Group 15 Final Project Report

Oscar Su, Allen Chen, Haohan Chen, Adrian Ruiz Doblas

2025-12-04

Abstract

The goal of this project is to predict whether a cold-rolled aluminum offcut will pass an extreme durability test using process measurements collected during manufacturing. The training data set contains labelled offcuts with variables such as alloy code, rolling and cutting temperatures, roll pressures, machine restart indicators, and quality indices, while the test set consists of approximately 160,000 unlabeled offcuts used for scoring on a private Kaggle leaderboard. We first construct an interpretable baseline model based on logistic regression, using a simple preprocessing pipeline with dummy indicators for categorical variables and standardized numerical features. This baseline achieves a validation log-loss of about 0.51 and an AUC of about 0.83, providing a useful reference point but leaving substantial room for improvement. We then move to more flexible tree-based and ensemble methods, ultimately obtaining significantly lower log-loss on the leaderboard (around 0.43) by using gradient-boosted trees and model stacking. The remainder of this report focuses on describing the data, the logistic regression baseline, and how its strengths and limitations motivated the use of more complex models.

Introduction

Many industrial manufacturing lines now collect rich process data at scale, creating opportunities for data-driven quality control. In our project, we study an aluminum cold-rolling process in which large sheets of aluminum are rolled, cut into offcuts, and then subjected to a suite of quality tests. Of particular interest is an "extreme durability' ' test: offcuts that fail this test may indicate subtle defects in the upstream process and can be costly if they propagate further in the production pipeline. Accurately predicting whether an offcut will pass this test from process measurements alone could support earlier interventions, reduce waste, and improve overall reliability.

The instructor provided a labelled training set and a separate test set for a private Kaggle competition in STATS 101C. Each row corresponds to one offcut and includes an identifier and several predictors: an alloy code, ordinal cutting and rolling temperature categories, roll pressures for the last two passes, indicators of micro-chipping and machine restarts, a source block label, and two continuous quality indices summarizing contour and clearance properties. The binary response variable, `y_passXtremeDurability`, equals one if the offcut passes the durability test and zero otherwise. The Kaggle leaderboard score is based on a held-out test set with hidden labels and uses a probabilistic classification metric (log-loss), so models must output calibrated probabilities rather than hard class labels.

Our modeling strategy is incremental. We first build a transparent baseline model using logistic regression, which allows us to understand the relationship between individual predictors and the probability of passing the test. This model also serves as a benchmark: any more complex method must justify its added complexity by achieving substantially better out-of-sample performance. After evaluating the logistic regression model on an internal validation split, we explore nonlinear, high-capacity methods such as gradient-boosted trees and ensemble stacking. These methods are better suited to capturing complex interactions and threshold effects that are likely present in a manufacturing process. In this report we describe the logistic regression baseline in some detail and summarize how its performance compares to our final ensemble model.

Explorative Data Analysis

Before trying to fit any kind of predictive models, we first conducted an exploratory data analysis (EDA) to inspect the distributions of the variables we were going to work with and to see what underlying relationships exist between these variables that would inform us of our modeling approaches.

Counts of the response variable

The response variable we're interested in this Kaggle competition is `y_passXtremeDurability` is a binary categorical variable that indicates whether the alloy cut passed the extreme durability test.

Table 1: Number of cases of each class in the training data

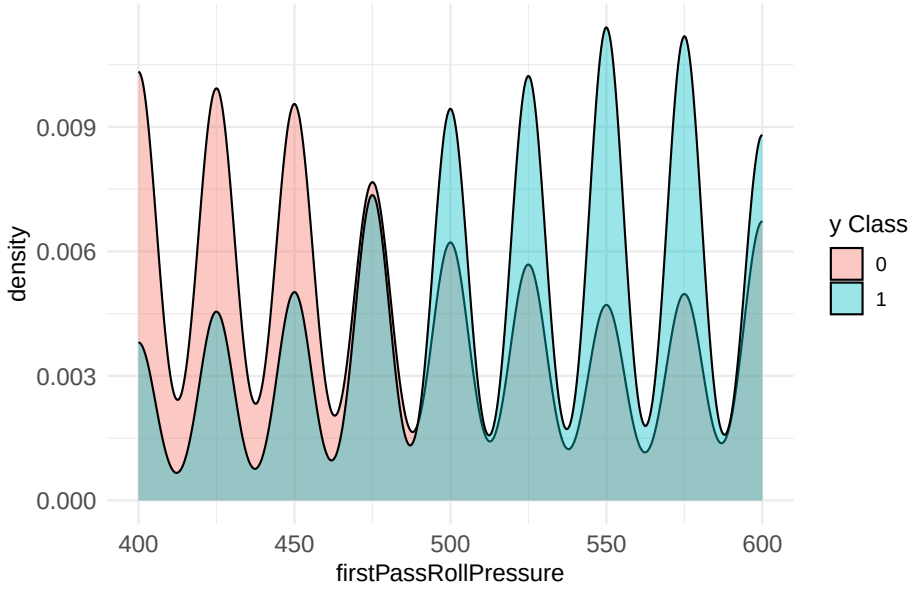
y_passXtremeDurability	cases
0	87517
1	72483

The table shows that although there are differences in the number of cases between the two `y_passXtremeDurability` cases, the discrepancy is not too large to cause concerns when fitting our model.

Relationship between response and predictor variables

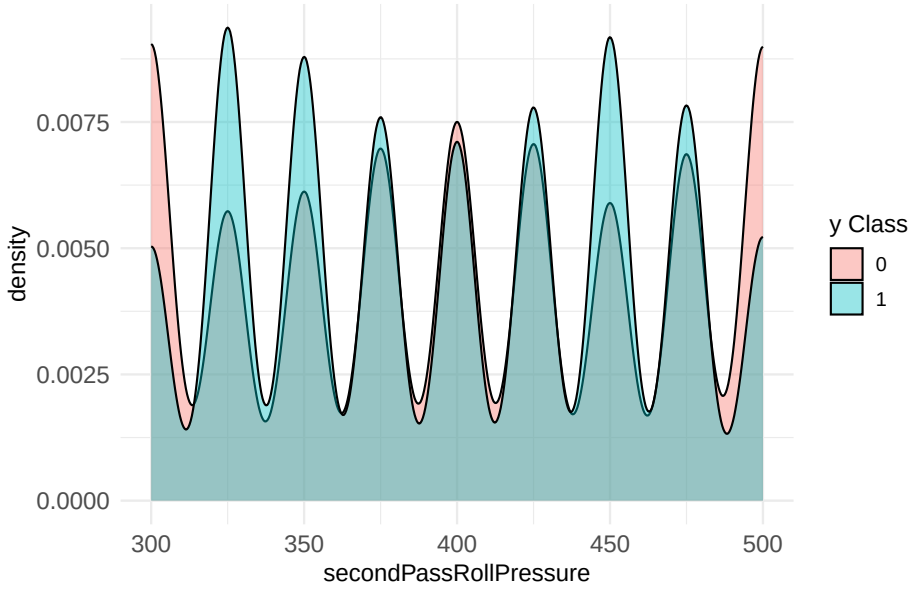
We then explore how the predictor variables differ between the two response variable classes, which will give us a better idea on how to approach our modeling.

Below are some interesting relationships we found:



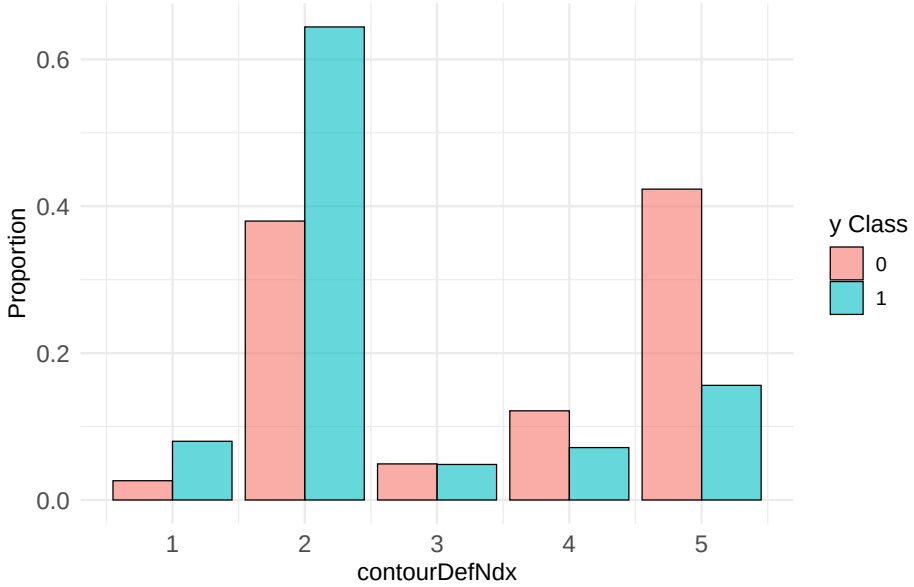
Distribution of first pass roll pressure by class

The distribution of first pass roll pressure between aluminum of class 0 and class 1 looks quite different. Specifically, class 1 aluminum has higher densities between 500 - 600 units of roll pressure while class 0 aluminum has higher densities between 400 - 450.



Distribution of second pass roll pressure by class

Interestingly enough, the distribution of second pass roll pressure between the two aluminum classes do not look that different, suggesting that the roll pressure of the second pass might not have much effect on aluminum durability.



Proportion of contour index by durability class

Another interesting variable we found is `contourDefNdex`. From the side-by-side bar graph, it is clear that durability class 0 has higher proportions in indices 2, 4, and 5, while class 1 has more than half of its observations in index 2.

While some variables like first pass roll pressure and contour index seem important in determining the durability class of the aluminum, we cannot say for certain that they are the only factors that matter. Since feature selection is a complex procedure, we should still perform thorough model fitting and use appropriate metrics to select the relevant features empirically.

Note that this does not mean that the variables highlighted in our EDA are meaningless. The highlighted variables can serve as important model tuning points as we are sure that they can be important in distinguishing the durability classes.

Data fitting and tuning

Initial model: Logistic Regression

Logistic regression provides a natural starting point for this binary classification problem. Let $Y_i \in \{0, 1\}$ denote whether offcut i passes the durability test, and let x_i be the corresponding vector of predictors. The logistic regression model assumes

$$\Pr(Y_i = 1 \mid x_i) = p_i = \frac{1}{1 + \exp\big(-(\beta_0 + x_i^\top \beta)\big)},$$

so that the log-odds of passing are linear in the predictors:

$$\log\left(\frac{p_i}{1 - p_i}\right) = \beta_0 + x_i^\top \beta.$$

The parameters (β_0, β) are estimated by maximizing the likelihood, which is equivalent to minimizing the empirical log-loss over the training data.

To assess out-of-sample performance, we evaluated the model on the 20% validation split that was held out during training. Table 2 summarizes the main metrics. The validation log-loss is about 0.51, with an AUC of approximately 0.83 and an accuracy of roughly 0.75 when using a 0.5 classification threshold. The mean squared error between the predicted probabilities and the binary labels is around 0.17. When we refit the same pipeline on the full training set and submit predictions to Kaggle, the public leaderboard log-loss is also close to 0.51, indicating that the internal validation split provides a reasonable estimate of the model's generalization performance.

Table 2: Performance of the logistic regression baseline on the validation set.

Metric	Log-loss	AUC	Accuracy	MSE
Validation	0.51	0.83	0.75	0.17

Second model: Boosted Trees

In our dataset with many categorical variables and predictors, exploring possibilities of non-linear relationship seems like a natural next step, and Boosted Trees was our choice. We explored two possibilities of boosted trees (LightBGM and XGBoost), but we found XGBoost's performance to be superior to LightBGM and thus decided to focus on XGBoost Boosted Trees (though it is worth noting that LightBGM's main advantage is the speed, allowing for faster iterations and feature engineering exploration, so it is expected that with the same parameters and model, XGBoost would perform better).

Feature Tuning

Our base model is y_passExtremeDurability against every predictor (except ID). This model performed much better than our logistic model (See end of section to see numerical results), but with some feature engineering, we can further improve the model.

First, we observed that cutTemp and rollTemp are ordered factors (with it being high > medium > low). Trees work by strictly comparing values (i.e., is cutTemp = high?), so to help "nudge" the model to learn the ordering of temperature, we turned the factors into ordered numerical values. High -> 2, medium -> 1, low -> 0. Adding this improved the accuracy of our model. It is worth noting that we explored the possibility of converting the other categorical predictors into numerical ones as well, but the other predictors were not ordered like cutTemp and rollTemp, so we decided against it.

Second, our hyperparameter tuning told us that an optimal max tree depth of 3 is best. With boosted trees, each step adds another tree onto the last one and by limiting each tree to just a depth of 3, this helps prevent overfitting (and makes training the model much quicker). Unfortunately, this is a tradeoff as while we prevent overfitting, with each tree being less shallow, complex interactions between the different predictors might not get explored by the boosted tree. Thus, to ensure the model finds and explores these interactions, we forcibly add interaction terms to our XG design matrix. These interactions are:

- rollTemp:firstPassRollPressure:secondPassRollPressure:alloy
- cutTemp:alloy:contourDefNdx
- firstPassRollPressure:secondPassRollPressure:clearPassNdx
- machineRestart:blockSource
- blockSource:alloy

Through our tests, these interaction terms helped improve the model accuracy quite a bit. It is worth noting that in these interaction terms, rollTemp and cutTemp are the *categorical* versions, not the numerical descriptions above. So while we exclude the base categorical versions of the temperature predictors, we use the categorical versions for the interaction terms, with testing showing better performance using this split.

Lastly, we suspect that alloys are similar to one another, which might explain why the alloy codes are the way they are (6000 series vs 2000 series). We want a way for the model to more intuitively find the similarities between these different alloys. The way we decided on was using neural network embeddings. We trained a basic neural network to predict y_passExtremeDurability and defined a layer for alloys (and the other numerical inputs). Then, once the neural network was trained, we grabbed the embeddings from the alloy layer and augmented them in our boosted tree model. This hopefully adds more information to the alloy code, allowing for comparisons between the different types of alloys. For our model, we went with 3 embeddings. With these new embeddings in our dataset, we replaced every instance of alloy in the interaction terms with the 3 embeddings instead. We also added an additional interaction "embedding1:embedding2:embedding3:alloy".

All 3 of these additions helped boost our boosted tree model.

Training and Cross Validation

To train the model, we employed the use of cross-validation. The XGBoost library has a built-in cross validation training function included in which it conducts cross validation for the number of rounds used for training and finds the best round accordingly. Using this function, we conducted 5-fold cross-validation for max of 1000 rounds, and it has an early stop of 50 rounds (meaning if testing AUC doesn't improve for 50 it will stop the cross validation early). This gives us the benefit of getting an estimate of how our model performs in real-world data and also gives us a good number of rounds for our model to train on.

Also worthy of note, we tested our model on both ROC-AUC and log-loss metric. The models obtained from either metric do not differ from each other by much from our testing.

Hyperparameter Tuning

Besides feature engineering, we also fine-tuned the hyperparameters to get better results. We combined manual tuning and grid search.

We first manually tested a few key values for the hyperparameters and then construct a large grid with the different candidate values and run cross-validation on each combination of the hyperparameters.

Through this process, we found that the best parameters were:

- Max Tree Depth = 3; Worth noting that Max Tree Depth of 3 or 4 did not differ by much and the differences between them can be explained by training variability, so we went with 3 as that resulted in quicker training time allowing for us to attempt more combinations
- Subsample = 0.85

- Column Sample by Tree = 0.85
- Min Child Weight = 2
- Learning Rate = 0.05

We also found that very minor amounts of L1 regularization helped the model with $lambda = 1$ helping performance.

Boosted Trees Results

Table 3: Performance of Boosted Trees on the validation set

Model	Log-loss	AUC
Base Model	0.4275250	0.882272
Enhanced Model	0.4243888	0.884271

Our final boosted tree resulted in a test AUC of 0.88 and, on the Kaggle competition, received a public score of 0.426434 and a private score of 0.422153, placing us in 16th place on the leaderboard.

While the result is clearly an improvement from the simple logistic regression, we believe we could do even better, so we tried stacking the models next.

Final model: Stacked Model

For the final model, we conceived of a model that could combine XGBoost, LightGBM, and Catboost, in which all three are important gradient boosting frameworks that can make up for each other's shortcomings. For instance, since there were many categorical predictors, Catboost could handle these predictors effectively without manual encoding, which helps preserve more of their true values. Furthermore, LightGBM is extremely effective when it comes to processing large datasets, and XGBoost is well-rounded in terms of speed, efficiency, and preventing overfitting.

Hyperparameter Tuning (Individual Models)

Table 4: XGBoost Hyperparameters

Hyperparameter	Value
n_estimators	300
max_depth	5
learning_rate	0.0701247852345678
subsample	0.892345678912346
colsample_bytree	0.801234567891234
min_child_weight	5.8
gamma	1.74234567891235
reg_lambda	0.312345678901234
alpha	9.23456789012346
eval_metric	logloss
tree_method	hist

Table 5: LightGBM Hyperparameters

Hyperparameter	Value
n_estimators	300.0000000
learning_rate	0.0282457

Hyperparameter	Value
num_leaves	18.0000000
feature_fraction	0.8423457
bagging_fraction	0.6482346
bagging_freq	1.0000000
min_data_in_leaf	48.0000000
max_bin	468.0000000
lambda_l1	0.2123457
lambda_l2	4.7654321

Table 6: CatBoost Hyperparameters

Hyperparameter	Value
iterations	300.0000000
learning_rate	0.1712346
l2_leaf_reg	1.0345679
random_strength	0.9781235
bagging_temperature	1.3145679
border_count	74.0000000
depth	4.0000000

We began setting up our final model by running each of the three models separately, and we used a train-test split with an 80% - 20% split to evaluate the performance of each individual model. This allowed us to use Optuna to automatically optimize for each individual model separately and reduce the amount of computational resources necessary than if every model was run in tandem. The best hyperparameter values are listed in the tables above for each individual model.

Training and Cross Validation (Stacked Model)

We ran a 5-fold stratified cross-validation scheme to obtain the out-of-fold predictions of the stacked model, with each individual model having the best hyperparameters. This would allow for optimization of the weight applied to each individual model towards the overall stacked model.

Weight Optimization

We used Optuna to optimize the weight applied to each of the individual models. This is expressed in the equation below:

$$\hat{y}_{\text{stacked}} = w_{\text{XGB}}\hat{y}_{\text{XGB}} + w_{\text{LGB}}\hat{y}_{\text{LGB}} + w_{\text{CAT}}\hat{y}_{\text{CAT}}$$

where $\hat{y}$ refers to the predictions for each model with each contributing to the predictions $\hat{y}_{\text{stacked}}$ of the final stacked model. The constraints are that the individual weights must be greater than or equal to zero and must sum up to one.

For our stacked model, the best combination of weights that maximized the AUC is the values in the table below:

Table 7: Optimized Weights for Stacked Model

Model	Weight
XGBoost	0.3847204

Model	Weight
LightGBM	0.0002437
CatBoost	0.9500863

Final Model Results

Our final stacked model resulted in a test AUC of **0.884329** and, in the Kaggle competition, received a public score of **0.426060** and a private score of **0.421648**, placing us in 14th and 15th place on the public and private leaderboards, respectively.

Limitations & Next steps

The main limitation of our final model was that the final model could have been improved with other methods that could bring more substantial results in terms of increasing the AUC. Therefore, our next steps would be to test other ways that would improve the AUC.

For instance, instead of the LightGBM model, which is more fitting for a larger dataset, we could have used a neural network or a different model to more efficiently capture relationships in a smaller dataset. The main indicator that showcases the necessity of replacing the LightGBM model in the stacked model is due to the weight applied to the LightGBM model, which is approximately 0.0002, meaning that its contribution to the predictions of the stacked model is almost negligible.

In addition, instead of running each of the individual models in JupyterLab, we could also explore other IDEs that have the option to utilize GPUs, such as in Colab, the increase the speed of the hyperparameter tuning process, such that more possibilities can be tested.

Finally, a possible improvement was to just fine-tune CatBoost earlier when we were exploring different methods, since CatBoost performs exceptionally well on the dataset even without any combination of stacked models.

Conclusion

In this project, we addressed the problem of predicting whether cold-rolled aluminum offcuts pass an extreme durability test, using process data supplied for the STATS 101C Final Kaggle Project. We began with an interpretable logistic regression baseline, after constructing a straightforward preprocessing pipeline that handles mixed categorical and continuous predictors. This model achieved a log-loss of about 0.51 on both an internal validation set and the Kaggle public leaderboard, along with an AUC around 0.83 and an accuracy near 0.75. These results demonstrate that the available process variables carry substantial predictive signal and that a relatively simple linear model can already separate passing and failing offcuts reasonably well.

Nevertheless, comparison with the best leaderboard scores showed that logistic regression leaves considerable room for improvement. Its linear decision boundary is too restrictive to capture the nonlinearities and high-order interactions that likely govern durability in a complex manufacturing environment. Guided by this observation, we treated logistic regression primarily as a transparent benchmark and shifted our attention to more expressive methods, including gradient-boosted trees and stacked ensembles. These models achieved public and private leaderboard scores around 0.43, significantly outperforming the baseline at the cost of reduced interpretability. Taken together, our results highlight a familiar trade-off in applied predictive modeling: simple models provide clarity and a solid starting point, while more sophisticated algorithms are often required to reach state-of-the-art predictive accuracy in competitive settings.