Credit Score Modelling

Chris Gough

Introduction

This report has been compiled for the Explore technical competency assessment in order to detail the process I took and the steps completed in order to build a classification model for the credit scoring data provided. I really enjoyed completing this assessment and learned a lot in the process.

Project steps were as follows:

1. Perform initial exploratory data analysis.
2. Clean and transform data.
3. Perform modelling tasks and hyperparameter tuning.
4. Save best model.
5. Create a docker image for the project repository.
6. Deploy docker image to web for production use.

Workflow

I first began by using a cookiecutter template for data science to better structure my project repository and filesystem. I also used the package dvc which is designed for data version control to keep track of any transformations performed on the dataset.

I began with a quick exploratory data analysis step to find understand the data a bit better, investigate what is missing and how I should handle any imputation and cleaning I needed to do. I decided to drop the date fields and rather using a numeric field for number of years data present. I also converted the payment note columns to boolean values to indicate if this was present. For missing values I set all to 0 this did not have an on the models and I did not want to drop any data from my analysis.

I then moved on to the modelling stage, where I used MLFlow in order to keep track of all my model testing and runs. I tested a few classification models using a function I wrote to quickly compare them all for basic performance and

then take the best model forward for more testing and hyperparameter tuning. This model comparison step used Stratified-K Fold cross validation and from this it was clear the the LightGBM model had the best performance. A table of model scores from this step is presented below.

model	fit time	score time	accuracy	precision	recall	f1
LGBM	2.127338	0.036013	0.745765	0.746466	0.694849	0.717059
DT	0.309155	0.009129	0.642168	0.607341	0.604957	0.605851
LOG	0.318079	0.012320	0.577925	0.566147	0.454487	0.484343
SVC	1.961555	2.165289	0.505343	0.373216	0.323239	0.301150

I then decided to move on to the next stage and wrote a model using a LightGBM classifier which performed very well on the test dataset with an accuracy score of around 0.76. At this stage I did some hyperparameter tuning using the Optuna package and found that I could not get the model to perform much better then what I had. I also tested a neural network classifier that I built with Tensorflow but unfortunately it was outperformed by LightGBM. I have included the confusion matrix for the prediction step below (Figure 1).

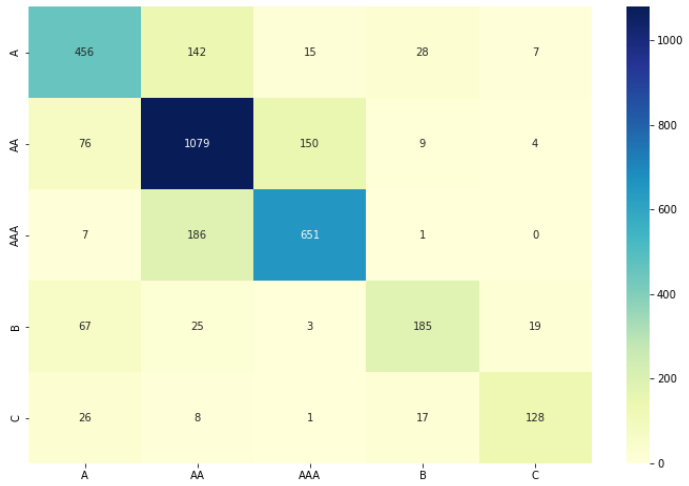

Figure 1: Confusion matrix for LightGBM model predictions

One of the criteria for this assessment was to find the top 3 features by importance and I have included this in the visualisation below. I found that the equity ratio, profit after net financial items and the cash ratio were the top 3

features by importance. Figure 2 below shows the top 10.

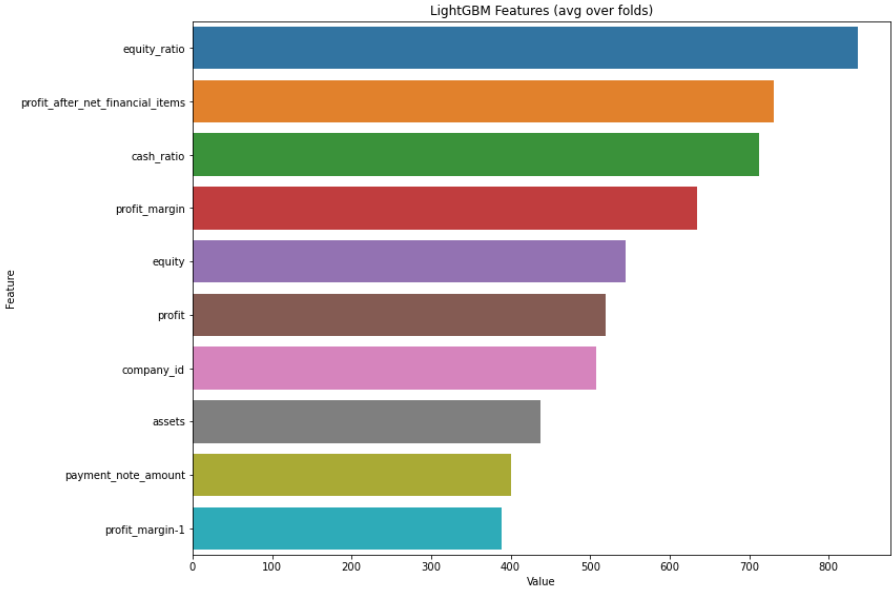

Figure 2: Feature importances from LightGBM model

I also found the SHAP values for this model by using the SHAP package and have included this below (Figure 3). SHAP uses a game theory approach to help explain the performance of a model and the influence of its feature better. I used the summary plot which goes one step above a feature importance chart by identifying the impact that each feature has on the models outputs. It seems from this chart that equity ratio has alrger effect on the AAA class then it does on the others while profit after net financial items has a more even effect across all classes.

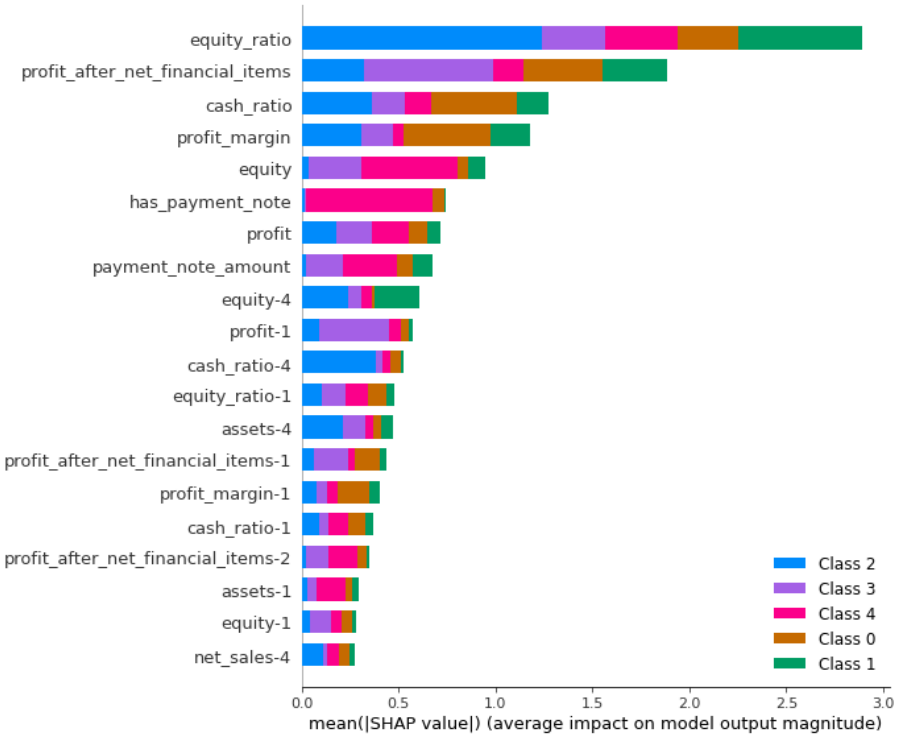

Figure 3: Shap values from LightGBM model

Finally I wrote the best performing model and its parameters to a pickle file. I then used the model to perform predictions on the test.csv file. I then wrote a streamlit app (app.py) to deploy my model to the web and allow a user to interact with the model for classification in either an online mode or a batch mode. In online mode the user can specify the values for each feature and then predict the outcome for a specific example and in batch mode the user can upload a csv to predict a large batch at once. I have included screenshots of the two modes below.

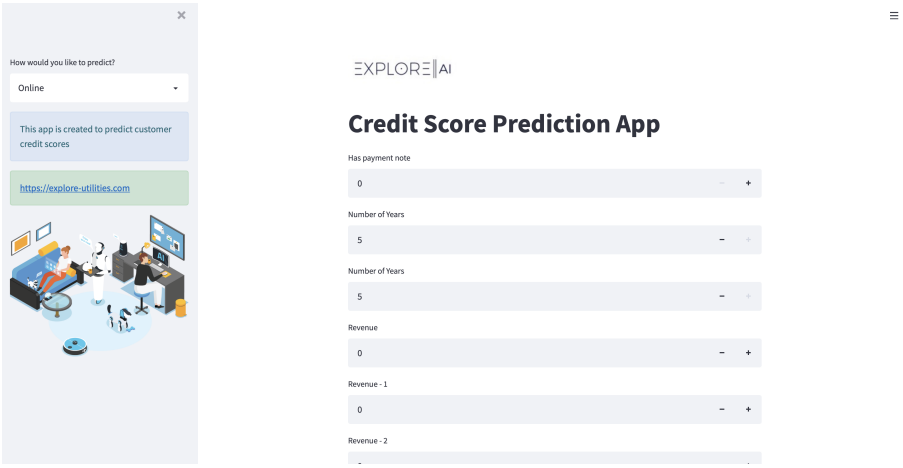

Figure 4: Online View From Web App

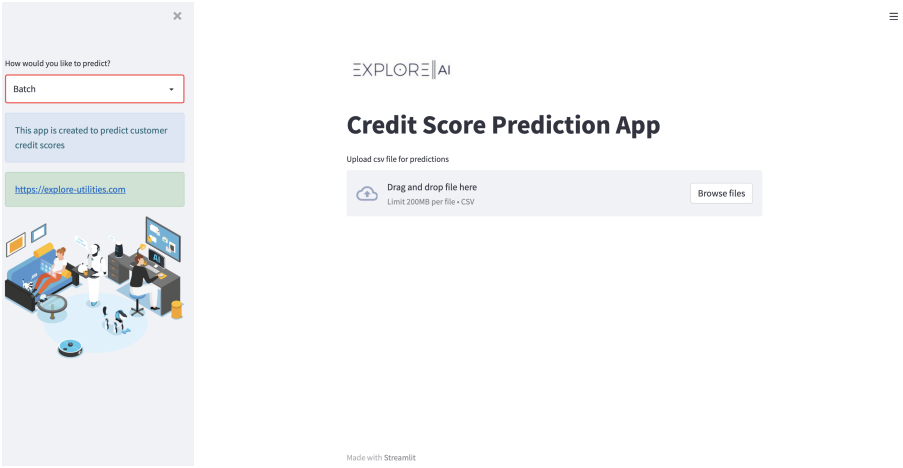

Figure 5: Batch View From Web App

Finally I packaged the web app into a docker image and deployed that image to Heroku.

Conclusion

I really enjoyed this task and hope that you will enjoy looking the report, code and playing with the web app as much as I have. Next steps to investigate would be using Github Actions to create a continuous integration / continuous deployment (CI/CD) pipeline so that we can retrain our model on the fly and

deploy it immediately into production. Another step I would add would be to use production model monitoring such as EvidentlyAI which can help identify model drift over time and indicate when it may be tine to retrain a model.