**Default prediction project**

The project entails forecasting the probability of default for users given a few dozen features. By glancing through the data, as well as by the nature of defaults, it is quite clear that the target class is highly imbalanced (with actual defaults accounting for ~1.5% of all users).

On the features side, the ones that have a greater–albeit still fairly weak–linear relationship to default rates could be interpreted as leading indicators to an actual default, such as having been flagged any “account_worst” status (though I am not claiming causality here). There is an issue, however, that such features have a very high proportion of “NaNs”, up to roughly 70%. Ordinarily I would exclude features with that level of sparsity, but I opted to leave them in given their higher correlation to the target variable.

In terms of model selection, again the primary issue was dealing with the class imbalance. The first step to address this was running a stratified cross-validation analysis. I also decided to combine sampling techniques (oversampling with SMOTE and undersampling with ENN) to even the distribution of labels before fitting the models, at the manageable cost of slowing down the selection pipeline quite a bit. Lastly, I chose the F1-score as the key performance indicator in order to balance the precision and recall scores.

After running a few iterations of classifier comparisons, it seemed clear that the ensemble methods were performing better, so I pre-selected both the Gradient Boost and Random Forest classifiers for a tuning of hyperparameters. Since both of these methods can handle the class imbalance quite well if properly tuned, I removed the over/under-sampling of the pipeline, mainly for efficiency reasons. The version with the best parameters of both classifiers performed somewhat similarly in terms of F1-score, but I eventually selected the Random Forest model because of its higher recall. This is largely an assumption at this stage, but I would imagine that tracking the recall more closely makes more sense from a margin standpoint.

I acknowledge that the final model score is far from optimal. I would argue that more value could be harnessed from an in-depth feature selection, particularly considering the above-mentioned higher-correlated sparse features, as well as the large number of exponentially-distributed features (mainly the non-categorical ones). Lastly, and perhaps equally importantly, calibrating the predicted probabilities would also be required before pushing the model to production.

Lastly, from an engineering standpoint, a number of obvious improvements could be mentioned. Besides testing, improved error handling and logging, I would probably add an authentication layer to the API (e.g. bearer token with a login endpoint), as well as support for sending data of multiple users at once.