better performance, with more consistent cross-validation scores.

Evidence:

- CV AUC scores became more stable, and the model exhibited better generalization across different folds.

Justifications for Decisions

- **Initial Logistic Regression**: Served as a simple and interpretable baseline.
- **Switch to XGBoost**: Chosen for its robustness in handling missing data and capturing complex relationships.
- **Hyperparameter Tuning**: Necessary to optimize model performance and prevent overfitting.
- **Advanced Cross-Validation**: Ensured that the model remained stable over time, a critical aspect given the competition's focus on stability.
- **Feature Engineering**: Aimed to enhance the model's ability to capture relevant patterns and relationships in the data, leveraging both existing and new features effectively.

Overview of Final Entries

For our final models, we submitted two models: our initial logistic regression model and a fine-tuned XGBoost model.

For our initial logistic regression model, the expected Gini score was 0.09, with the lower bound being 0.05 and the upper bound being 0.12. We submitted this model to the final submission because we wanted to show the progress our team had made in model development since our last presentation. The model's performance was not the best so we pivoted our development to XGBoost. The second and last model that we submitted was the Fine-Tuned XGBoost Model. The expected score of our XGBoost Model was 0.510, the lower bound 0.484, and the upper bound 0.577.

We chose the upper, lower, and expected gini score values from our stratified cross validation scores. The lower bound was the lowest score that we got in the SCV, the upper bound was the highest values of the SCV, and the expected score was the average score.

On the day of presentation, we were surprised that our model performed way higher than the upper bound gini score that we got from stratified CV testing. We hypothesized that because the test case was built from the training data that our model had, there might have been some data leakage making it

so that the model could perform much better than expected.

Retrospective Analysis

What went right in our approach was submitting a final model that we understood well and knew how to optimize. We ultimately chose XGBoost over LightGBM or CatBoost because we were more familiar with its framework compared to the other two. With XGBoost, we were more accustomed to optimizing its parameters and identifying feature importance. This worked well for us, as we scored well in the competition and created a model that better generalized the hidden data instead of overfitting to the training data. What went wrong was not spending enough time optimizing parameters for XGBoost. We had written a program to test different values for the parameters (learning rate, max depth, regularization terms, etc.) and to choose the optimal ones, but we ran out of time and could not include this in our Kaggle submission. Additionally, we could have created more features but did not, as we already felt overwhelmed with all the data available to us. Specifically, we did not conduct as much feature engineering as we would have liked because we felt it would only marginally improve our final model. On the other hand, what went right for our baseline model (logistic regression) was that it was also easy to understand and work with. We ran tests on the model and could understand results easily, finding the best columns to use to optimize our model. However, what went wrong was that it did not perform as well as we expected. Even with our experiments, while accuracy improved by a decent percentage, it was overall still lower than the XGBoost model. We only included 20 features at a time in this baseline model due to runtime barriers, so it could not learn from the training data effectively.

In hindsight, our baseline logistic regression model could have been improved by incorporating more features and optimizing the model with regularization. We believe that by adding regularization, we could make our model more generalized and thus improve our Gini score by preventing overfitting to the training data. Our XGBoost model could also have been improved by optimizing the hyperparameters, as mentioned earlier, given that we could not include the program to do this in our final submission. Optimizing the hyperparameters would have increased the performance, generalization ability, and convergence speed. We also believe that including more features in our model, as well as creating new features, would have helped us improve both models.

The next steps we would take, given more time, would be to first fix the performance issues of our baseline model by incorporating more features and adding regularization. Then, to improve the XGBoost model, we would add the program we coded to optimize the hyperparameters and find ways to add and create more features for the model. If time permits, we would also create an ensemble of different boosting models to leverage the strengths of each model and improve overall performance. Specifically, we would create such an ensemble through stacking, voting, and averaging. This approach would hopefully help reduce variance and bias, leading to more robust and accurate predictions.