e-mail to PM

Hi,

I have finished the prototype model for the credit default model. Overall, the results seem promising and would serve as a good proof-of-concept for ML in credit modeling. I will summarize the project in this e-mail.

The are 3 **datasets** available, namely borrower, loan and payments. All three datasets have 887 379 rows and 14, 18 and 41 columns respectively. As the payment data seems to be dated recently, I argue it cannot be used in a model where one is trying to predict credit default straight after receiving their loan approval. I also omit data from loans that were originated prior to 2010 to avoid market dynamics from the financial crisis.

Loan status is used to construct the **target variable**. I find it reasonable to only include the loans that are terminated – i.e. the ones with a status as either 'Charged Off' or 'Fully Paid' - in order to compare apples to apples. The stratified train-test split shows a class imbalance where positives (defaults) make up 18% of the train data. I have chosen not to oversample/undersample the data due its shortcomings (loss of training data or reusage of data causing bias) compared to the relatively low imbalance.

The **EDA** is quite extensive as I go through all borrower and loan features in detail, evaluating their usefulness. The features that seem to be most important (verified by the model's features importance) are address state (US state provided by the of the borrower's address), sub-grades (granular grade assigned by the client) and annual income of the borrower.

The **data pre-processing** consists of cleaning some of the features, dropping the ones with (almost) no variance, dropping those that correlate >99% with other features, dropping the ones with >90% NaNs, and those that require too much feature engineering to make use of. Some string features such as 'Description' and 'Employment Title' could be worth looking more into with more feature engineering.

As **scoring metric**, I'm using roc-auc as in this case one might argue that predicting both classes is important, and hence focusing on both TPR and FPR. I would think a company with a business model as the intermediate between borrowers and lenders would benefit from a decent FPR to not forego too much business opportunities.

The **modeling** is done by first spot-checking 3 different models, namely logistic regression with the SGD algorithm, sklearn's Random Forest and LightGBM. Showing a consistenly higher auc (above 0.70), I progress with the LightGBM to find the best hyperparameters using Bayesian Optimization (with the Hypopt library). Due to its requirement for computational power, I use a sample of the training data to perform the kfold cv iterations to find optimal hyperparameters. Using these optimal hyperparameters on a kfold cv on the full training set actually shows a slightly lower auc score than with the spot-checked parameters (but still above 0.70). This is likely due to the sampling of the training data. On the bright side, the optimization shows a positive trend in score with respect to the number of iterations, and would likely lead to a higher score when using the full training set for optimization. When developing the model for production, I would thus suggest to use bayesian optimization for the hyperparameters, but training it on the

full training set.

The **results on the test set** are good, with a roc-auc of ~0.71 – showing no signs of overfitting. The model makes it easy to experiment with different thresholds for class prediction, and hence work with the TPR/FPR trade-off. I think this should be done along with the client to agree on which metrics are most important.

To assess the model's **business value**, I compare it to a model that might be similar to the one being used today, i.e. a model based on sub-grades. The subgrade model is built by allocating a probability of default linearly across the sub-grades (from 0% on A1 to 50% on G5). By tweeking the threshold of the subgrade model to match the ML model's number of TPs, one can compare the performance by looking at the confusion matrices. The ML model clearly outperforms the subgrade model with 201 less FNs and 1307 less FPs, clearly demonstrating the ML model's added business value.

The **limitation** of the model, as briefly mentioned, is that I have not treated the classes imbalance. Although the imbalance isn't as bad as many other applications, one could possibly obtain improvements by using sklearn's RandomUnderSampler() and LightGBM's scale_pos_weight parameter. I am though not too comfortable with those techniques, which is the reason why I haven't tried them out here. For the next stage, when working with a model to be deployed into production, I would suggest to try out these techniques.

Feel free to reach out to me if you have any questions about the model or presentation.

Best regards,

Adrian Bergem