LOAN DEFAULT PREDICTION FOR PROFIT MAXIMIZATION

Nguyen Ngoc Dung 20204905 Nguyen Huy Hai 20200194

Duong Vu Tuan Minh 20209705 Nguyen Hai Long 20204920

July 22, 2022

Abstract

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. The information was obtained from [7]. Following a rigorous exploration, analysis, and preprocessing process, our data frame has more than 1.3 million tuples, each of which contains 78 features. We run several algorithms, evaluate them, and draw some conclusions. We finally get the best result of XGBoost with f0.5score=0.949691 and pr_auc=0.993086.

1 Introduction

Identifying defaulters is a crucial step in any lending procedure. This action assists both the business and lenders in preventing financial loss. Given the enormous amount of data, this problem contains, it ideally suits the settings of a machine learning classifier. Data exploration and preprocessing are done early on in the task. Plots are created to visualize the relationship and influence of the features after removing columns with an excessive amount of missing data. Leaky data and outlier points are also found and eliminated from the data frame. We ran machine learning models on the elite data set after gaining a general understanding of the data. We use a total of 8 algorithms in the second phase of the solve, including the following: Logistic Regression, Decision Tree, Support Vector Machine, AdaBoost, XGBoost, Random Forest, Perceptron and MLP. Each of these algorithms was selected for a valid reason that could work with the problem's context. Each algorithm will be added to a pipeline that also incorporates additional methods for enhancing performance, such as scaling and oversampling. Additionally, hyperparameter-tuning will be offered to enable a detailed comparison between an algorithm that has been tuned and one that has not. The results are then compared, and conclusions are produced to reinforce the initial attempts and identify good models in terms of accuracy and timeliness.

2 Related work

There are numerous related works and papers that are relevant to the assignment that we have encountered while working on this project. A well-known example is Fayes Sayah's notebook, which can be seen at Kaggle. This notebook is presently the most popular in the group with 163 upvotes. The author has produced a large number of works that really shed light on this project. In order to fulfill the objectives, various strategies are introduced, such as outlier removal and data scaling. In this notebook, three models—the ANN, the XGBoost Classifier, and the Random Forest Classifier—are run. Some elements of this notebook, meanwhile, don't fully match the nature of the issue. While our technique uses the f0.5 score, an abstract measurement of the f-beta score, the author uses the f1 score as the primary prediction evaluation metric. This measure directs our concentration toward lowering false positives, which will result in fewer false positive predictions from our model and better financial results overall.

Furthermore, without explaining the reason, Fayes uses directly the dataset that has already been filtered to remove any leaky columns. To sum up, despite all the issues we discovered, this notebook succeeds at data discovery and preprocessing; using the f1 score, all models reach an accuracy of about 88 percent, which is pretty bright.

3 Data Overview

3.1 Introduction to the data set

We worked with a public data set in Kaggle [7]. 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).

3.2 EDA & Feature Engineering

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 don"t remove these features, it can lead to over-optimistic classification results but our model will be not useful when applying it to real life. By studying many resources [2], 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.

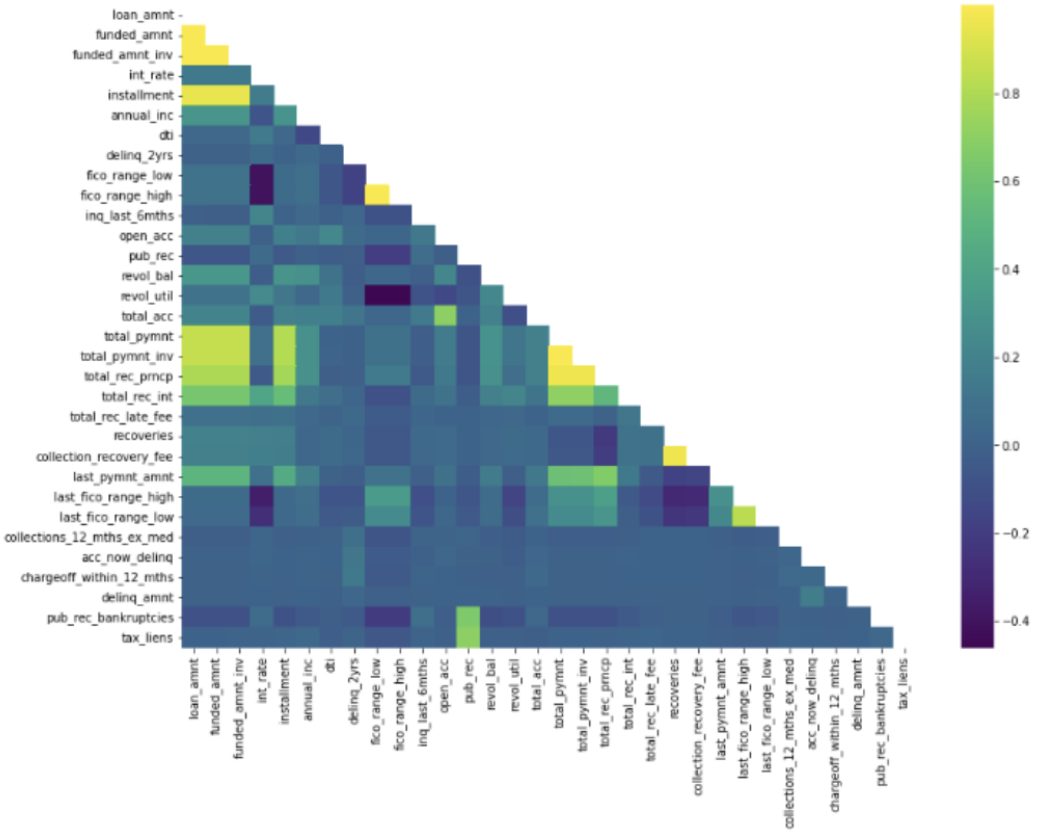

Figure 1: Correlation Lending Club Loan

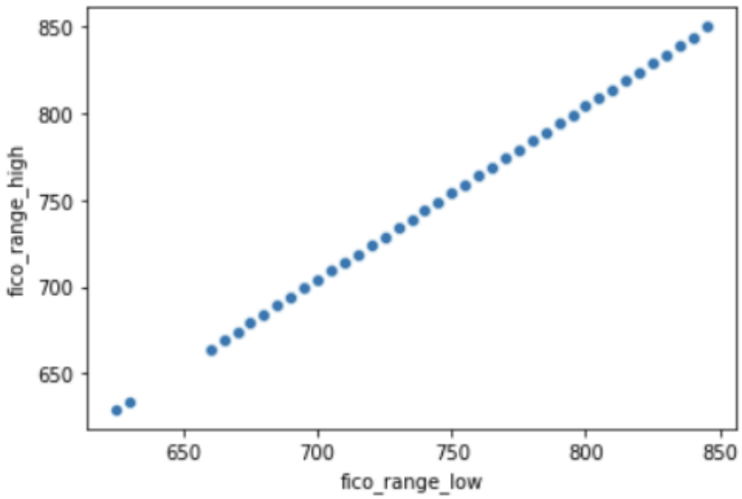

Figure 2: Correlation between 2 features

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.

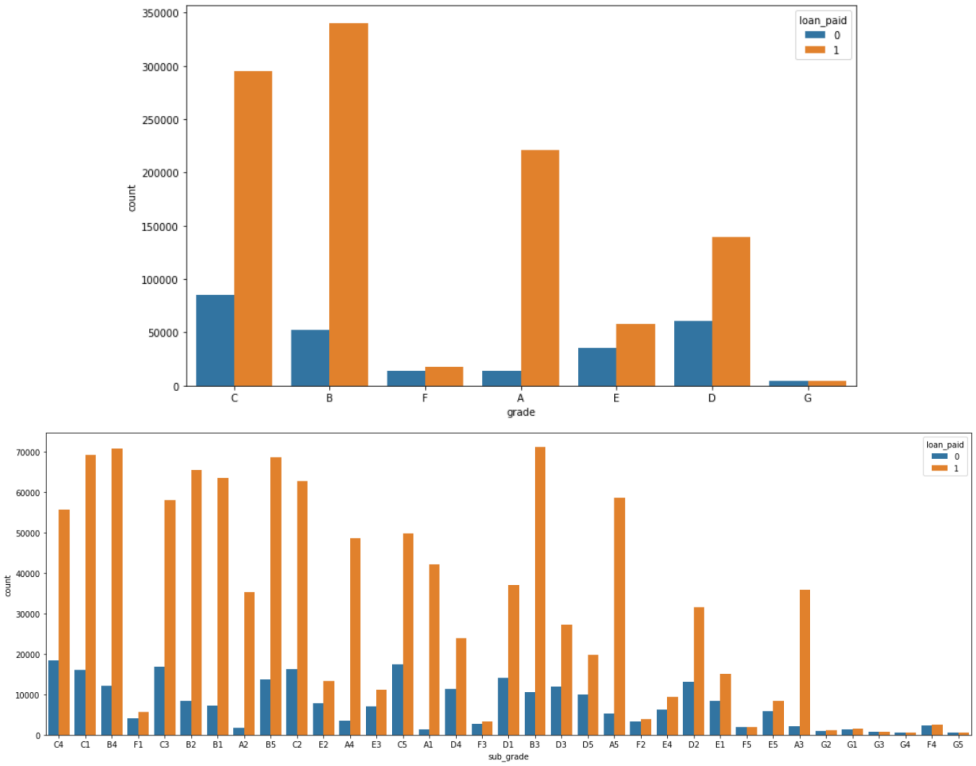

Figure 3: Grade and sub-grade

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.

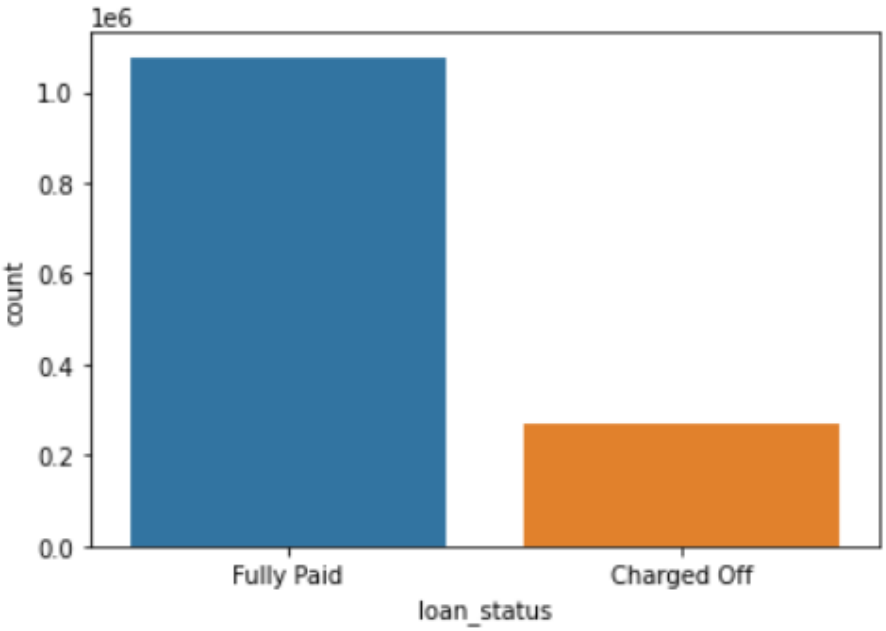

Figure 4: Distribution of target variable

4 Problem Overview

4.1 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 two approaches: adjusting class weight and oversampling.

Adjusting class weight: We used this method with Logistic Regression, Support Vector Machine, Decision Tree and Random Forest. Despite class weight has different meaning and math formula in each model, its impact is to help the model focus more on minority class rather than the majority one. One well known strategy of class weight is 'balanced', the model automatically assigns the class weights inversely proportional to their respective frequencies. To be more precise, the formula to calculate this is:

$$w_j = n\_samples/(n\_classes * n\_samples_j)$$

Where:

- w_j is the weight for each class(j signifies the class)
- $n\_samples$ is the total number of samples or rows in the dataset
- $n\_classes$ is the total number of unique classes in the target
- $n\_samples_j$ is the total number of rows of the respective class

For more details about math formula and meaning of class weight in each model, refer [1].

Oversampling: Oversampling methods duplicate examples in the minority class or synthesize new examples from the examples in the minority class.Some of the more widely used and implemented oversampling methods include:

- Random Oversampling
- Synthetic Minority Oversampling Technique (SMOTE)
- Borderline-SMOTE
- Borderline Oversampling with SVM
- Adaptive Synthetic Sampling (ADASYN)

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.

For more details information, refer [1]

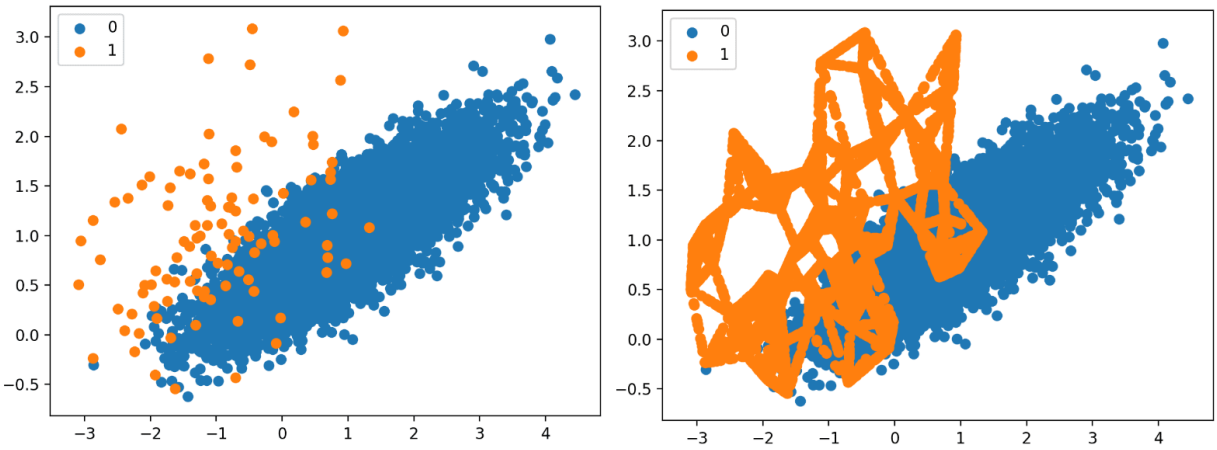

Figure 5: Imbalance data before (left) and after (right) applying SMOTE technique

4.2 Evaluation Metrics

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[4]. F0.5 - Measure helps us concentrate on minimizing the false positive predicting[2], meanwhile, Precision - Recall AUC compare false positives to true positives rather than true negatives and thus won't be affected by the relative imbalance[3]. Here is the Fbeta-Measure:

- **Fbeta-Measure** = ((1 + beta^2) * Precision * Recall) / (beta^2 * Precision + Recall)

Figure 6: Fomula of Fbeta

As you seen from the above formula, Beta indicating how much more important recall is than precision. For example, if we consider recall to be twice as important as precision, we can set Beta to 0.5. The standard F-score is equivalent to setting Beta to one.

4.3 Classification Pipeline

The classification goal is to predict which class the loan belongs to: either 'Default' (0) or 'Fully Paid' (1). 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 and its performance was compared to use different class weight. 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.

As SMOTE is time-consuming, typically when the size of data set is large, in some classifies requiring long fit time such as SVM and RandomForest, we only ran 5-fold cross validation one time.

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 GridSearch 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.

4.4 Classical 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/class weight 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		0.875301	0.00145	0.875276	0.0011	26.256346	1.61895	0.872807	0.974491
MinMaxScaler + Base		0.904226	0.000221	0.904201	0.000741	24.521906	1.83919	0.903247	0.981305
MinMaxScaler + SMOTE + Base		0.931736	0.000175	0.931689	0.000418	710.393442	14.557333	0.931779	0.981423
MinMaxScaler + Base (Tuning)	C=100, max_iter=1000, solver='sag'	0.905356	0.000745	0.905214	0.001680	12.082988	2.275316	0.904805	0.981496
MinMaxScaler + (SMOTE + Base) Tuning	sampling strategy=0.8, C=100, max_iter=500, solver='newton-cg'	0.933286	0.000268	0.933425	0.000410	133.473057	24.914387	0.933435	0.981504
MinMaxScaler + (class weight + Base) Tuning	C=100, class_weight= {0: 3.2, 1: 1.0}, solver='saga'	0.933255	0.000107	0.933256	0.000357	34.047845	8.343945	0.933325	0.981580

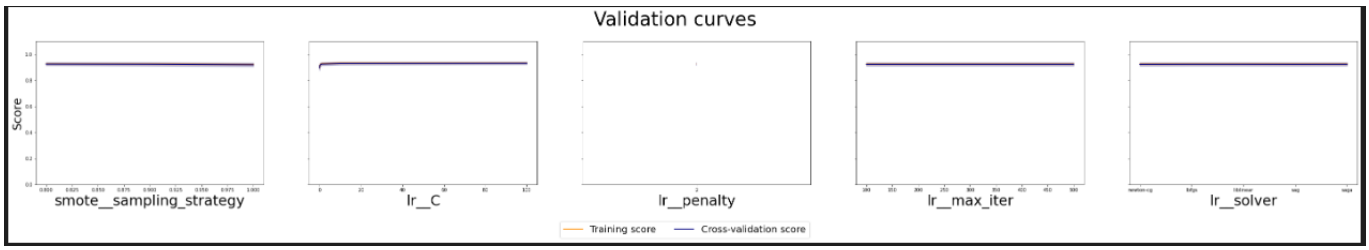

Figure 7: Validation curves

	precision	recall	f1-score	support
0	0.66	0.35	0.45	53638
1	0.85	0.96	0.90	215038
accuracy			0.83	268676
macro avg	0.76	0.65	0.68	268676
weighted avg	0.82	0.83	0.81	268676

```
[[ 18562  35076]
 [  9477 205561]]
f0.5_score= 0.8728067419279781
pr_auc_score= 0.9744911999251716
```

0	0.54	0.91	0.68	53638
1	0.97	0.81	0.88	215038
accuracy			0.83	268676
macro avg	0.75	0.86	0.78	268676
weighted avg	0.89	0.83	0.84	268676

```
[[ 48627   5011]
 [ 41745 173293]]
f0.5_score= 0.9334352450945539
pr_auc_score= 0.9815043674359029
```

Figure 8: Confusion matrices of best model (left) and base model (right) of Logistic Regression

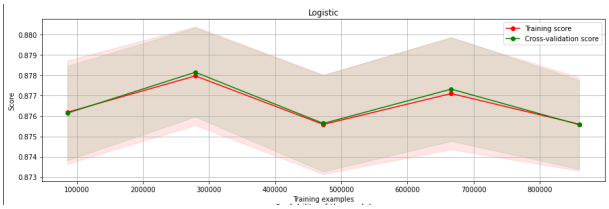

Figure 9: Score of best model (left) and base model (right) of Regression

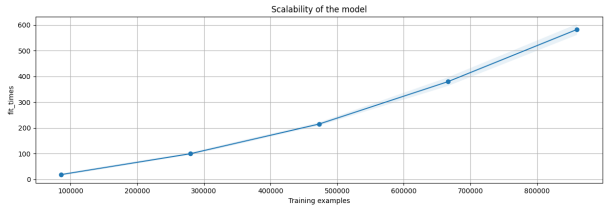

Figure 10: Scalability of best model (left) and base model (right) of Regression

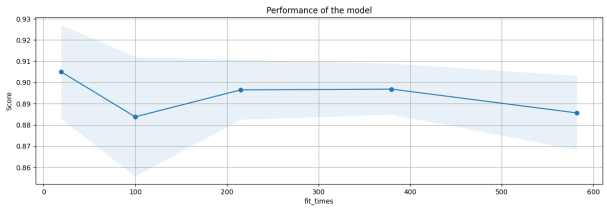

Figure 11: Performance of best model (left) and base model (right) of Regression

Perceptron

The perceptron is the building block of artificial neural networks, it is a simplified model of the biological neurons in our brain. A perceptron is the simplest neural network, one that is comprised of just one neuron.

The perceptron is a simplified model of the real neuron that attempts to imitate it by the following process: it takes the input signals, computes a weighted sum of those inputs, then passes it through a threshold function and outputs the result.

Advantages:

- It can deal with real-valued inputs (which makes it more useful and generalized) and boolean values.

Disadvantages:

- The output of a perceptron can only be a binary number (0 or 1) due to the hard limit transfer function.
- The perceptron can only be used to classify the linearly separable sets of input vectors. If input vectors are non-linear, it is not easy to classify them properly.

Applying the same pipeline, which is used for Logistics Regression, we obtain the following results:

Perceptron	Parameters	mean train score	std train score	mean test score	std test score	mean fit time	std fit time	f0.5 score
Base		0.874516	0.036919	0.874598	0.037038	13.454261	0.521713	0.846232
MinMaxScaler + Base		0.887178	0.019047	0.887251	0.019246	13.86338	0.106149	0.836828
MinMaxScaler + SMOTE + Base		0.897841	0.013538	0.897772	0.01365	527.701653	17.297408	0.899689
MinMaxScaler + Base (Tuning)	perceptron penalty='l1'	0.894408	0.008845	0.864685	0.008909	329.646469	43.775573	0.858140
MinMaxScaler + (SMOTE + Base) Tuning	perceptron penalty='l1', smote sampling strategy=0.8	0.914760	0.005980	0.915667	0.006923	12.414894	0.674265	0.916345

Theoretically speaking, the SMOTE tries to balance the labels of the data set in order to improve the performance of a model, and now it works in the practise of the perceptron. Obviously, our preceptron model that uses SMOTE performs really well, which is shown in column f0.5 score. Furthermore, after spending time tuning the parameter of the model "MinMaxScaler + SMOTE + Perceptron", we gain a much better result compared to the base model (0.91 and 0.84).

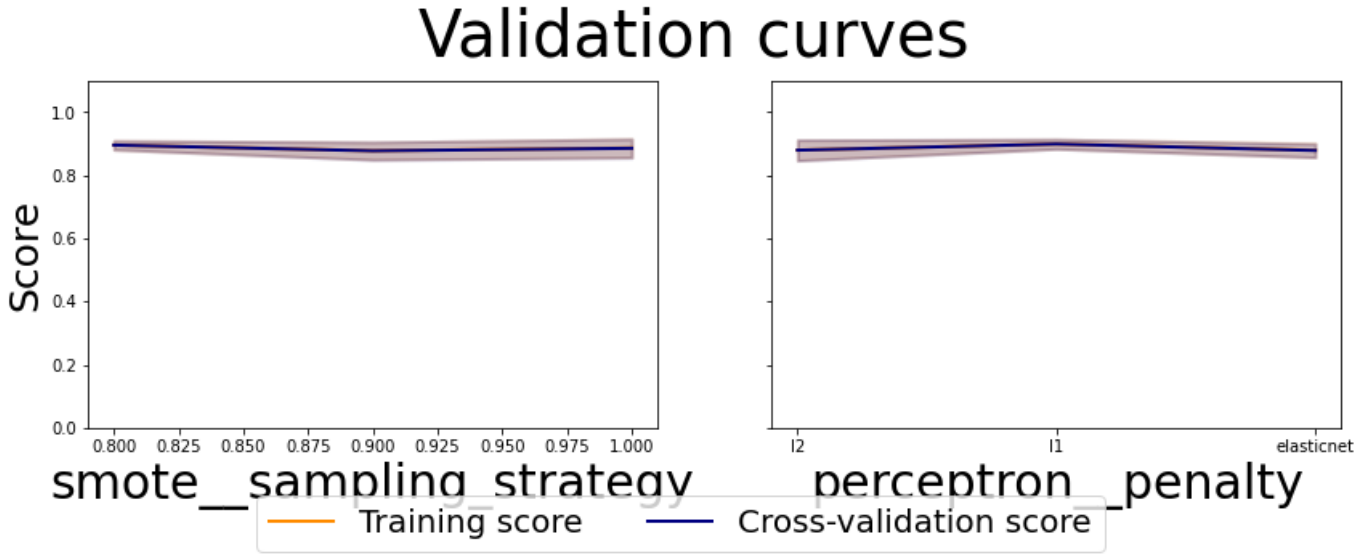

Figure 12: Validation curves

	precision	recall	f1-score	support
0	0.56	0.76	0.65	53638
1	0.93	0.85	0.89	215038
accuracy			0.83	268676
macro avg	0.75	0.80	0.77	268676
weighted avg	0.86	0.83	0.84	268676

```
[[ 40526  13112]
 [ 31383 183655]]
f0.5_score= 0.9163451770571176
```

	precision	recall	f1-score	support
0	0.68	0.12	0.20	53638
1	0.82	0.99	0.89	215038
accuracy			0.81	268676
macro avg	0.75	0.55	0.55	268676
weighted avg	0.79	0.81	0.75	268676

```
[[  6202  47436]
 [  2948 212090]]
f0.5_score= 0.846232908960038
```

Figure 13: Confusion matrices of best model (left) and base model (right) of Perceptron

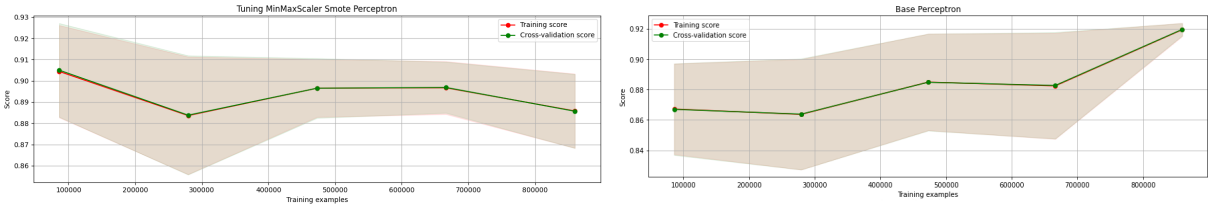

Figure 14: Score of best model (left) and base model (right) of Perceptron

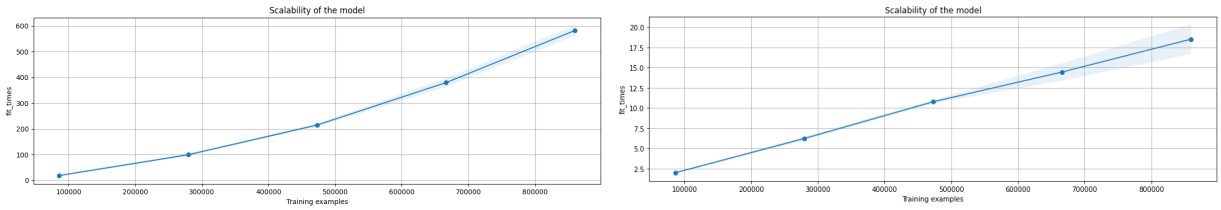

Figure 15: Scalability of best model (left) and base model (right) of Perceptron

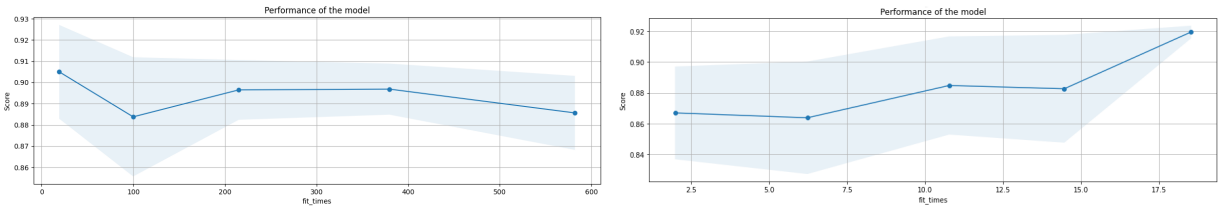

Figure 16: Performance of best model (left) and base model (right) of Perceptron

SVM

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection. A support vector machine constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space, which can be used for classification, regression or other tasks. Intuitively, a good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier. The figure below shows the decision function for a linearly separable problem, with three samples on the margin boundaries, called "support vectors":

Advantages:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

- Versatile: different Kernel functions can be specified for the decision function.

Disadvantages:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (in sklearn).

SVM	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	max_iter=1500, cache_size=2000	0.793601	0.089995	0.793832	0.089089	331.514857	2.654696	0.821429	0.830260
MinMaxScaler + Base		0.846169	0.013284	0.845901	0.013504	333.654148	6.134611	0.852462	0.933826
MinMaxScaler + SMOTE + Base		0.846906	0.016294	0.846967	0.016419	1220.819262	64.310314	0.834872	0.921614
MinMaxScaler + Base (Tuning)	C=0.001, kernel=rbf'	0.846906	0.008845	0.864685	0.008909	329.646469	43.775573	0.858140	0.942260
MinMaxScaler + (class weight + Base) Tuning	C=0.1, class_weight= 0: 3.6, 1: 1.0, kernel='linear'	0.889730	0.019498	0.887921	0.017702	6.506483	0.173380	0.8394214	

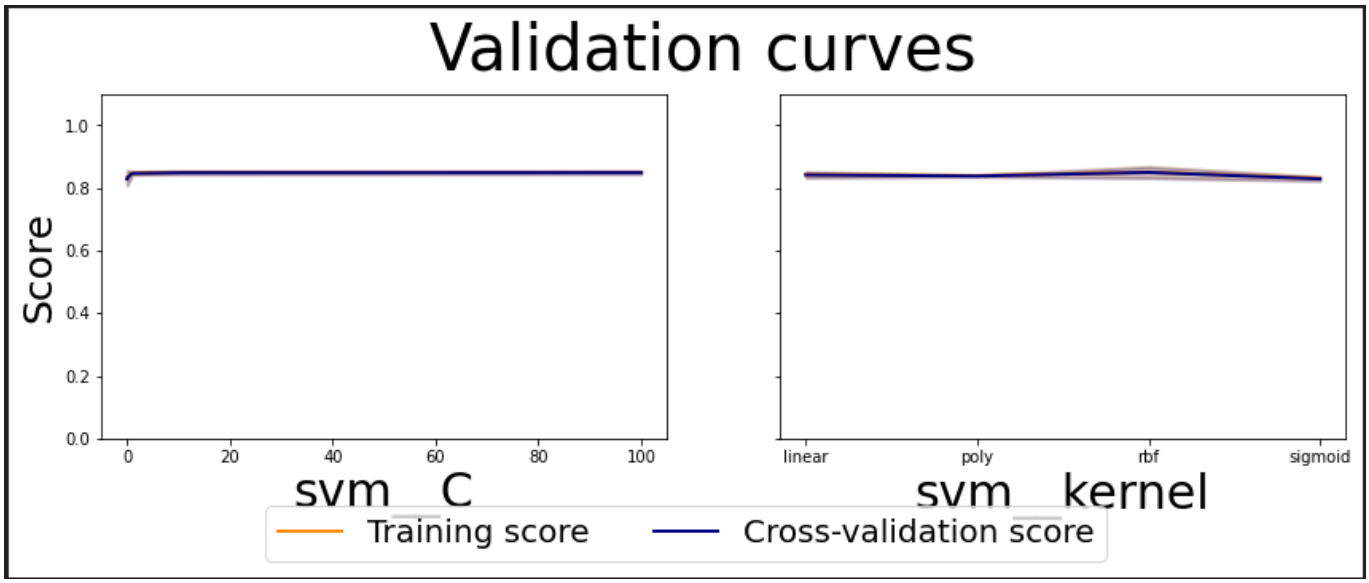

Figure 17: Validation curves

	precision	recall	f1-score	support
0	0.22	0.11	0.15	53638
1	0.80	0.91	0.85	215038
accuracy			0.75	268676
macro avg	0.51	0.51	0.50	268676
weighted avg	0.69	0.75	0.71	268676

```
[[  5806  47832]
 [ 20321 194717]]
f0.5_score= 0.8214285111631964
```

	precision	recall	f1-score	support
0	0.43	0.44	0.43	53638
1	0.86	0.85	0.86	215038
accuracy			0.77	268676
macro avg	0.64	0.65	0.65	268676
weighted avg	0.77	0.77	0.77	268676

```
[[ 23481  30157]
 [ 31268 183770]]
f0.5_score= 0.8581400257390641
```

Figure 18: Confusion matrices of best model (left) and base model (right) of SVM

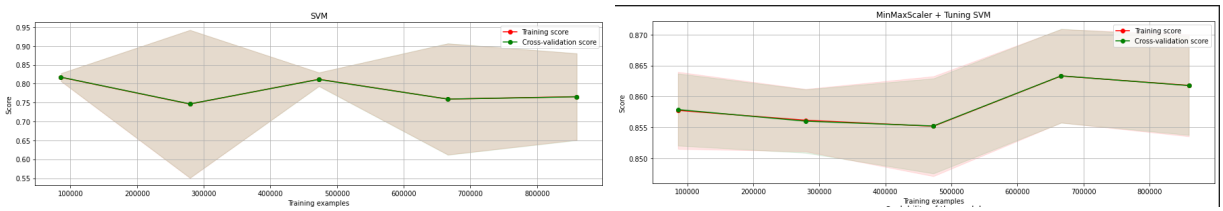

Figure 19: Score of best model (left) and base model (right) of SVM

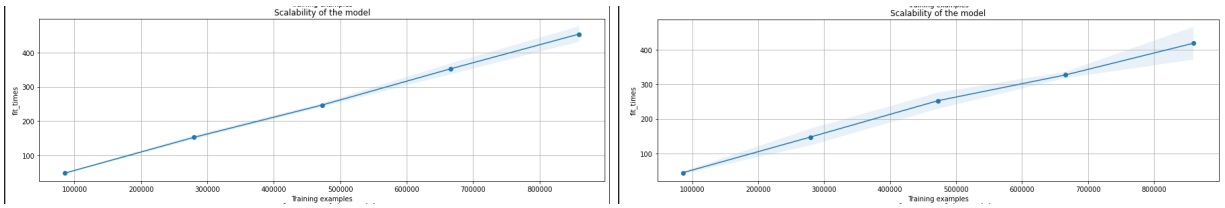

Figure 20: Scalability of best model (left) and base model (right) of SVM

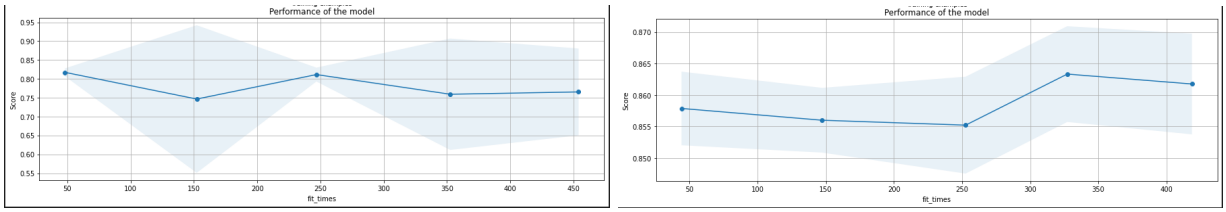

Figure 21: Performance of best model (left) and base model (right) of SVM

Decision Tree

Decision trees 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[10]. Furthermore, decision tree is the base algorithm for other advanced decision-making and ensemble learning techniques.

Advantages:

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

Disadvantages:

- 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	0.927076	0.000584	29.713296	1.736299	0.929246	0.957652
MinMaxScaler + Base		1.0	0.0	0.927115	0.000697	28.542332	1.72978	0.929127	0.957564
MinMaxScaler + SMOTE + Base		1.0	0.0	0.927115	0.000697	28.542332	1.72978	0.918826	0.951853
MinMaxScaler + Base (Tuning)	max depth=20, min samples leaf=20, criterion='gini'	0.939233	0.000545	0.929216	0.000783	22.88438	0.365039	0.930866	0.984321
MinMaxScaler + (SMOTE + Base) Tuning	max depth=5, min samples leaf=50, criterion='gini'. smote sampling strategy=1.0.	0.937240	0.000814	0.937158	0.000997	186.085667	10.462585	0.937954	0.980921
MinMaxScaler + (class weight + Base) Tuning	max depth=20, min samples leaf=100, criterion='gini' class_weight= 0: 0.5, 1: 0.5, kernel='linear'	0.949395	0.000290	0.944921	0.000595	31.826684	0.9456807	0.987454	

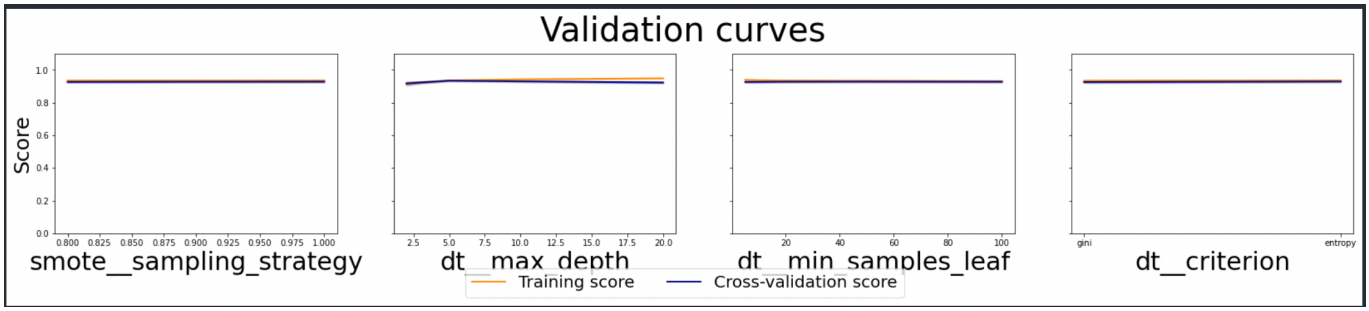

Figure 22: Validation curves

	precision	recall	f1-score	support
0	0.54	0.93	0.69	53638
1	0.98	0.81	0.88	215038
accuracy			0.83	268676
macro avg	0.76	0.87	0.78	268676
weighted avg	0.89	0.83	0.84	268676

```
[[ 49694   3944]
 [ 41591 173447]]
f0.5_score= 0.937954925470635
pr_auc_score= 0.9809206639922832
```

	precision	recall	f1-score	support
0	0.71	0.72	0.71	53638
1	0.93	0.93	0.93	215038
accuracy			0.88	268676
macro avg	0.82	0.82	0.82	268676
weighted avg	0.89	0.88	0.89	268676

```
[[ 38688  14950]
 [ 15982 199056]]
f0.5_score= 0.9292459260061509
pr_auc_score= 0.9576523418344546
```

Figure 23: Confusion matrices of best model (left) and base model (right) of Decision Tree

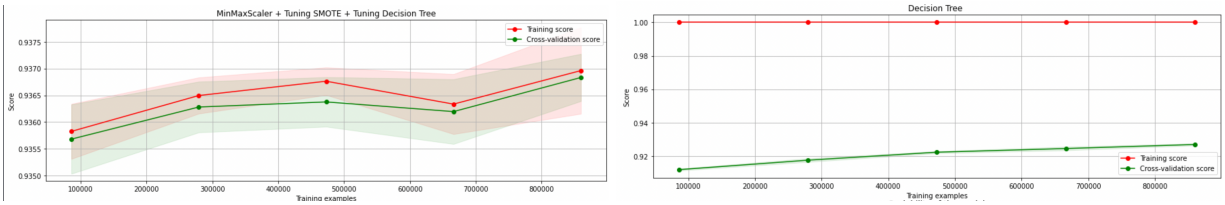

Figure 24: Score of best model (left) and base model (right) of Decision Tree

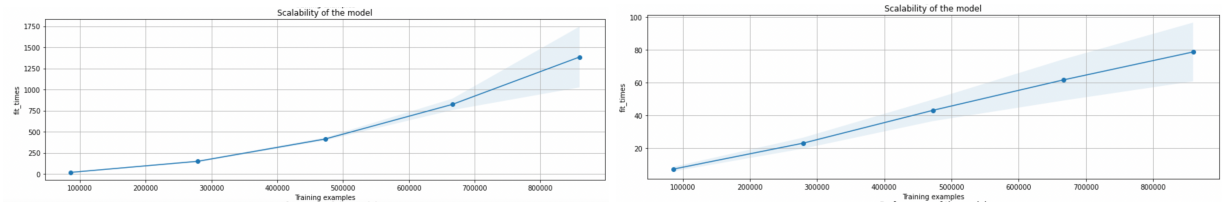

Figure 25: Scalability of best model (left) and base model (right) of Decision Tree

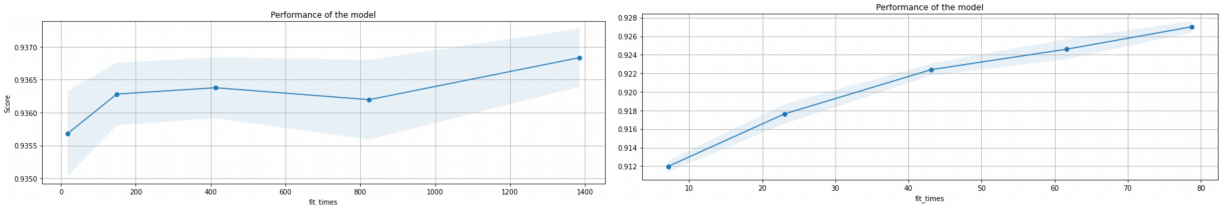

Figure 26: Performance of best model (left) and base model (right) of Decision Tree

4.5 Ensemble Models

Random forest

Decision Tree is a quick, simple, and understandable way to solve a classification problem. However, there are certain problems with this method[5]. The underlying technology under decision trees is ID3, a top-down greedy algorithm that picks the best feature at each stage to create a node. As a result, the algorithm can easily overfit if the tree-building process is not strictly controlled[6].

A classification system called Random Forest uses several decision trees. It attempts to produce a stochastic forest of trees whose prediction by the whole is more accurate than that of any individual tree by using bagging and feature randomness. Random Forest is a superb model since each created tree only uses some features and training points, therefore the output will be expected to have low bias and low variance.

Advantages:

- Large datasets can be handled efficiently.
- Versatile, can perform regression and classification tasks, also with numerical and categorical data.

Disadvantages:

- Training time is slow.
- Many of the hyperparameters in a random forest control trade offs between bias and variance. Therefore, in order to achieve an optimal model and avoid overfitting, tuning is necessary, which is also slow.

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:

Random Forest	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		0.99999	0.00001	0.920658	0.00051	376.3120	25.9211	0.921326	0.9864
MinMaxScaler + Base		0.99999	0.00001	0.920769	0.00046	439.7531	46.2193	0.921319	0.98641
MinMaxScaler + SMOTE + Base		0.99999	0.00001	0.933361	0.00051	1381.733	43.2558	0.934771	0.98560
MinMaxScaler + Base (Tuning)	max features='sqrt'; n_estimators=50	0.999966	0.000009	0.920785	0.000713	174.502827	4.308237	0.921831	0.98586

The oversampling approach SMOTE works fairly well with the underlying model as can be seen from the results above; it has certainly made the outcome brighter. In contrast, adjusting with hyperparameters cannot result in an observable change from the base model.

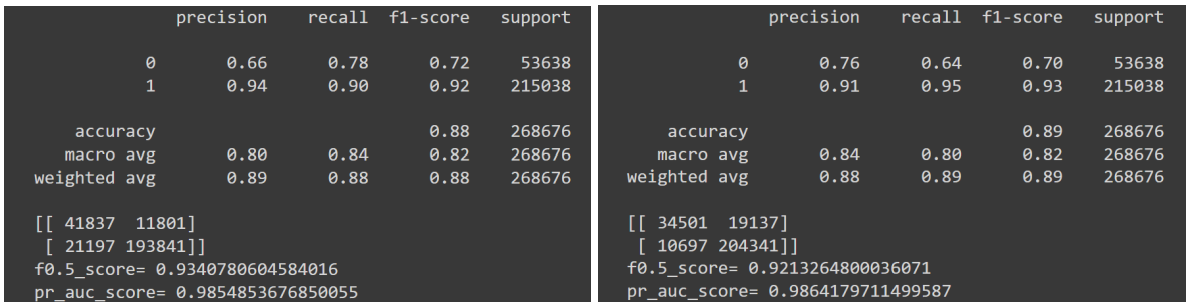

Figure 27: Confusion matrices of best model (left) and base model (right) of Random Forest

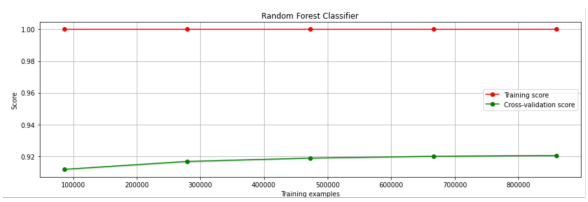

Figure 28: Score of base model of Random Forest

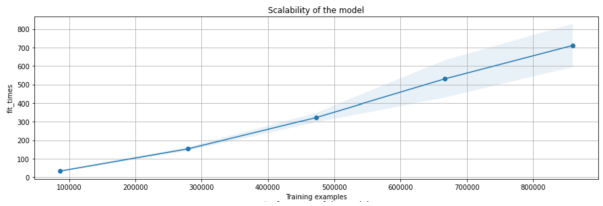

Figure 29: Scalability of base model of Random Forest

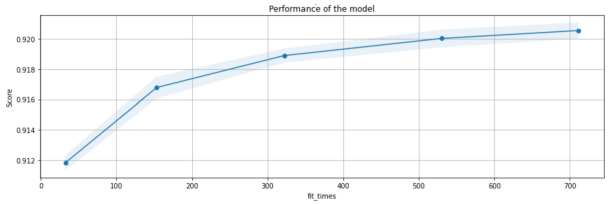

Figure 30: Performance of base model of Random Forest

XGBoost

Extreme Gradient Boosting, also known as XGBoost, is a powerful ensemble learning technique that uses two key technologies: Gradient Boosting and Decision Trees[12]. 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[8].

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.

The setup for pipeline is similar to the one in Random Forest. The result is then presented here:

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		0.920363	0.000241	0.920223	0.000504	178.637775	2.997294	0.920278	0.985985
MinMaxScaler + Base		0.920363	0.000241	0.920223	0.000504	171.318313	5.226494	0.920278	0.985985
MinMaxScaler + SMOTE + Base		0.94018	0.00022	0.940009	0.000499	1237.307	29.86738	0.940456	0.984511
MinMaxScaler + Base (Tuning)	max depth=6, n_estimators=500, reg_alpha=1.0, reg_lambda=1.0	0.958749	0.000345	0.9483265	0.000595	813.649615	4.073549	0.949691	0.993086

We can observe that the f0.5 score for the base model and the base model with MinMaxScaler are exactly the same. It can be stated that XGBoost does not require feature scaling because the boosting technique just verifies the presence of a sample in a node. The findings are greatly improved by the SMOTE technique that was incorporated into the procedure. However, tuning can even raise the score higher by using proper hyperparameters.

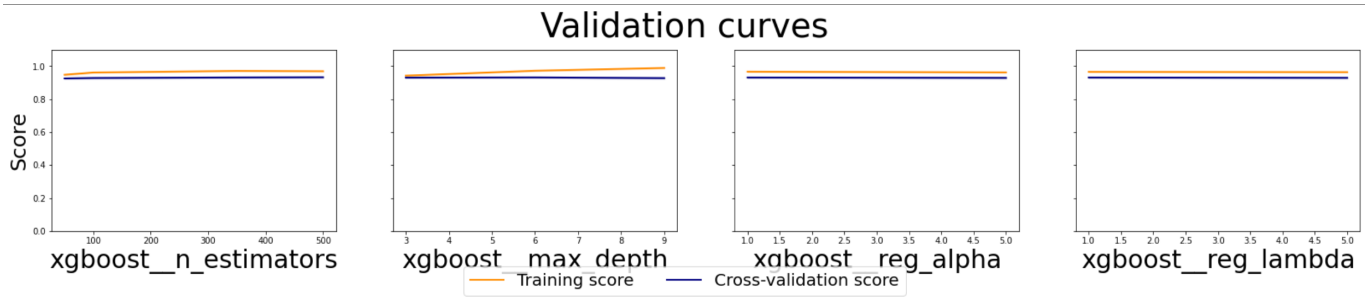

Figure 31: Validation curves

	precision	recall	f1-score	support
0	0.85	0.78	0.81	53638
1	0.95	0.97	0.96	215038
accuracy			0.93	268676
macro avg	0.90	0.87	0.88	268676
weighted avg	0.93	0.93	0.93	268676

```
[[ 41764  11874]
 [  7480 207558]]
f0.5_score= 0.9496909676911616
pr_auc_score= 0.9930858669023386
```

	precision	recall	f1-score	support
0	0.73	0.65	0.69	53638
1	0.92	0.94	0.93	215038
accuracy			0.88	268676
macro avg	0.82	0.80	0.81	268676
weighted avg	0.88	0.88	0.88	268676

```
[[ 34909  18729]
 [ 12717 202321]]
f0.5_score= 0.920278411044742
pr_auc_score= 0.9859853315541941
```

Figure 32: Confusion matrices of best model (left) and base model (right) of XGBoost

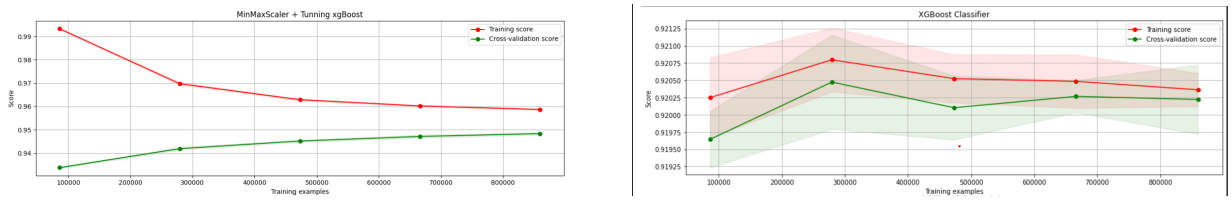

Figure 33: Score of best model (left) and base model (right) of XGBoost

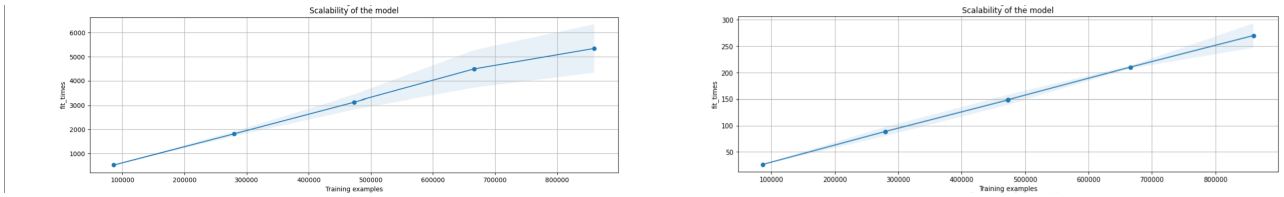

Figure 34: Scalability of best model (left) and base model (right) of XGBoost

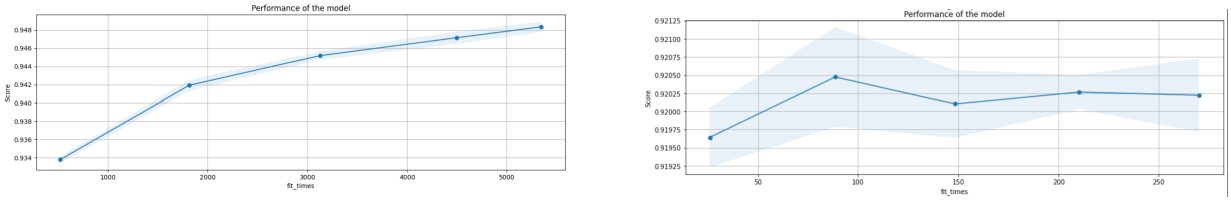

Figure 35: Performance of best model (left) and base model (right) of XGBoost

AdaBoost

AdaBoost is an ensemble learning method (also known as "meta-learning") which was initially created to increase the efficiency of binary classifiers. AdaBoost uses an iterative approach to learn from the mistakes of weak classifiers,

and turn them into strong ones. Rather than being a model in itself, AdaBoost can be applied on top of any classifier to learn from its shortcomings and propose a more accurate model. It is usually called the "best out-of-the-box classifier" for this reason[9].

Advantages:

- Less prone to overfitting as the input parameters are not jointly optimized[11]
- easier to use with less need for tweaking parameters

Disadvantages:

- Needing a quality dataset.
- More computationally expensive than the individual classifiers
- Sensitive to Noisy data and outliers

AdaBoost	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		0.917582	0.000472	0.917464	0.000908	119.126291	6.713171	0.917203	0.983849
MinMaxScaler + Base		0.917582	0.000472	0.917464	0.000908	115.364578	0.348798	0.917203	0.983849
MinMaxScaler + SMOTE + Base		0.932709	0.000752	0.932554	0.001039	943.144256	29.594005	0.9340511	0.981976
MinMaxScaler + Base (Tuning)	Working								
MinMaxScaler + (SMOTE + Base) Tuning	Working								

Like these ensemble algorithms above, adaBoost itself have a very good performance. When apply SMOTE, it even rise greater.

4.6 Deep Learning Models

Multi Layer Perceptron

The Multilayer Perceptron was developed to tackle the limitation of Perceptron, which has no ability to handle non-linear inputs and outputs. A Multilayer Perceptron has input and output layers, and one or more hidden layers with many neurons stacked together. Each layer is feeding the next one with the result of their computation, their internal representation of the data. This goes all the way through the hidden layers to the output layer. And while in the Perceptron the neuron must have an activation function that imposes a threshold, like ReLU or sigmoid, neurons in a Multilayer Perceptron can use any arbitrary activation function.

Advantages:

- MLP can be applied to complex non-linear problems
- Work well with large input data

Disadvantages:

- MLP with hidden layers have a non-convex loss function where there exists more than one local minimum. Therefore different random weight initializations can lead to different validation accuracy
- MLP requires tuning a number of hyperparameters such as the number of hidden neurons, layers, and iterations

- MLP includes too many parameters because it is fully connected. Therefore, it requires lots of computational power.

Below is the results after applying our tuning pipeline to the MLP. As the MLP consumes plenty of computational resouces, we have no ability to tuning the model MinMaxScaler + SMOTE + MLP, so that the last row is empty.

MLP	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		0.876214	0.005335	0.876115	0.005016	646.127894	194.734635	0.884666	0.973610
MinMaxScaler + Base		0.914479	0.002467	0.913958	0.002515	645.883598	44.239567	0.917118	0.984435
MinMaxScaler + SMOTE + Base		0.93672	0.000185	0.935511	0.000599	2029.161879	32.893923	0.935723	0.984072
MinMaxScaler + Base (Tuning)	hidden layer sizes = (50,50,50)	0.924010	0.002080	0.922807	0.002384	374.981939	2.476638	0.922871	0.987012
MinMaxScaler + (SMOTE + Base) Tuning	Working								

Compare the three model without tuning, it is clear to see that the model, which uses SMOTE, performs much better. As assumption, we are sure that the result of the model MinMaxScaler + SMOTE + MLP after being tuned will be enhanced a lot.

	precision	recall	f1-score	support
0	0.57	0.88	0.69	53638
1	0.97	0.83	0.89	215038
accuracy			0.84	268676
macro avg	0.77	0.86	0.79	268676
weighted avg	0.89	0.84	0.85	268676

```
[[ 47194   6444]
 [ 35789 179249]]
f0.5_score= 0.9357231601257033
pr_auc_score= 0.9840721628730936
```

	precision	recall	f1-score	support
0	0.58	0.49	0.53	53638
1	0.88	0.91	0.89	215038
accuracy			0.83	268676
macro avg	0.73	0.70	0.71	268676
weighted avg	0.82	0.83	0.82	268676

```
[[ 26342  27296]
 [ 18760 196278]]
f0.5_score= 0.8846659346959527
pr_auc_score= 0.973610296118056
```

Figure 36: Confusion matrices of best model (left) and base model (right) of MLP

Figure 37: Score of best model (left) and base model (right) of MLP

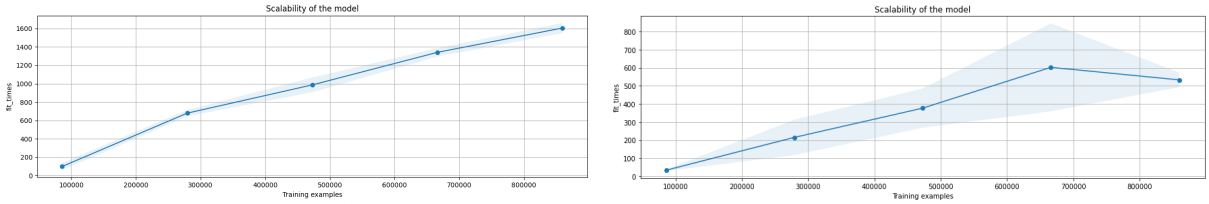

Figure 38: Scalability of best model (left) and base model (right) of MLP

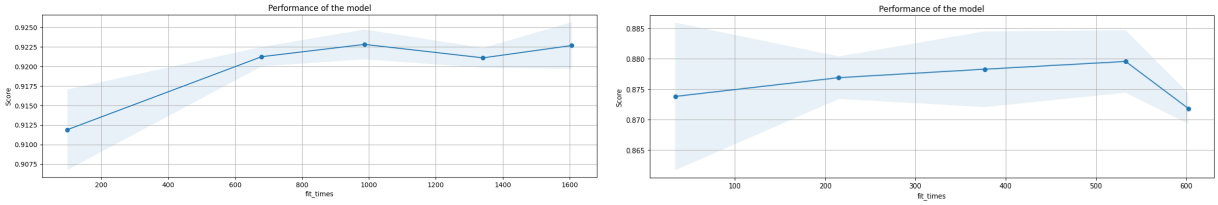

Figure 39: Performance of best model (left) and base model (right) of MLP

5 Conclusion and future work

5.1 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. Also, to make our models work much better, we try to tune the parameters for SMOTE as well as the other algorithms. And as you can see in the tables, the performance of the best model, the XGBoost, satisfies our demand really well.

5.2 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.

Using MapReduce models for SMOTE (e.g. SMOTE-BD and ApproxSMOTE) as well as classifiers to speed up running time.

6 List Of Tasks

Nguyen Ngoc Dung : EDA, outline code, Logistic Regression, SVM

Nguyen Huy Hai : Decision Tree, Adaboost

Duong Vu Tuan Minh: Xgboost, Random Forest

Nguyen Hai Long: Perceptron, Multi layer Perceptron

References

[1] Jason Brownlee. https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification. 17 March 2021.

[2] Json Brownlee. https://machinelearningmastery.com/precision-recall-and-f measure-for-imbalanced-classification. 3 January 2020.

[3] Json Brownlee. https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification. 6 January 2020.

[4] Json Brownlee. 'https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification/?fbclid=iwar3x$_{-f}w7sgm-yxweykz28ds1nrkrxb6pl2typpzlglgy1qmgj0kmaa_toy'.8January2020.

[5] DataCamp. https://blog.paperspace.com/adaboost-optimizer.

[6] Niklas Donges. https://blog.paperspace.com/adaboost-optimizer. 6 July 2022.

[7] Nathan George. https://www.kaggle.com/datasets/wordsforthewise/lending-club. 23 July 2021.

[8] H2O.ai. https://xgboost.readthedocs.io/en/stable/. 2016-2022.

[9] Vihar Kurama. https://blog.paperspace.com/adaboost-optimizer. 2020.

[10] Mukesh Mithrakumar. https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680. 12 November 2019.

[11] Avinash Navlani. https://www.datacamp.com/tutorial/adaboost-classifier-python. 20 November 2018.

[12] xgboost developers. https://xgboost.readthedocs.io/en/stable/. 2021.