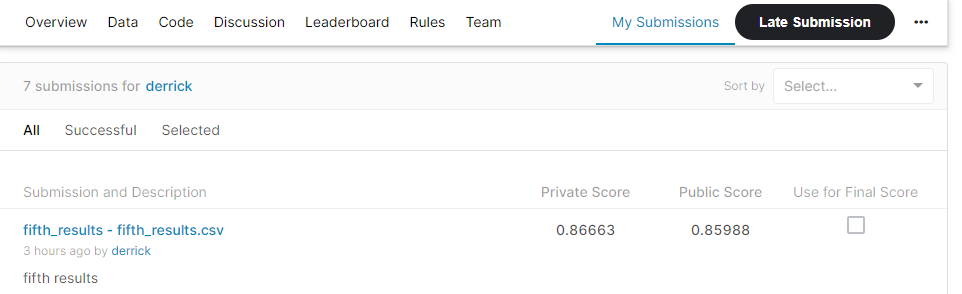
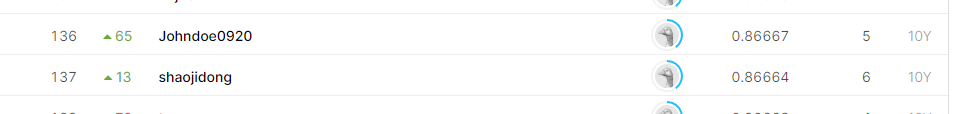
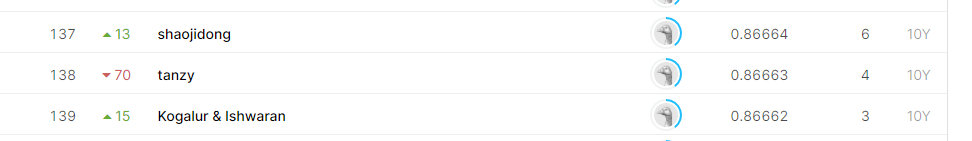
Model Results





My model scored 0.86663 on the private leaderboard, which places me 138/139 in rank

Answers

**Tell us how you validate your model, which, and why you chose such evaluation technique(s).**

Model validation is important because we cannot simply assume that a model trained on historical data will have the desired accuracy and variance in production (real-time, real-world). In order to have some level of assurance of the accuracy of the generalizability of the model, we need to validate it - test its performance on unseen data that we ‘create’.

I used cross-validation - a process where some number of samples of the data is kept separate from the training data, and later used for testing. Specifically, I used both K Folds Cross Validation to identify which Machine Learning algorithm (among a list of seven) performed best. Following the selection and tuning of the algorithm, I used Train-Test-Split to assess its performance on a single created test set (I explain later why I do this instead of another K Folds)

*K-Folds Cross Validation*

I use the K-Folds technique for the most important parts of the validation process - identifying which ML Algo (or combination of them with a Stacking Classifier) and tuning the chosen model (for the best ROC AUC score). K-Folds has the best chance of resulting in the least bias because every observation from the original dataset has the chance of appearing in the training and test set. This is particularly important in imbalance datasets where members of the minority class have smaller representation, and their important information more likely to be missed by the algorithm.

However, K-Folds is computationally expensive. Given my resource limitations (working on a simple non-GPU laptop), more than two rounds of K-Folds would have been too time-costly.

*Train-Test-Split*

As a compromise, after the selection and tuning, I used Train-Test-Split for the comparatively simpler task of assessing the tuned model’s performance on a single test set. If this approach was done alone, there is a possibility of high bias, especially if we have limited data, because we would miss some information about the data which we have not used for training. However, in my approach, this is balanced by performing K-Fold beforehand.

**What is AUC?**

**Why do you think AUC was used as the evaluation metric for such a problem? What are other metrics that you think would also be suitable for this competition?**

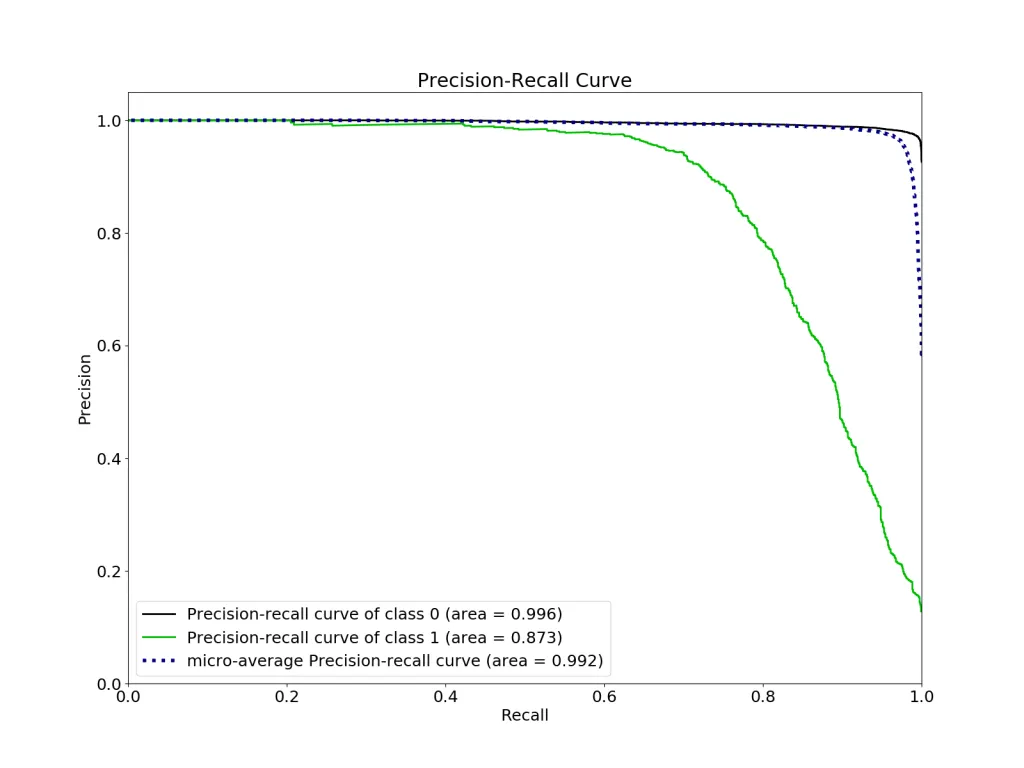
AUC is short for “Area under the curve”, a kind of metric score to judge the performance of a model. However, in speaking about AUC, we need to be clear there are several kinds of AUC scoring.

We will first define ROC AUC - the scoring method used by the Kaggle case. The Receiver Operating Chart (ROC) is a chart that visualizes the tradeoff between true positive rate (TPR) and false positive rate (FPR). For every probability threshold, the TPR and FPR are plotted. The higher TPR and the lower FPR for each threshold, the better the model performs. Visually, models that curve towards the top-left corner are better. In order to get the single metric that tells us how well the model is performing, we can calculate the Area Under the ROC Curve, or ROC AUC score.

ROC AUC is useful because it tells us that this metric shows how good at ranking predictions your model is. It tells you what is the probability that a randomly chosen positive instance is ranked higher than a randomly chosen negative instance.

However, I was initially very confused and concerned about optimising using ROC AUC. Generally, we should not use it when data is heavily imbalanced, like in this case study. It was discussed extensively in this [article by Takaya Saito and Marc Rehmsmeier](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4349800/). The intuition is the following: the false positive rate for highly imbalanced datasets like ours (where the minority class is < 10% of the majority class) is pulled down due to a large number of true negatives.

Instead, I would have preferred using Precision-Recall AUC (PR AUC), which gives the average of precision scores calculated for each recall threshold. Similar to ROC AUC score, we can calculate the Area Under the Precision-Recall Curve to get the single metric that describes model performance. PR AUC fits our case better because it is more suitable data is heavily imbalanced, like our case. This was discussed extensively in this [article by Takaya Saito and Marc Rehmsmeier](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4349800/). The intuition is the following: since PR AUC focuses mainly on the positive class (PPV and TPR) it cares less about the frequent negative class.



**What insight(s) do you have from your model? What is your preliminary analysis of the given dataset?**

As I questioned what would make a borrower’s delinquent behaviour, my first intuitive guess was that chances of delinquent events increased the moreoverleveraged the borrower was. Meaning, the borrower has borrowed beyond the ability of their means to repay. This led to my choices during Feature Engineering - where I sought to derive variables that could distill signals of overleveraging or struggle to repay.

One quick way to do this, was to normalise ‘due’ or ‘late’ data by the amount that the borrower has leveraged. What do I mean by this? For example, we have data on the number of instances a borrower has been due or late. But it has less meaning in the simple aggregate, than when normalised over the number of loans. Why? The latter gives us an indication of the *consistency* of the inability to repay on time over the number of loans. There is added meaning onto a person being late 10 times, when we discover that he/she is late on *average 5 times per loan*.

I attempted to engineer some features that express this, namely:

* ratio\_PastDue\_to\_no\_loans: number of past due instances divided by TOTAL number of loans taken
* Ratio\_late\_to\_no\_loans: number of late instances divided by TOTAL number of loans taken

Indeed, ratio\_PastDue\_to\_no\_loans showed a feature importance score of 129, the seventh most important feature.

Ratio\_late\_to\_no\_loans also had a positive importance score, although less at 43. Possibly because ‘late’ events were rarer.

This confirmed the insight that features that express overleveraged behaviour, were important predictors for delinquency.

The second insight was that developing cluster labels and distance of each observation from their respective centroid as features resulted in better performances. The intuition behind this is that clustering can help ‘mine’ the underlying hidden structure of the data, representing each observation with a layer of abstract simplicity that can help classifiers make better decisions. Indeed, to my surprise, the “distance\_from\_centroid” feature has a very large feature importance of 366 (fourth highest). Encoding clusters as features using distance from the examples to cluster representative seemed to work significantly better than merely relying on their cluster labels (feature importance of only 4)