Prediction of the level of risk of a loan application

Bui Duy Hien - 20200208
Duong Duy Manh - 20205162

1 Introduction

Lending between individuals is expanding all across the world. The program establishes a connection between those in need of loans and those who can offer them. According to the degree of risk where the borrower cannot repay the loan, each loan will be given a grade ranging from A to G. The borrower's assessments, including credit histories, income, and other factors, will be included in the grading process. Additionally, there are 5 sub-grades inside each grade. The safest loan is A1, with a rate of about 6%. If the level of riskiness increases, the interest rate increases. In particular, the G5 offers a staggering interest rate of 26.06 percent. Investors adore this model since, despite the high-interest rate, lending a moderate amount of money is possible, and the process isn't as hazy and confusing as investing money into real estate or stocks. According to Lending Club, the borrowing community has received roughly 6 billion USD in funding, and the company has made a profit of about 600 million USD. The question here is: "Can we predict if a borrower would default or not in each specific situation?". If the company can properly address this query, it will be able to increase profit and prevent losses. We decided to develop a variety of machine learning models using the abundant data provided, with the goal of forecasting defaulters by identifying them as 0 if they are unable to pay the loan or 1 if they are able to.

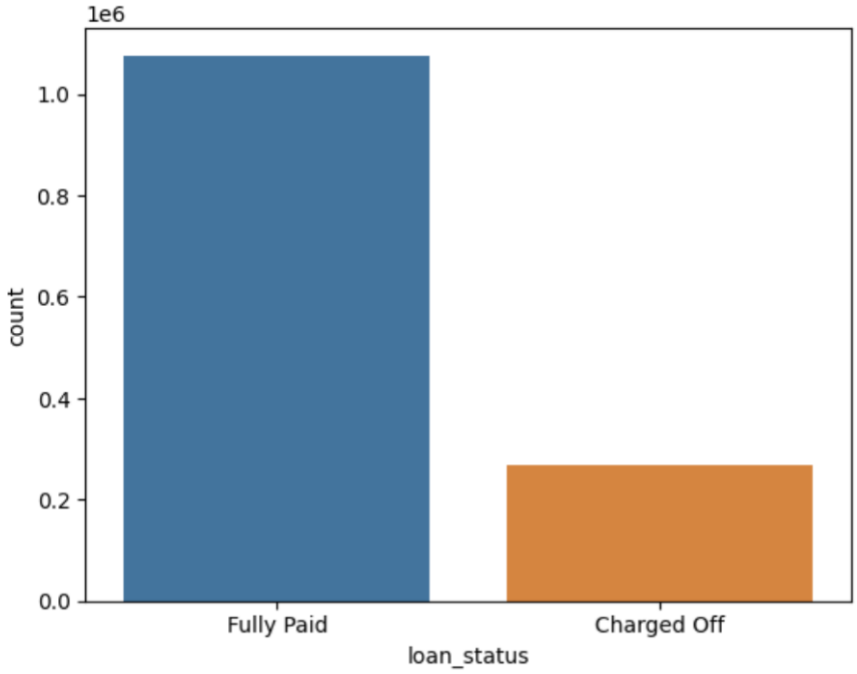
2 Data Overview

2.1 Dataset

We worked with a public data set in Kaggle. There are two different data sets: "accepted 2007 to 2018Q4.csv.gz" and rejected 2007 to 2018Q4.csv.gz. While the first set contains information about loans which were accepted from 2007 to 2018, the second set stores data about rejected offers in similar time periods. In our problem, we only consider the first one because the second doesn't provide useful information (all these applications don't satisfy basic conditions to gain a loan).

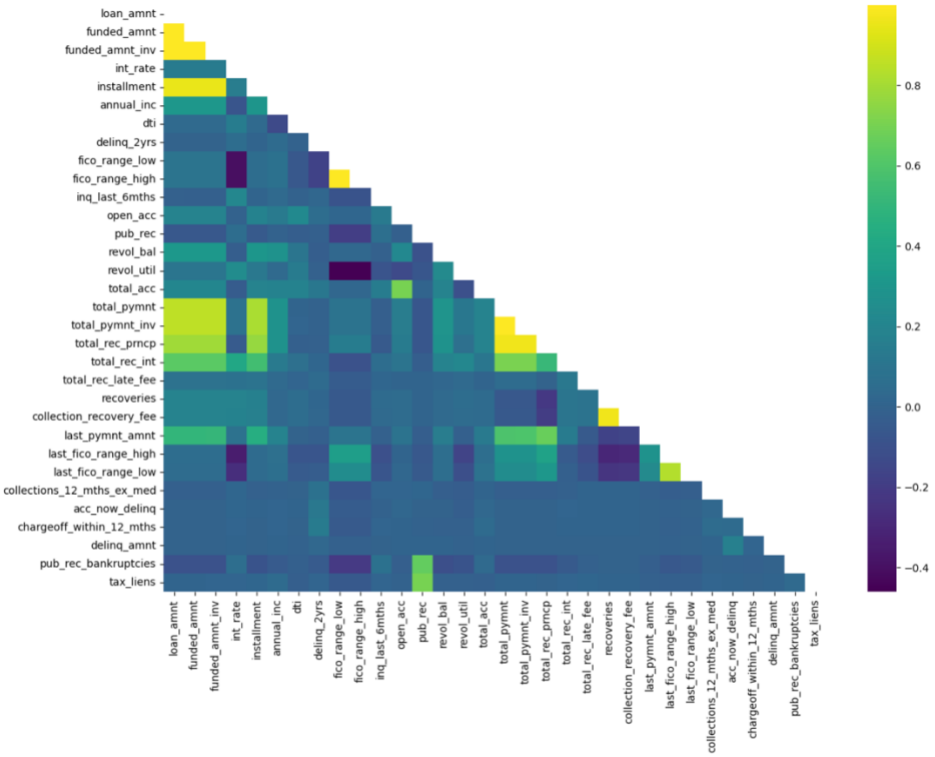
2.2 Data processing

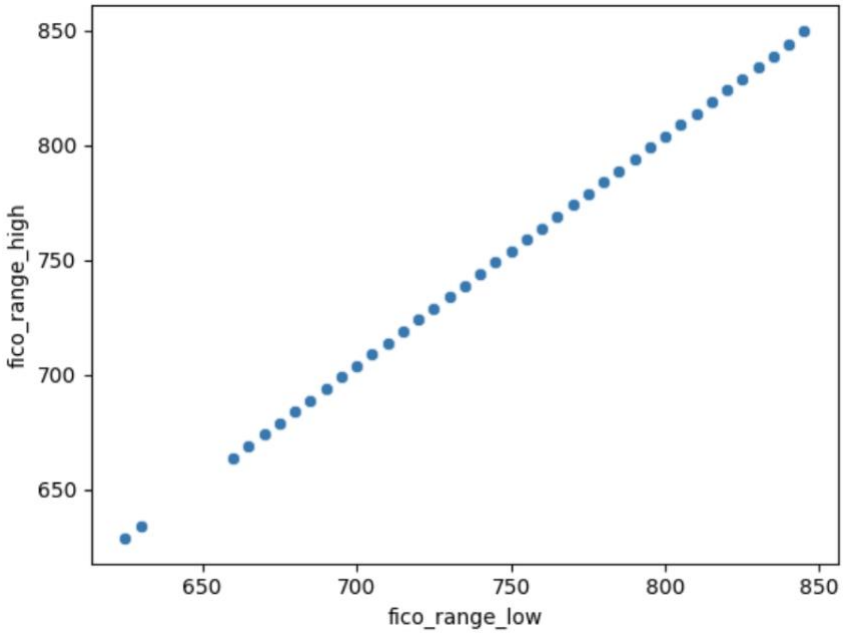
The original data set has over 2.2 million rows and 151 columns. We filtered out loans whose statuses are not yet final, such as "Current" and "Late (31-120 days)". We treat "Fully Paid" as our positive label, and "Default" as negative. Columns with empty values for most of the rows are dropped in order to have a cleaner data set (we only keep columns with less than 1% of missing values).



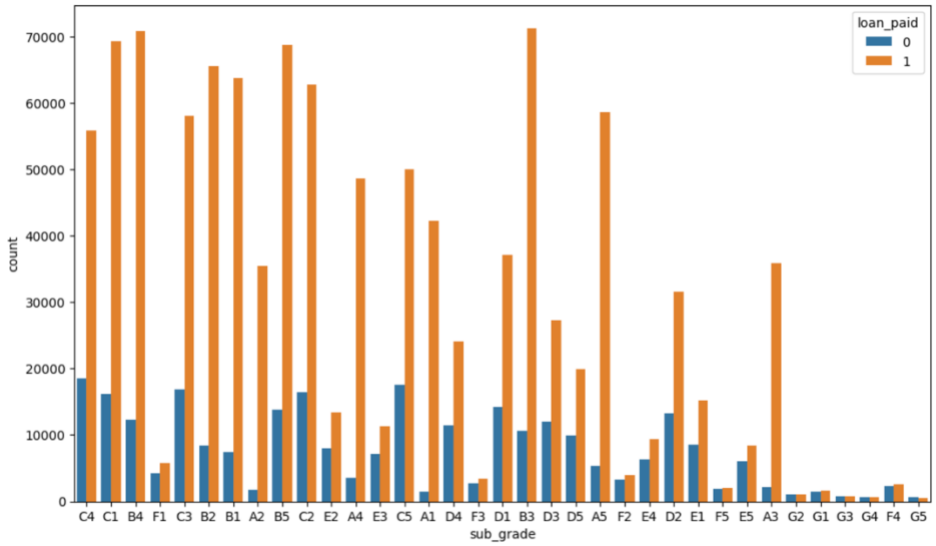
The first important step is to figure out which feature is leaking from the future, i.e. features can only be collected after the loan offer is accepted. If we dont remove these features, it can lead to over-optimistic classification results but our model will be not useful when applying it to real life. We found and removed some leakage features such as 'last pymnt d', 'last fico range low' and "last fico range high".

Considering continuous variables, we found and removed constant features ("out prncp", "out prncp inv" and "policy code") and highly correlated features ("fico range low" and "fico range high") by drawing correlation heat map between them.





After that, to deal with categorical variables, we decided to remove or make dummies variables based on their unique values. If they are constant variables or have too many unique values, we will drop these columns, otherwise, we will get dummy variables from them. For more details about which feature is kept or removed.



We then removed rows which have missing values in remaining features (they account for only less than 0.1% of the whole data set). Finally, we get the processed data set of size (1343380, 79), with 20% negative and 80% positive examples. In order to preserve the distribution of target variable, the data set were strategically split into train and test set with ratio 0.8:0.2.

3 Problem Overview

3.1 Handling imbalanced data

Most machine learning algorithms work best when the number of samples in each class are about equal. This is because most algorithms are designed to maximize accuracy and reduce errors. However, if the data set in imbalance then in such cases, you get a pretty high accuracy just by predicting the majority class, but you fail to capture the minority class, which is most often the point of creating the model in the first place. To handle this problem, we used a approache: oversampling (**SMOTE**).
Oversampling: Oversampling methods duplicate examples in the minority class or synthesize new examples from the examples in the minority class. The most popular and perhaps most successful oversampling method is SMOTE; that is an acronym for Synthetic Minority Oversampling Technique.
SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample as a point along that line. Specifically, a random example from the minority class is first chosen. Then k of the nearest neighbors for that example are found (typically k=5). A randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point between the two examples in feature space.
This approach is effective because new synthetic examples from the minority class are created that are plausible, that is, are relatively close in feature space to existing examples from the minority class.

A general downside of the approach is that synthetic examples are created without considering the majority class, possibly resulting in ambiguous examples if there is a strong overlap for the classes

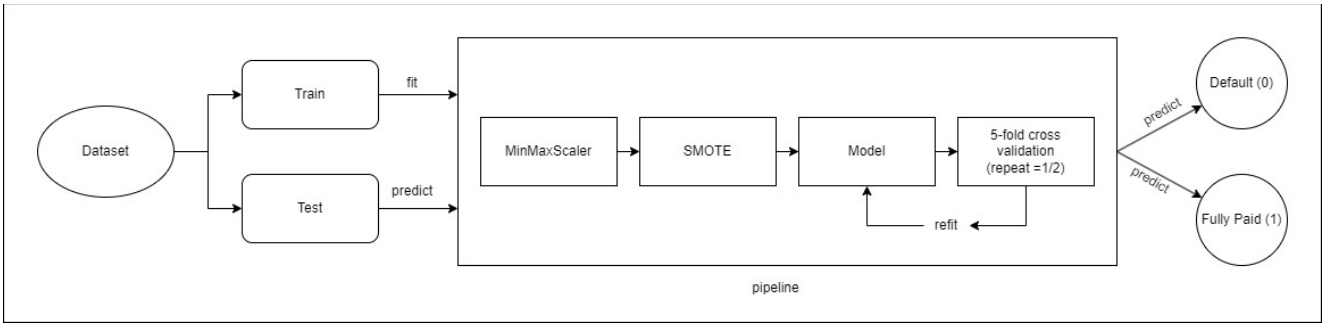
3.2 Evaluation metric

Since the problem is a binary problem with imbalanced target data, we chose F-Measure and AUC as the two performance metrics to evaluate our model. Besides, we also want to focus on reducing wrong predictions about non-paid applicants, thus, F0.5 - Measure and Precision-Recall AUC were our final evaluation metrics. F0.5 - Measure helps us concentrate on minimizing the false positive predicting, meanwhile, Precision - Recall AUC compare false positives to true positives rather than true negatives and thus won't be affected by the relative imbalance. Here is the Fbeta-Measure:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

3.3 Classification pipeline

The classification goal is to predict which class the loan belongs to: either 'Default' (0) or 'Fully Paid' . Our pipeline consist of two optional components, which are scaler and sampler, and a mandatory component, which is a classifier.
In terms of scaler, we use MinMaxScaler instead of StandardScaler due to many input variables doesn't satisfy normal distribution. Also, we only considered SMOTE as sampler in this project. That putting them to a pipeline helps us avoid leaking information from test/validation set to training (this often happen when we perform data preprocessing such as scaling or re-sampling all data set or even training data set, which can causes over-optimistic in testing results).
To minimize impacts of randomness and have consistent results, we fixed random state = 42 for all algorithms Further more, we used 5-fold cross validation when tuning the pipeline, repeated these task 2 times and captured all train/test scores as well as fit times. After that, the mean and standard deviation of each score were calculated and used for comparing models to find the best pipeline.



Moreover, because our data set is fairly large with over 1.3 million examples, it takes much time to tune hyperparameters if we use exhaustive GridSearch. After experiments, we found that HalvingGridSeach is much faster than GridSearch while the performance of models stayed highly.
HalvingGridSearch is like a competition among all candidates (hyperparameter combinations). While Grid- Search trains the candidates on all of the training data, HalvingGridSearch takes a different approach called successive halving. In the first iteration, HalvingGridSearch trains all candidates on a small proportion of the training data. In the next iteration, only the candidates which performed best are chosen and they will be given more resources to compete. So, with each passing iteration, the 'surviving' candidates will be given more and more resources (training samples) until the best set of hyperparameters are left standing.

3.4 Models

*) Logistic Regression

Logistic regression is a classification algorithm used to find the probability of event success and event failure. It is used when the dependent variable is binary(0/1, True/False, Yes/No) in nature. It supports categorizing data into discrete classes by studying the relationship from a given set of labelled data. It learns a linear relationship from the given data set and then introduces a non-linearity in the form of the Sigmoid function where output is probability and input can be from -infinity to +infinity.

Advantages:

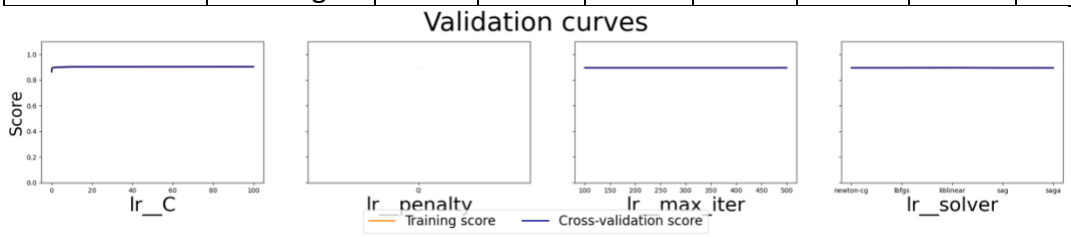
- Logistic regression is easier to implement, interpret, and very efficient to train and predict.
- It makes no assumptions about distributions of classes in feature space.
- Good accuracy for many simple data sets and it performs well when the dataset is linearly separable.
- It can interpret model coefficients as indicators of feature importance.
- Logistic regression is less inclined to over-fitting but it can overfit in high dimensional data sets. One may consider Regularization (L1 and L2) techniques to avoid over-fitting in these scenarios.

Disadvantages:

- The major limitation of Logistic Regression is the assumption of linearity between the dependent variable and the independent variables. So, non-linear problems can't be solved with logistic regression because it has a linear decision surface. Linearly separable data is rarely found in real-world scenarios.
- Logistic Regression requires average or no multicollinearity between independent variables.

To test the effects of various improvement techniques, a pipeline will be set up. First, seeing the difference between a scaled and non-scaled base model; after that, SMOTE will be used, then hyperparameter tuning will be used to get a better outcome. The following are the results of the pipeline described above:

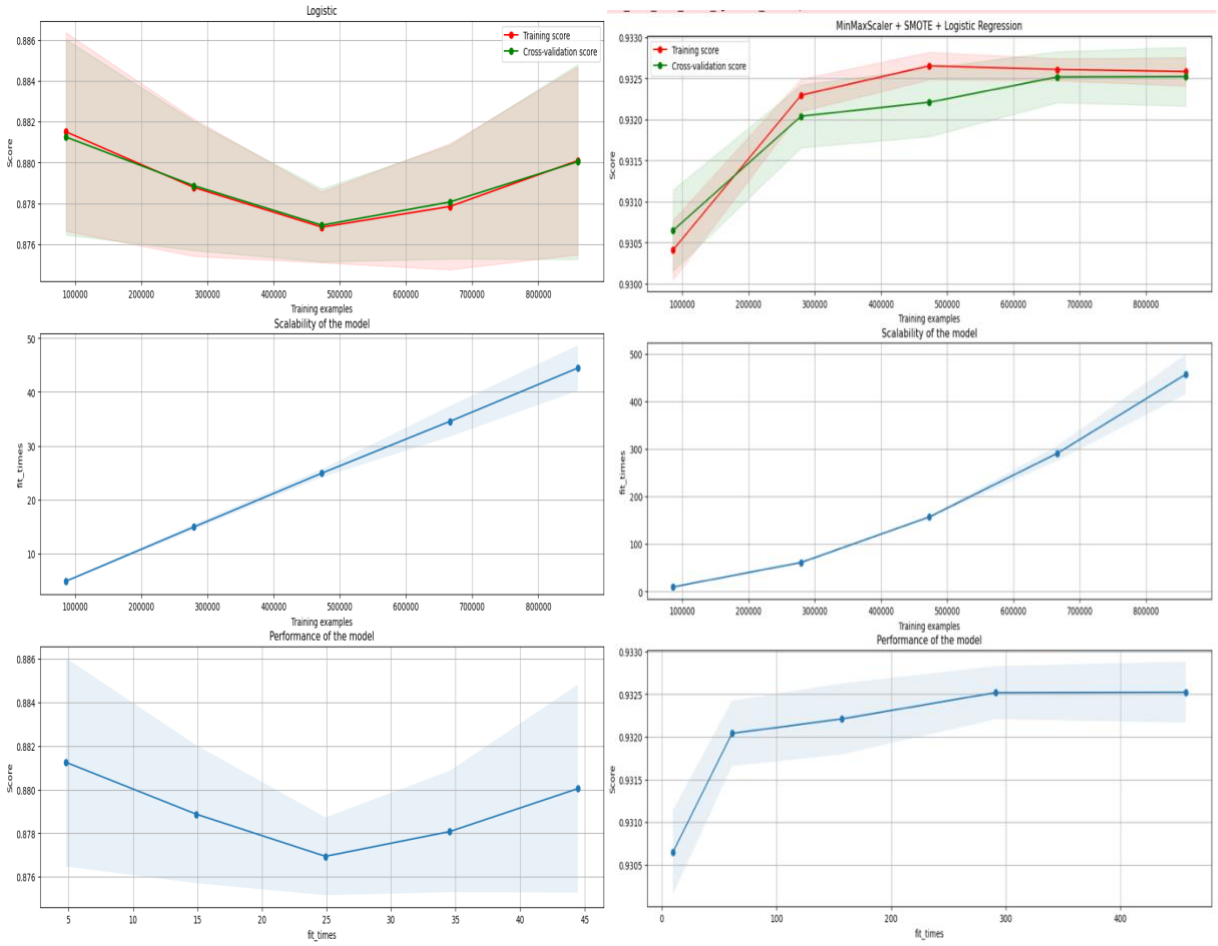
Logistic Regression	Parameters	mean train score	Std train score	Mean test score	Std test score	Mean fit time	Std fit time	f0.5 score	pr auc score
Base		0.879165	0.004624	0.879105	0.004847	23.890243	0.285269	0.876595	0.971961
MinMaxScaler+ Base		0.903176	0.000202	0.903139	0.000621	22.246391	0.352062	0.9029236	0.981731
MinMaxScaler+ SMOTE + Base		0.93257	0.000136	0.932542	0.000604	143.104936	1.880894	0.9325637	0.9817184
MinMaxScaler + Base (Tuning)	C=100, max_iter=500, solver='newton-cg'	0.904020	0.000454	0.904211	0.001057	42.898820	12.784984	0.903604	0.981787



Base model vs best model:

```
Best parameter (CV score=):
Pipeline(steps=[('lr', LogisticRegression())])
              precision    recall  f1-score   support

           0       0.67      0.37      0.48     53638
           1       0.86      0.95      0.90    215038

    accuracy                           0.84    268676
   macro avg       0.76      0.66      0.69    268676
weighted avg       0.82      0.84      0.82    268676

[[ 20086  33552]
 [ 10068 204970]]
f0.5_score= 0.8765949948936212
pr_auc_score= 0.9719612634317316
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
              precision    recall  f1-score   support

           0       0.52      0.94      0.67     53638
           1       0.98      0.78      0.87    215038

    accuracy                           0.81    268676
   macro avg       0.75      0.86      0.77    268676
weighted avg       0.89      0.81      0.83    268676

[[ 50217   3421]
 [ 47054 167984]]
f0.5_score= 0.932562637538333
pr_auc_score= 0.981718393276838
```



***) Decision Tree**

Decision tree is a non-parametric supervised learning method which is very powerful for classification problems. With the goal to create predictions for the value of a target variable, this algorithm learns simple decision rules(like if -else rule) inferred from the data features. Furthermore, decision tree is the base algorithm for other advanced decision-making and ensemble learning techniques.

Advantage

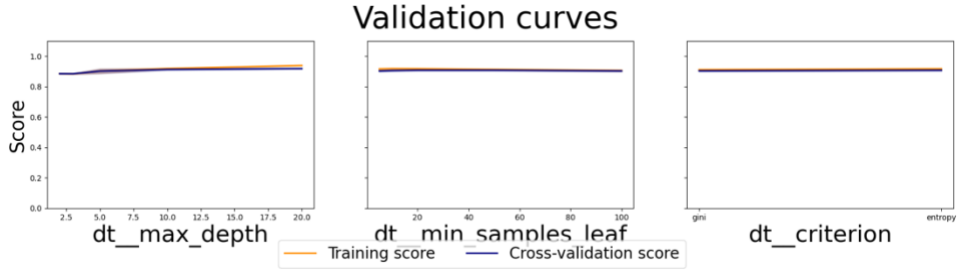
- Robust to noise, requires little data preparation
- Fast time running.
- Low computational cost.

Disadvantage

- Unstable nature
- Less effective in predicting the outcome of a continuous variable

Then, we try to arrive at optimal hyperparameters for Decision Tree (In here, we try to find the optimal maximum depth of tree, the optimal minimum number of samples required to be at a leaf node and the optimal criteria measure function). After training, we obtain these results:

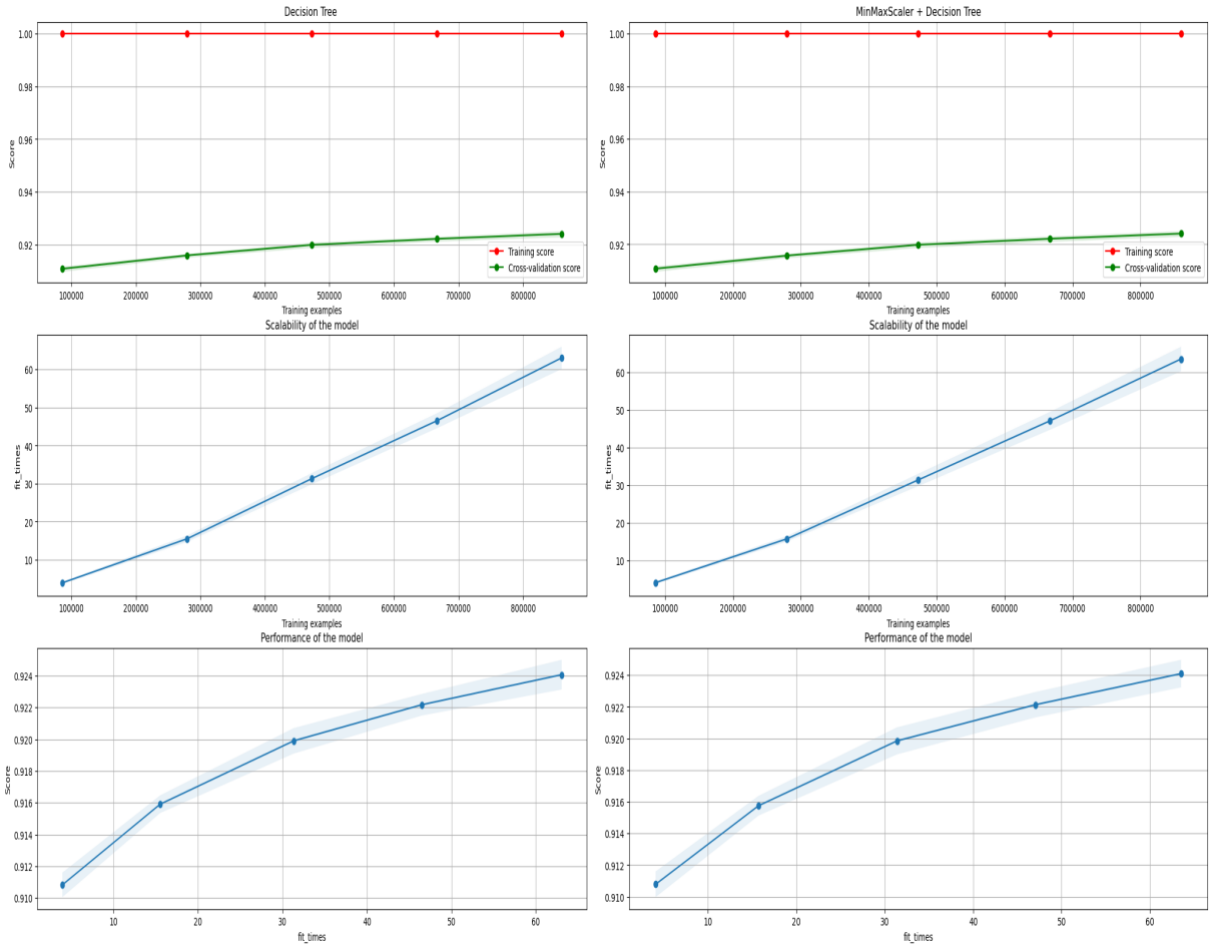
Decision Tree	Parameters	mean train score	Std train score	Mean test score	Std test score	Mean fit time	Std fit time	f0.5 score	pr auc score
Base		1.0	0.0	0.924054	0.00081	46.863223	2.819843	0.92582661	0.95557646
MinMaxScaler+ Base		1.0	0.0	0.924052	0.000901	47.646974	1.628248	0.9258558	0.9556
MinMaxScaler+ SMOTE + Base		1.0	0.0	0.913945	0.001209	208.074827	4.0287	0.914774737	0.949302
MinMaxScaler + Base (Tuning)	max_depth=20, min_samples leaf=20, criterion='gini'	0.904022	0.000109	0.903961	0.000561	138.248221	16.668730	0.903604	0.981787



Base model vs best model:

```
Best parameter (CV score=):
Pipeline(steps=[('dt', DecisionTreeClassifier())])
              precision    recall  f1-score   support

           0       0.70      0.71      0.70     53638
           1       0.93      0.92      0.92    215038

    accuracy                           0.88    268676
   macro avg       0.81      0.81      0.81    268676
weighted avg       0.88      0.88      0.88    268676

[[ 37885  15753]
 [ 16513 198525]]
f0.5_score= 0.9258266100825444
pr_auc_score= 0.9555764593308762
```

```
Best parameter (CV score=):
Pipeline(steps=[('minmax', MinMaxScaler()), ('dt', DecisionTreeClassifier())])
              precision    recall  f1-score   support

           0       0.70      0.71      0.70     53638
           1       0.93      0.92      0.92    215038

    accuracy                           0.88    268676
   macro avg       0.81      0.81      0.81    268676
weighted avg       0.88      0.88      0.88    268676

[[ 37905  15733]
 [ 16546 198492]]
f0.5_score= 0.9258557864354094
pr_auc_score= 0.9555987267224878
```



***) XGBoost**

Extreme Gradient Boosting, also known as XGBoost, is a powerful ensemble learning technique that uses two key technologies: Gradient Boosting and Decision Trees. Gradient Boosting works by training each predictor sequentially using its predecessor's error. XGBoost is very much the same as Gradient Boosting, with each weak predictor being a decision tree.

Advantages:

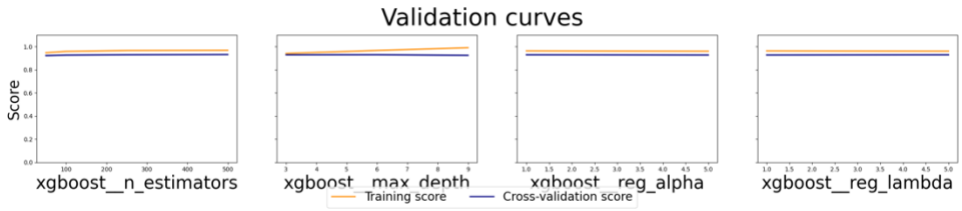
- XGBoost has a parallel processing mechanism that can speed up its computation time.
- When the model is given the right hyperparameters, it can provide amazing results.

Disadvantages:

- On sparse, chaotic, and noisy data, XGBoost does not perform as well. This is due to the classifier in Gradient Boosting being compelled to correct the mistakes made by the preceding learners.

To test the effects of various improvement techniques, a pipeline will be set up. First, seeing the difference between a scaled and non-scaled base model; after that, SMOTE will be used, then hyperparameter tuning will be used to get a better outcome. The following are the results of the pipeline described above:

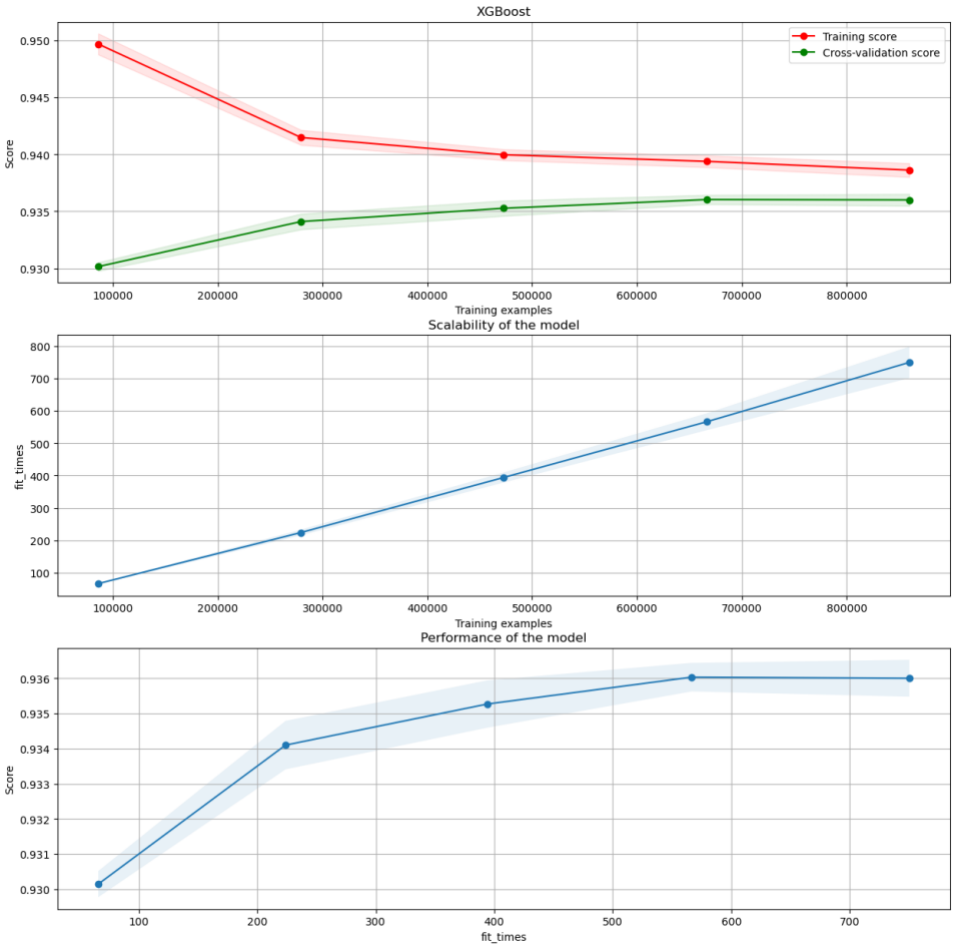
XGBoost	Parameters	mean train score	Std train score	Mean test score	Std test score	Mean fit time	Std fit time	f0.5 score	pr auc score
Base		0.938512	0.000564	0.935891	0.000603	180.950729	4.053333	0.93592	0.99044567
MinMaxScaler+ Base		0.938512	0.000564	0.935892	0.000603	187.579811	1.828526	0.935915	0.99044568
MinMaxScaler+ SMOTE + Base		0.936914	0.0003	0.934959	0.000646	484.756524	5.293513	0.934637	0.9891555
MinMaxScaler + Base (Tuning)	max_depth=6 n_estimators=500, reg alpha=1.0, reg lambda=1.0	0.956559	0.000463	0.947280	0.000478	108.509162	2.992077	0.9488374	0.99310503



Base model vs best model:

```
Best parameter (CV score=):
Pipeline(steps=[('minmax', MinMaxScaler()),
                ('xgboost',
                 XGBClassifier(base_score=None, booster=None, callbacks=None,
                               colsample_bylevel=None, colsample_bynode=None,
                               colsample_bytree=None,
                               early_stopping_rounds=None,
                               enable_categorical=False, eval_metric=None,
                               feature_types=None, gamma=None, gpu_id=None,
                               grow_policy=None, importance_type=None,
                               interaction_constraints=None, learning_rate=None,
                               max_bin=None, max_cat_threshold=None,
                               max_cat_to_onehot=None, max_delta_step=None,
                               max_depth=6, max_leaves=None,
                               min_child_weight=None, missing=nan,
                               monotone_constraints=None, n_estimators=500,
                               n_jobs=None, num_parallel_tree=None,
                               predictor=None, random_state=None, ...))])
              precision    recall  f1-score   support

           0       0.85      0.77      0.81     53638
           1       0.94      0.97      0.96    215038

    accuracy                           0.93    268676
   macro avg       0.90      0.87      0.88    268676
...
[[ 41490  12148]
 [  7391 207647]]
f0.5_score= 0.9488374345879889
pr_auc_score= 0.993105031739742
```

```
Best parameter (CV score=):
Pipeline(steps=[('xgboost',
                 XGBClassifier(base_score=None, booster=None, callbacks=None,
                               colsample_bylevel=None, colsample_bynode=None,
                               colsample_bytree=None,
                               early_stopping_rounds=None,
                               enable_categorical=False, eval_metric=None,
                               feature_types=None, gamma=None, gpu_id=None,
                               grow_policy=None, importance_type=None,
                               interaction_constraints=None, learning_rate=None,
                               max_bin=None, max_cat_threshold=None,
                               max_cat_to_onehot=None, max_delta_step=None,
                               max_depth=None, max_leaves=None,
                               min_child_weight=None, missing=nan,
                               monotone_constraints=None, n_estimators=100,
                               n_jobs=None, num_parallel_tree=None,
                               predictor=None, random_state=None, ...))])
              precision    recall  f1-score   support

           0       0.81      0.71      0.76     53638
           1       0.93      0.96      0.94    215038

    accuracy                           0.91    268676
   macro avg       0.87      0.84      0.85    268676
weighted avg       0.91      0.91      0.91    268676
...
[[ 38217  15421]
 [  8890 206148]]
f0.5_score= 0.9359183666057094
pr_auc_score= 0.9904456702196608
```



4 Conclusion and future work

4.1 Conclusion

So far, we have introduced and tackle our classification problem by using different pipelines, both in data analysis and machine-learning model. For clarity, in data analysis task, we handle the categorical features, treat the data imbalance as well as do some research to comprehend the domain knowledge of our problem, i.e. we have to understand the role of every features, thus helping us to discover some leaky features, which are pretty hard to be explored only by normal analysis. And as you can see in the tables, the performance of the best model, the XGBoost, satisfies our demand really well.

4.2 Future work

Experiment with other powerful ensemble models such as LightGBM and CatBoost.
Using focal loss instead of cross entropy in deep learning models like ANN and DNN to overcome imbalanced data set.

5 List of tasks

Bui Duy Hien: Data processing, Logistic Regression
Duong Duy Manh: Decision Tree, XGBoost

References

https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification
https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/
https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/
https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/
https://www.kaggle.com/datasets/wordsforthewise/lending-club
https://xgboost.readthedocs.io/en/stable/
https://www.anyscale.com/blog/how-to-tune-hyperparameters-on-xgboost
https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/