Credit Risk Analysis Using LightGBM

Capstone Project: Winter 2024

By: Henry Ker, Christian Ibanez, Braedon Shick

Table of Contents

Abstract

In today's data-driven world, the financial industry leverages all metrics to determine the responsible extension of credit for both borrowers and lenders. This data driven focus has helped revolutionize the industry but has also hurt a portion of society who lack tracked financial data. To address the underlying issue, the development of scorecards is essential in maintaining an accurate and stable model for underrepresented populations. Our capstone project, a Kaggle competition put on by Home Credit, aims to predict loan default risks. For the project, we utilized economic and behavioral data to develop a model that focuses on both short term performance and stability in the long run. The approach we took began with Polars to process large amounts of data efficiently, feature selection with SHAP Analysis, and Bayesian Optimization for hyperparameter tuning for our LightGBM Model. This approach resulted in a semi competitive competition result but really fell short with our initial goals on where we wanted to end up. Nonetheless, our project provided really valuable insight in the unique challenges with financial data, improved our data science skills massively, and opened our eyes to how challenging Kaggle Competitions are.

1. Introduction

1.1 Project Statement

For our capstone project, we chose a Kaggle competition[1] sponsored by Home Credit, a housing loan provider. The objective of this contest is to develop a model that predicts whether a client will default on a loan. This contest emphasizes the trade-off between short term performance and stability over time, and penalizes submissions which show deteriorating performance in later weeks. The creators provide economic and behavioral data, in the form of credit application history, credit bureau reports and other personal data. The findings of the competition would allow the credit institution to build scorecards which are less vulnerable to covariate shift. The dataset does contain some level of personal and demographic data, including gender, marital status, age, and zip code (although all of these have been hashed). Although our submission will be just one of many, it is important to take seriously the potential impact this could have on people's lives, and design our model in such a way that it is not perpetuating bias, or excluding any groups from loan opportunities for which they should be deemed eligible.

This is the first Kaggle competition for any member of this group, so we also see it as a chance to learn more about the platform and the competition dynamic. We chose this project for the opportunity it provided to work with a massive industry dataset, and the benchmark provided by other teams' submissions.

1.2 Evaluation Strategy

The competition scores submissions using a custom metric, derived from the gini coefficient of the ROC-AUC score for each week. First, the gini score for each test week is calculated as:

$$Gini = 2 * AUC - 1$$

A linear regression is then fit to these gini scores, with slope a, and the residuals calculated between this line and the actual gini scores. We then calculate the final metric:

$$\text{Stability Metric} = \text{mean}(gini) + 88.0 \cdot \min(0, a) - 0.5 \cdot \text{std}(residuals)$$

A successful outcome of our capstone project would be ranking in the top 10% of all teams on the Kaggle leaderboard at the time of project submission (the contest continues for another month).

1.3 The Data

The dataset comes from Home Credit Group and is licensed to Kaggle for this competition only, and therefore will not be available on our Github. The competition host has already split the data

into train and submission datasets; the submission dataset, which is kept hidden from the participants, will be used to score the competing models. This dataset is quite large: there are millions of client ids, with information from 17 different internal and external sources. Each source has from 3 to over a hundred features, and can represent either a single metric per client or extensive client history.

1.4 Development Environment

For this competition, we used Kaggle's provided notebooks as our primary development environment. This provided easy access to the data, some amount of free compute resources, and ensured that all team members were working in identical environments. This came with tradeoffs, however. The free version of Kaggle notebooks provides only CPU compute, which significantly lengthened some parts of the development cycle. In addition, git integration with Kaggle is limited, and functions mostly as a backup. Most collaboration took place by sharing links to Kaggle notebooks.

2. Methodology

The diagram below describes the structure of our final notebook.

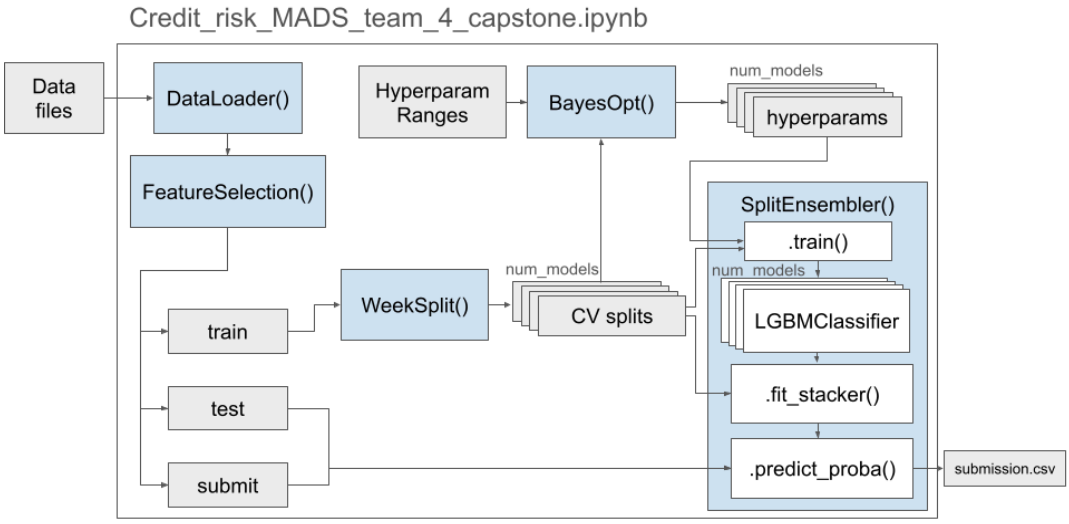

Figure 1: Capstone Notebook Architecture

After loading the data and creating a train/test split, we created a number of sampled data subsets from the train data, each further divided into CV splits. We then ran a series of hyperparameter searches to find the optimal parameters for a model trained on each data subset. With these optimized parameters and associated data subsets, we trained an ensemble of models, then generated predictions for the submission data to create the output csv file.

All non-tuned hyperparameters (number of models, data files, features, and number of CV splits) were declared in the first cell and referenced throughout the notebook. This allowed us to conceptually treat the notebook as a script, although our use of the Kaggle environment meant that running as a notebook was simpler. During model development, BayesOpt() required internet connection for logging and hyperparameter sweep through Weights and Biases API. For our submission, internet access was disabled, so a saved set of tuned hyperparameters would be loaded from a file stored on Kaggle servers. Data visualizations were also created in this same notebook (not shown above), and disabled for submission.

2.1 Model Choice

For this project, we used lightgbm's LGBMClassifier. Light GBM is a gradient boosting decision tree ensemble developed by researchers from Microsoft[2]. Gradient boosting decision trees like LightGBM and XGBoost have been the state-of-the-art classification architecture for years now, and are often at the core of winning Kaggle classification models [3]. During the early stages of this project we considered alternatives such as neural network or support vector based classifiers, but time constraints restricted us to experiment with only a single model type.

2.2 Data Loading and Augmentation

We used a programming language called Polars, which was for data loading and initial processing. Not only is Polars faster than pandas, but the lazy query evaluation allowed us to work with this dataset that would otherwise be too large to fit in local memory. At a later stage of our pipeline, we converted these data frames to pandas for better integration with scikit-learn.

The description of each file in the dataset included a depth, from 0-2. A depth 0 file had one entry per client; a depth 1 or 2 file might have many more (e.g., all account transactions of a client, or all transactions from multiple accounts if depth 2). These higher depth features required aggregation into a single value per client; we chose to use the max value of any numerical column, and the most frequent value of any categorical column (both per client). Although this did introduce some amount of information loss, we found these aggregated features to still be quite impactful for model results.

To reduce memory usage and standardize the datasets, we converted all string features to categorical and all numerical features to float32. Then we converted all dates to the number of days relative to the date of the client's loan decision, in order to make features more cohesive. Some other minor data transformations were performed to standardize format between the train and submission sets.

Early in our project, we followed a fully featured data engineering pipeline, with scaling, SMOTE oversampling, imputation, and one-hot encoding for categorical features. After we settled on lightGBM as our only model type, however, we abandoned this pipeline. Decision tree based architectures are less affected by scaling, data imbalance, or missing values, and in fact perform worse with one-hot encoding of categoricals than with their built in methods. In our testing, we did not see a significant performance difference with or without the pipeline, so we decided it did not justify the additional compute time.

2.3 Data Exploration

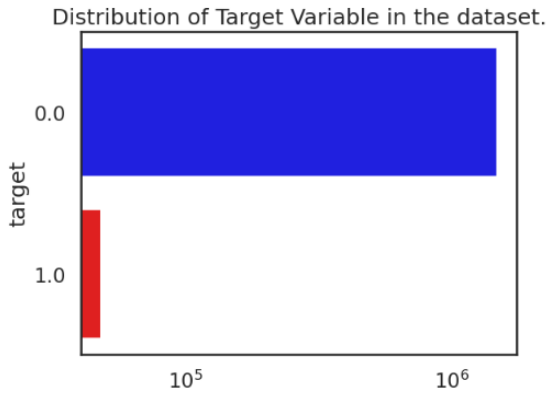

Figure 2 Distribution values for the target variable

The data is heavily unbalanced (Figure 2) , with the majority class being non defaulted loans. We experimented with oversampling techniques (SMOTE), although we found LGBM's built in tools performed better. We observed that for some of the features the proportion of missing values was over 90%, and some weeks were missing data from many sources (see fig 3 below).

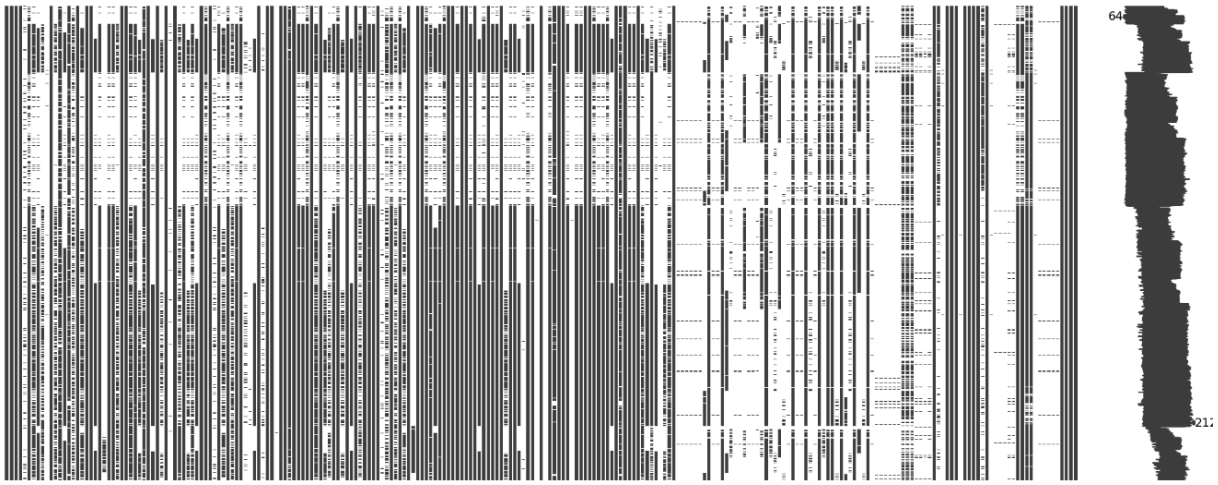

Figure 3 Graphical representation of missing data (white space).

The size of the dataset was a challenge, not only in terms of computational resources but also to exploration and interpretation. We used visual methods to identify patterns which might aid in reducing dimensionality. One of our thoughts was to identify clusters of variables with high correlation and eliminate extras, however as the submission score was negatively affected we decided to keep the whole dataset. We also explored the features with highest positive and negative correlations with the target to use for feature selection.

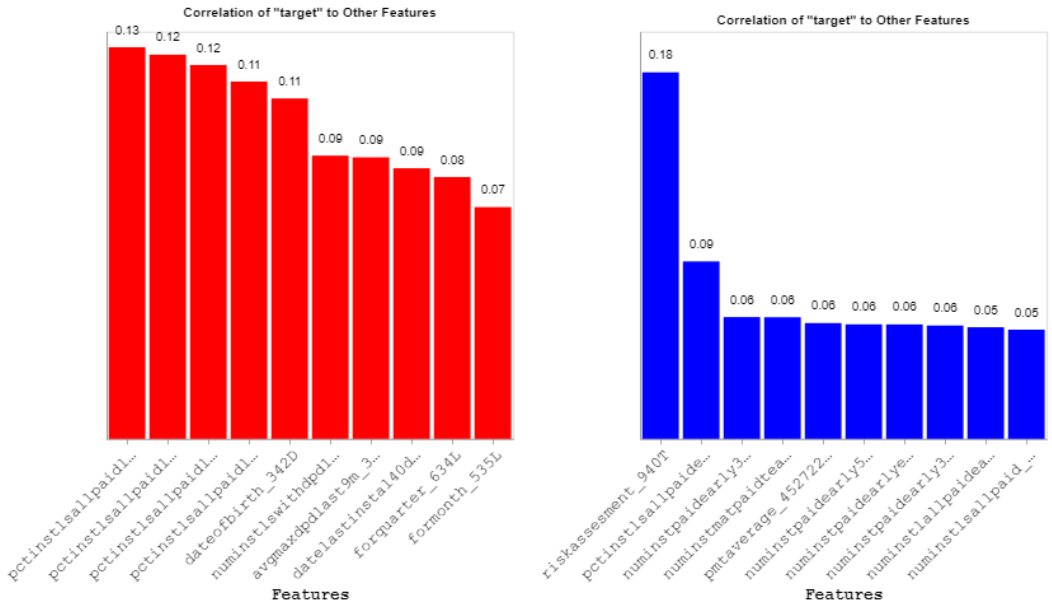

Figure 4. Correlation between target and the rest of the features (Left - Positive Correlation, Right - Negative Correlation)

2.4 Feature Selection

We explored various feature reduction methods in order to reduce runtime, one of which was Shapley Additive Explanations (or SHAP) analysis. SHAP analysis determines the contribution, or importance, of each feature in a model to the predicted result. This method was applied separately to each file in the dataset, and the top 10 most important features for each were recorded. We combined these to produce a reduced dataset, which would hopefully contain most of the information from the original dataset while allowing us to significantly reduce our train time.

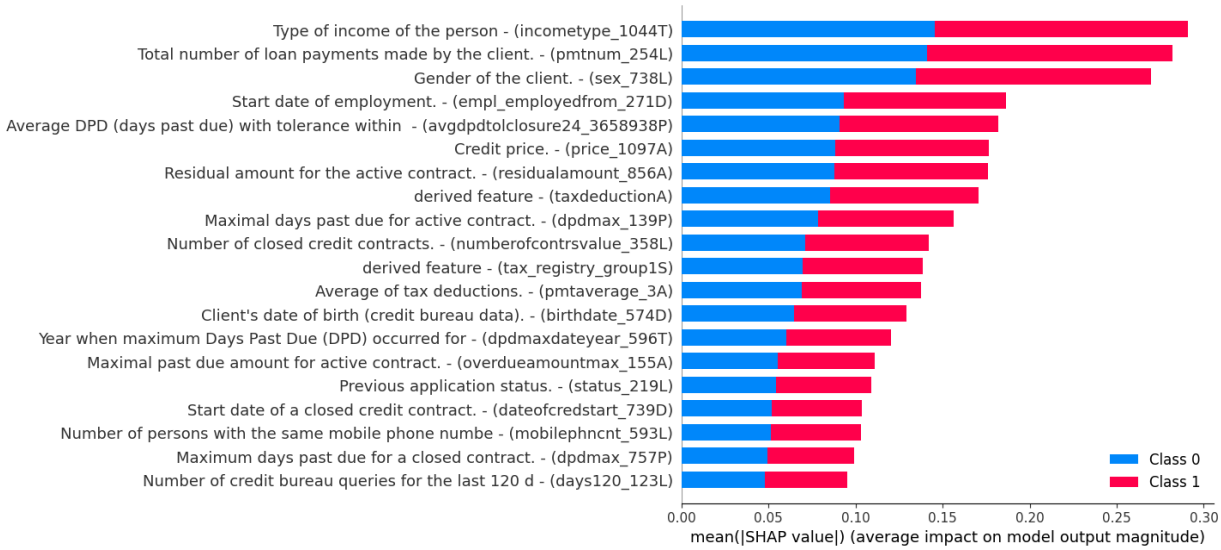

Figure 5:Preliminary SHAP Feature Analysis

Early SHAP results indicated that client gender was one of the highest importance features in the dataset. This highlighted a moral conflict of the contest: would winning the contest require creating a model that discriminated based on gender? To assess this impact, we trained models both with and without this feature. The dataset contained a few other demographic features (notably excluding race and nationality), however no others were identified as high importance by our analysis.

2.5 Data splitting

The dataset contained loans approved by Home Credit Group over a 90 week period, indicated by the feature *WEEK_NUM*. Thus, to avoid temporal data leakage, we could not shuffle the data before creating our splits. Before any processing was applied, we split off the last 30 weeks to use as a test set. All sampling from the remaining train set was also done by *WEEK_NUM*, to avoid leakage (see method in figure below).

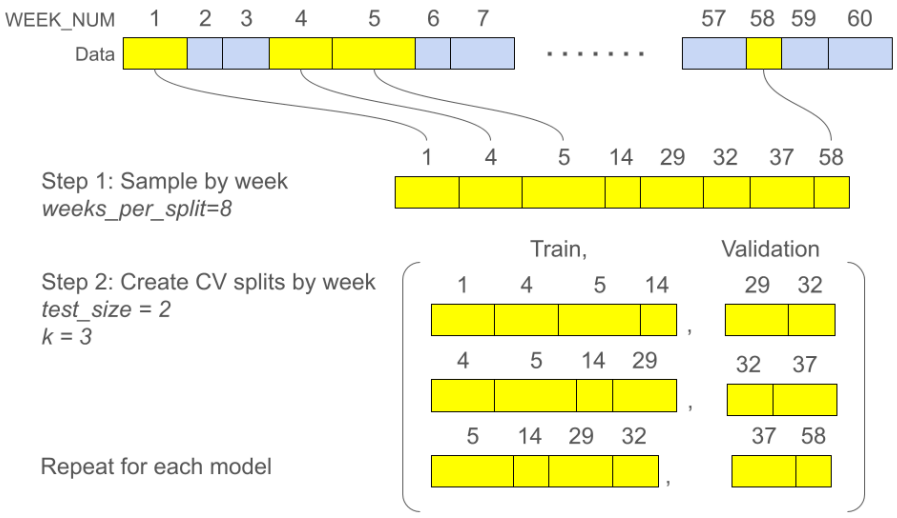

Figure 6: Data Week Architecture

The subsets for each model in the ensemble were sampled by *WEEK_NUM* pseudo-randomly (a custom routine was implemented to ensure data for each week appeared in at least one model's training subset). From this subset, overlapping cross-validation splits were created with the week number order maintained, and the later weeks of each splits reserved for validation. This is not strictly cross-validation: one of the assumptions of cross-validation is that train and test data are drawn from the same distribution, which is not the case here by design. It functions similarly, however, allowing us to gauge the variance in a model's predictive power over a range of different data splits. These cross validation splits were used for hyperparameter search; the final models were trained on their respective entire data subset.

2.6 Hyperparameter Search

We experimented with several methods of hyperparameter tuning. The number of samples and the size of the hyperparameter space made full grid search infeasible with the resources available to us. Initially, we used sklearn's implementation of HalvingRandomSearchCV. A large number of models are created with random sets of hyperparameters, and iteratively evaluated with increasing resources (in this case, number of estimators), with the worst performing parameter sets discarded at each step Midway through the project, we switched to using Bayesian Optimization, controlled by a Weights and Biases sweep. Bayesian Optimization iteratively refines its predictions using a balance of exploration (investigating choices with high uncertainty) and exploitation (confirming choices with high expected score). Hyperparameters selected by Bayesian Optimization performed similarly to those chosen by HalvingRandomSearchCV, but the search process took less than half the time. In addition, the logging provided by Weights and Biases proved valuable for better understanding the hyperparameter space. Below you can see the results of a sweep for a single model, where each line indicates a set of hyperparameters and color indicates their score (ROC-AUC). The hyperparameter choices converge to the better performing ranges as the sweep controller becomes more certain about the space.

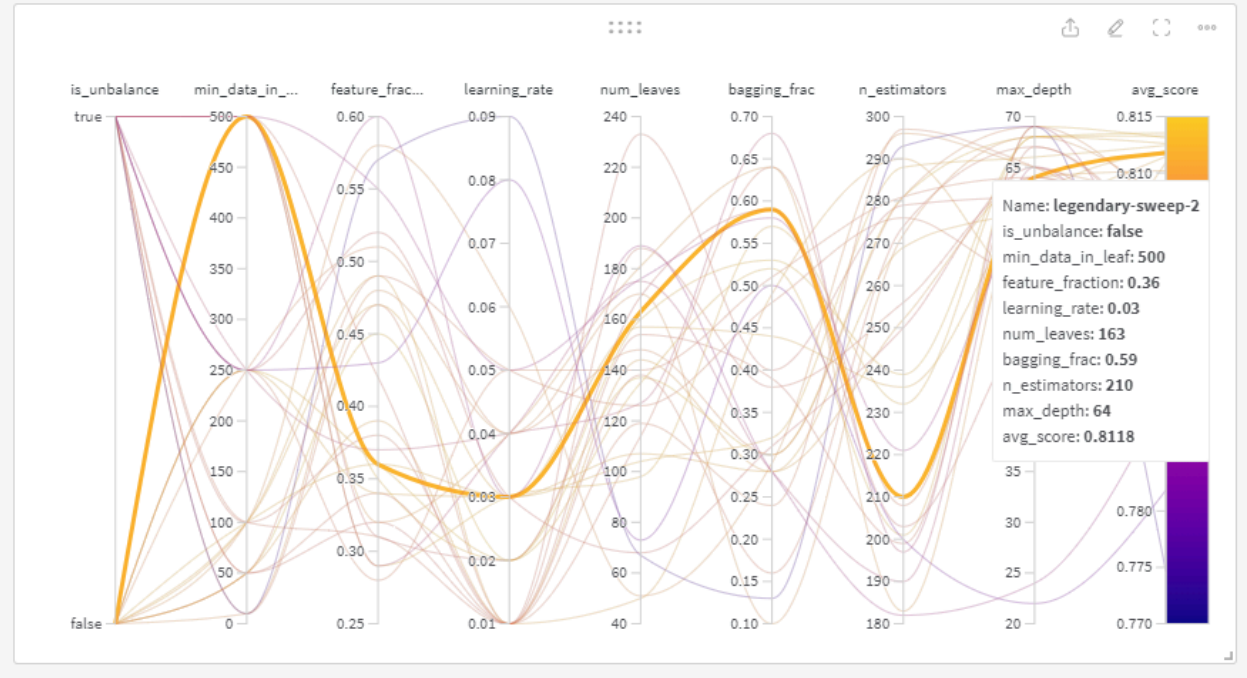

Figure 7: Weights and Biases Progression

2.7 Ensemble Stacking

Although our early experiments used a single model, we found that combining the predictions of several similar models offered a noticeable improvement in results. Each model was trained on a unique data subset, and hyperparameter search performed separately for each model. We experimented with two different methods of combining the model predictions: an arithmetic mean, and a logistic regression model trained with model outputs as features. The difference in final score between the two methods (both on our held out test data and the contest leaderboard) was minimal.

3. Results

3.1 Model Comparison

Model	**Feature Selection**	**Test score** (ROC-AUC)	**Test Score** (competition metric, 0.0-1.0)	**Competition Score** (competition metric, 0.0-1.0)
LGBM Single Model	Full	0.84	0.642	0.485
LGBM Ensemble 1 (6 models, each 33% size)	Full	**0.84**	**0.645**	**0.494**
LGBM Ensemble 2 (6 models, each 33% size)	Top 10 from each source	0.79	0.544	0.345

3.2 Competition Metric

The competition metric is derived from the gini scores for each week in the hidden submission dataset, with penalties applied for high prediction variability and decreasing performance over time. We implemented this metric in our notebook to evaluate our models on our test data: the results of the best performing model are shown below.

```
gini mean: 0.6659313924375029
falling_rate:  0
residual std_dev:  -0.021645623993061944
final score: 0.644285768444441
```

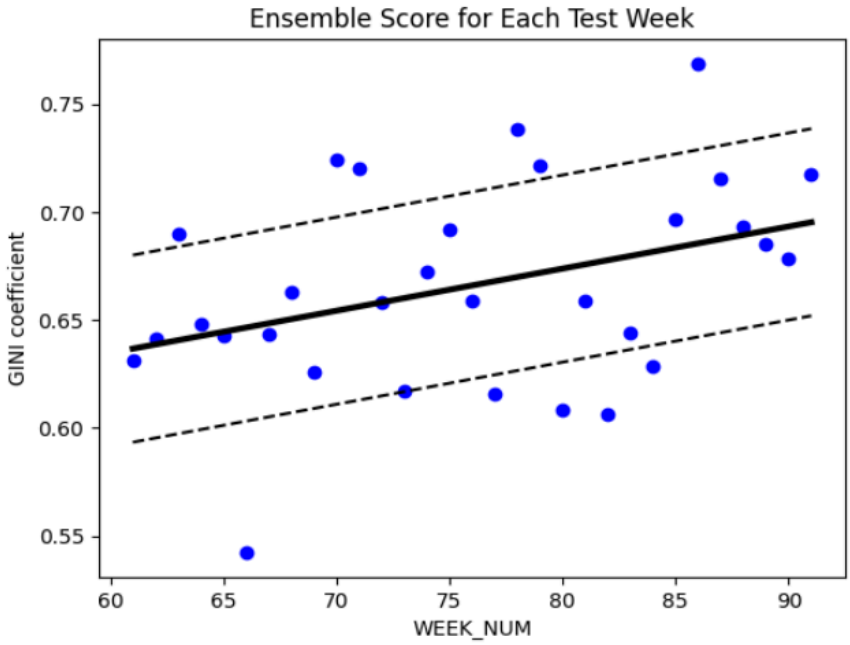

Figure 8: Model Gini Score by Week

Note that as the model scored higher in the later weeks of the test set, the falling rate penalty was 0.

3.3 Performance by Week

In order to better understand the predictive ability of the classifiers in the ensemble, we evaluated them individually on the train and test sets. The ROC-AUC by model is shown below, with out-of-sample weeks represented by a line and in-sample weeks represented with points.

Figure 9: Model Performance Per Week

Note that every model appears to overfit to its training weeks significantly. Our hyperparameter search method selected only for highest ROC-AUC, and did not penalize overfitting. Despite this, performance does not drop for test weeks (although variance increases). Also of interest is the fact that the models all perform quite similarly on out-of-sample data, despite training on different data subsets. Score variance between weeks is greater than any variance between models. It appears that they fail to specialize in any meaningful way, and instead have all learned fairly general classification functions.

3.4 Effect of Gender

We were concerned that our feature importance analysis identified gender as one of the variables that best “explain” our model’s predictions. We initially chose to retain this feature in order to assess its impact, as other teams submitting to this contest were likely to retain any features which offered a competitive edge. When we analyzed the correlation between gender

and the rest of the variables, however, we found that gender was not strongly correlated with any main economic features such as income or debt. In addition, excluding the gender feature from the model did not impact predictive power. This does not mean that the model is not discriminating by gender, of course. It is likely that the model chose gender as a proxy for economic status due to the high completeness of the feature. As the economic divide by gender shifts overtime, this model will continue to discriminate based on outdated assumptions. One interesting finding was that for women, age was positively correlated to the target while for men, age was negatively correlated. This could be explained by familial pressures reducing earning chances, making them more vulnerable to default.

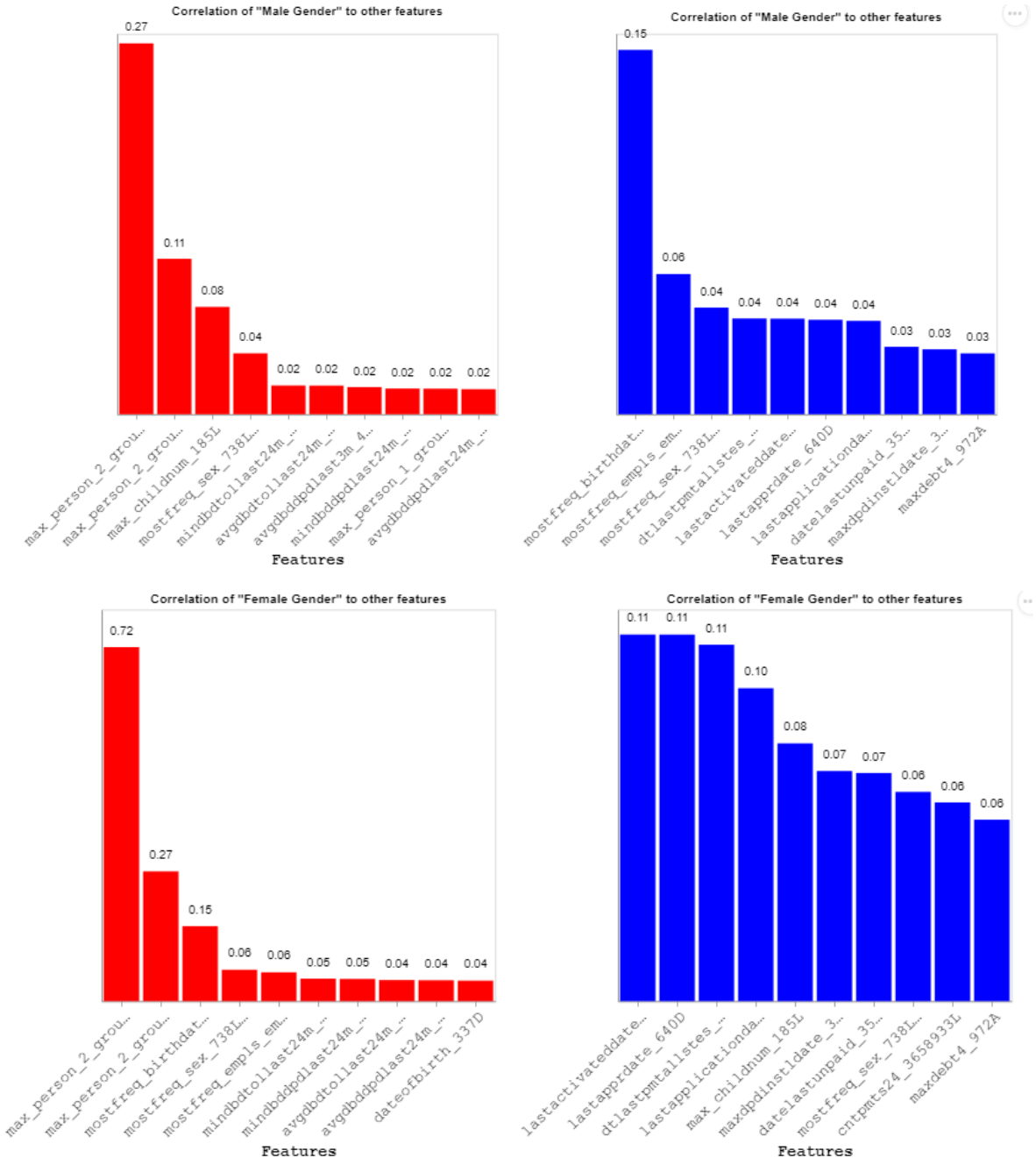

Figure 10: Gender vs Model Features Positive (red) and Negative (blue) Correlation

4. Discussion

4.1 Evaluation

Our best performing contest submission, the ensemble trained on the full dataset, scored 0.494. This was well below the top team on the leaderboard with a score of 0.600, and in fact just below the 50th percentile score of 0.509. As our initial goal for this project was placing top 10% on the leaderboard, we rule this a failure.

We believe that several factors contributed to this underperformance. First, it appears that our reserved test set was not an adequate substitute for the contest creators submission set. Our model performed quite well on our test set, scoring well above any leaderboard submissions. An ROC-AUC score of 0.84 on our test set means that our model was able to separate positive and negative cases reasonably well in the short term. As the contest metric heavily penalizes dropping performance over time, however, we believe there is a significant covariate shift at some point in the hidden submission data to which our model was not able to adapt. Dealing with covariate shift is of course not an easy challenge, but it was a primary focus of this competition. We had hoped that training models on smaller subsets of the data might allow them to specialize at predicting for different input distributions, but our architecture was not able to induce sufficient specialization.

4.2 Next Steps

We expect that significant gains could be achieved by modifying ensemble architecture to induce more specialization. Winning Kaggle models can often have quite complex architectures and contest specific customization; ours is quite rudimentary in comparison to the processes shared by [4,5] for similar classification competitions. Future modifications should stem from experimentation with more model types (e.g. neural networks and SVCs), as well as more intentional data subset selection: perhaps grouping weeks with similar input distributions, or training on a small continuous set of weeks. Models might be selected for the ensemble based on the distance between their predictions in order to intentionally incorporate variance. Although stacking algorithms did not make much difference when testing our ensemble, a more sophisticated algorithm that incorporated model prediction confidence might improve performance with an ensemble of specialist models.

In addition, improvements could be made to our feature selection and creation process. Our model trained on the reduced set of features performed significantly worse on the competition leaderboard, despite scoring only moderately worse on our test set. This suggests that the large number of features which were perceived as low importance for our training data might be key to achieving better stability over covariate shift. Future work should investigate this hypothesis by measuring the impact of different feature sets on score. This form of data leakage, tuning hyperparameters on submission results, is unfortunately standard practice for Kaggle competitions and is key to achieving top results. In addition, our feature engineering steps were

minimal, only modifying date features. Additional research into the field of credit risk might suggest potential derivative features which could improve predictions.

4.3 Conclusion

Although these results were not what we hoped, we found this project to be a great introduction to the Kaggle format, and believe that the lessons learned from this contest should shorten our experimentation cycle for future Kaggle competitions. In retrospect, our initial goal of top 10% was quite ambitious considering the deep pool of experienced teams on Kaggle and the large cash prize for this contest.

Throughout this journey we were able to learn many valuable lessons and skills to help launch us into the competitive career field of Data Science. We found the chance to learn Polars, what many in the industry consider the next Pandas for data manipulation, to be quite valuable. Another positive aspect of the project was learning a new machine learning algorithm like LightGBM and all the built in functionality it has like dealing with imbalanced data. In addition we had the opportunity to explore tools such as Weight and Biases to better track model performances and improve our model tuning capabilities, and Overleaf in the development of the poster, with a LaTeX based script that simplified the structuring and communication of our results.

A1. References

[1] Home Credit Group. (2024). *Home Credit - Credit Risk Model Stability*. Kaggle. Retrieved April 15, 2024, from https://www.kaggle.com/competitions/home-credit-credit-risk-model-stability

[2] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*. https://neurips.cc/. https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

[3] Kumar, S. (2024) *Winning solutions of kaggle competitions.* Retrieved April 17, 2024, from https://www.kaggle.com/code/sudalairajkumar/winning-solutions-of-kaggle-competitions

[4] Tunguz, B. et. al. (2018) *Home Credit Default Risk - 1st Place Solution.* Retrieved Apr 17, 2024, from https://www.kaggle.com/c/home-credit-default-risk/discussion/64821

[5] FL2O, Silogram (2019) *Santander Customer Transaction Prediction - #1 Solution.* Retrieved April 17, 2024, from https://www.kaggle.com/c/santander-customer-transaction-prediction/discussion/89003

A2. Statement of Work

Henry Ker: Primary developer for DataLoader, CVSplits, BayesOpt, and SplitEnsembler. Researched hyperparameter optimization and ensembling methods. Contributed text and visuals to report methodology, results, and discussion.

Christian Ibanez Diaz: Exploratory Data Analysis, Poster Development, Report Development

Braedon Shick: Exploratory Data Analysis, Machine Learning Model Selection, Machine Learning model validation,Poster Development, Report Development

A3. Dataset Descriptions

- •Base.csv:
  - -> case_id - case id number
  - -> date_decision – date when loan was granted
  - -> MONTH - Month
  - -> WEEK_NUM – Week Number
  - -> Target – Label for approved or not approved.
- Applprev_1_0.csv, Applprev_1_1.csv, Applprev_2.csv
  - Data from clients previous loan applications, retrieved from two different providers.
- Applprev_2.csv
  - Information about card blockage on previous credit accounts.
- Credit_bureau_a_1_0.csv, Credit_bureau_a_1_1.csv, Credit_bureau_a_1_2.csv, Credit_bureau_a_1_3.csv.
  - Detailed history for client loan contracts partitioned into 4 files.
- credit_bureau_a_2_0, credit_bureau_a_2_1…credit_bureau_a_2_10
  - Info about collateral and payment dates, partitioned into 10 files.
- credit_bureau_b_1.csv
  - History for client loan contracts.
- credit_bureau_b_2.csv
  - Num and value of overdue payments.
- Debitcard_1.csv
  - Information about debit card usage.
- Deposit_1.csv
  - Deposit account history, with features such as amount of the transaction, opening date.
- Other_1.csv
  - Transaction history of client debit account.
- Person_1.csv
  - Demographic information: zip code, marital status, gender etc (all hashed).
- Person_2.csv
  - Data about clients contacts and their employment.
- Static_0_0.csv, static_0_1.csv, static_0_2.csv
  - Three partitions of a large datasets.Contains transaction history for each case_id (late payments, total debt, etc).
- Static_cb_0.csv
  - Data from an external credit bureau, among the variables included are: demographic data, risk assessment, number of credit checks.
- Tax_registry.csv
  - Info from all three tax registry providers combined. Because this information comes from three different sources, we had to standardize the name of the variables.