Machine Learning Engineer Nanodegree

Capstone Project

Hyungoo shim

August 22, 2018

Home Credit Default Risk

1. Definition

Project Overview

This project is be based the Kaggle competition Home Credit Default Risk[1]. Problem is predicting whether or not a client will repay a loan or have difficulty is a critical business need and also many people struggled to get loans due to bad credit history. It can aggravate your financial life. There are lots people are in a dangerous financial situation with who is earning a minimum wage. Every time someone borrows money from the bank but If you have a non-credit history or bad credit report. Thus, If you don't have any credit record, unfortunately, you will suffer from financial problems.

Home Credit is a non-financial institution, founded in 1997 in the Czech Republic[2]. The company operates in 14 countries and focuses on lending primarily to people with no credit record and will become victims of untrustworthy lenders.
Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. However, One thing's for certain, when you don't have a credit report, you don't have any dept. In the other word, If you don't have any debt, then you do not have any credit history.
In order to make sure this underserved population, Home Credit company uses a variety of data like telco and transactional information to predict clients' repayment abilities.

[1] https://www.kaggle.com/c/home-credit-default-risk

[2] http://www.homecredit.net

Now this problem's solution is very important in our society. At least, It can help increase the client's potential stable financial situation and help achieve their dream of starting a family or sharing a home.

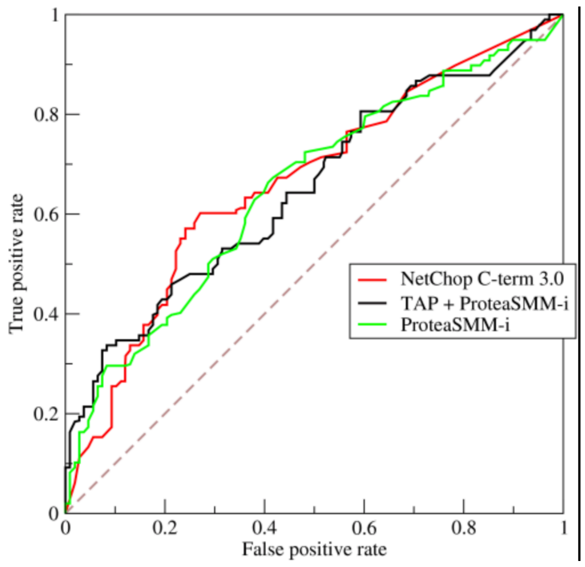
Problem Statement

The purpose of Home Credit Default Risk is to predict using historical loan application data. We have to know outcome that whether or not the customer will be able to repay a loan. It's definitely supervised and categorical problem like Email labeled as spam / not spam.
The labeled datas are included in the training data and our model is to learn to predict the label and This label is a zero or 1. Zero is will repay the loan on time and 1 is will have difficulty repaying the money.

Metrics

In the description of the competition’s dataset provided, the metric used to evaluate the performance of a given classifier will be the area under the ROC curve. When we measure the result, we do not predict 0 or 1, but rather a probability between 0 and 1. A ROC curve is one that plots the False positive rate(FPR) against the True positive rate (TPR) for a given dataset using a given classifier.

A larger area results in a more successful classifier and the model with a high ROC AUC will also have a high accuracy with a graph plotting the ROC curves.



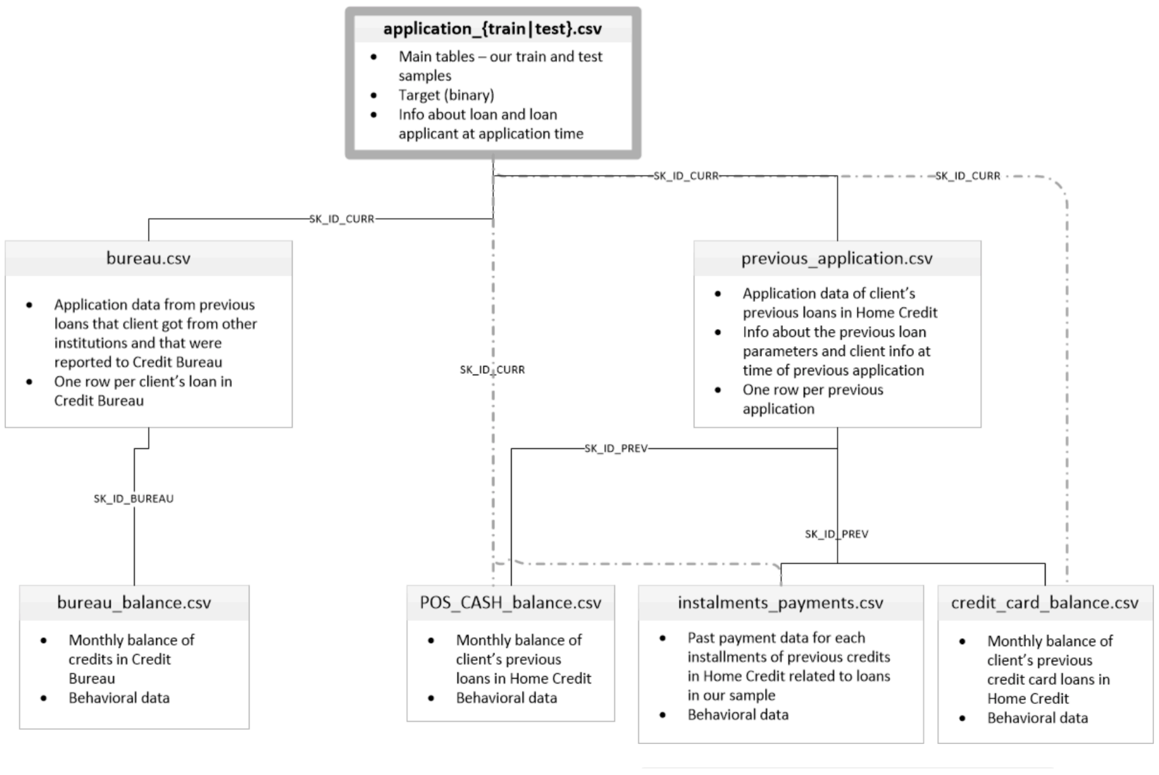
https://en.wikipedia.org/wiki/Receiver_operating_characteristic

A single line on the graph indicates the curve for a single model, and movement along a line indicate changing the threshold used for classifying a Positive instance. It's threshold will be at 0 in the upper right to and goes to 1 in the lower left. Cleary, a curve that is to the y-axis and above another curve indicates a better model. So, we can select the best algorithm with ROC curve graph and model's higher score with between 0 and 1.

2. Analysis

Data Exploration

The datasets by Home Credit are 7 different sources of data. These Data frames contain 24 attributes, 30000 instances, and more 220 columns across seven files.


application_{train|test}.csv
Main tables – our train and test samples
Target (binary)
Info about loan and loan applicant at application time
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
bureau.csv
Application data from previous loans that client got from other institutions and that were reported to Credit Bureau
One row per client's loan in Credit Bureau
previous_application.csv
Application data of client's previous loans in Home Credit
Info about the previous loan parameters and client info at time of previous application
One row per previous application
SK_ID_PREV
SK_ID_BUREAU
SK_ID_PREV
bureau_balance.csv
Monthly balance of credits in Credit Bureau
Behavioral data
POS_CASH_balance.csv
Monthly balance of client's previous loans in Home Credit
Behavioral data
instalments_payments.csv
Past payment data for each installments of previous credits in Home Credit related to loans in our sample
Behavioral data
credit_card_balance.csv
Monthly balance of client's previous credit card loans in Home Credit
Behavioral data

- **application_train / test** : This dataset is main table. It contains the loan and loan application. Every loan has row and this row is identified by **SK_ID_CURR** what is connecting with the *bureau* data, *previous_application.*

- **Bureau / bureau_balance :** Application data from previous loans that client got from other institution and contains monthly balance of credits in Credit Bureau

- **previous_application :** Application data of client's previous loans in Home Credit

- **POS_CASH_balance :** Monthly balance of client's previous loans in Home Credit

- **Installments_payments :** Post payment data for each installments of previous credits in Home Credit related to loans
- **credit_card_balance :** Monthly balance of client's previous credit card loans in Home Credit

DATA SOURCE	The number of Observations	The number of Features
application_train data (with Label)	307,511	122
application_test data (No Label)	48,744	121
bureau data	1,716,428	17
bureau_balance data	27,299,925	3
POS_CASH_balance data	10,001,358	8
previous_application data	1,670,214	37
installments_payments data	1,736,093	8
credit_card_balance	3,840,312	23

With application_train datasets, The dependent variable is the label we want to predict. 0 for the loan was repaid on time and 1 indicating the client had payment difficulties. However, 1 is about 8% and 0 is approximately 90% from this information and we know that it is a highly unbalanced data pattern.

Thus, there is a lot of extra data provided, like credit card histories, previous loan data applications of all agencies and monthly snapshots of cash loans and payments.

Exploratory Visualization

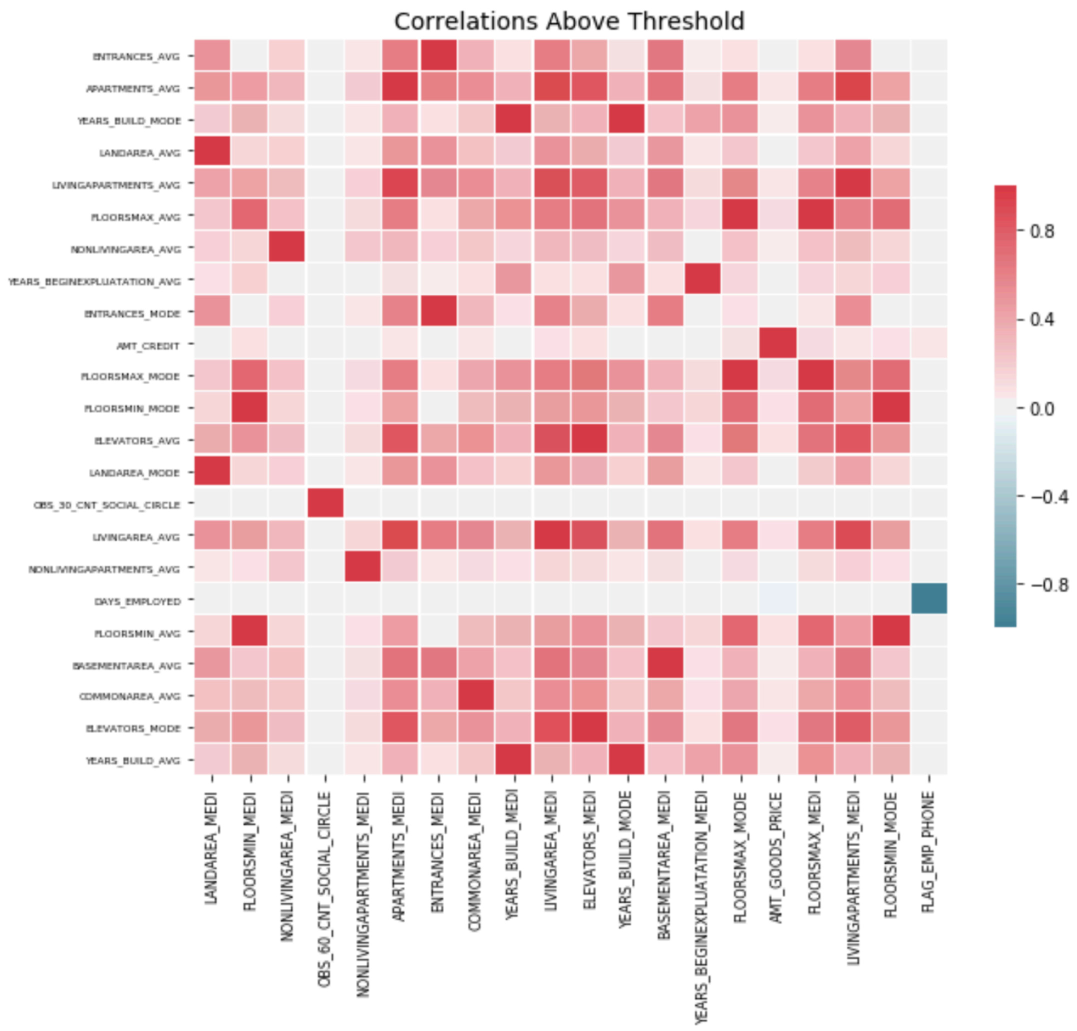
To visualize data's EDA, I choose the application_train dataset. this dataset has a main table and crucial information of clients. It covers the client's gender, age, information on income, family status, etc. So, I can predict whether a client will repay a loan or have difficulty with only this dataset roughly.

Firstly, FeatureSelector Class can be used to determine whether or not there are any interesting. this Class is open-source by Will Koehrsen[3] for selecting features to remove from data and has five advantages for explore data and remove features. Below is these method by will koehrsen gitbub.

1. Find columns with a missing fraction greater than a specified threshold
2. Find features with only a single unique value

[3] https://github.com/WillKoehrsen

3. Find collinear features as identified by a correlation coefficient greater than a specified value
4. Find feature with 0.0 importance from a gradient boosting algorithm
5. Find feature that do not contribute to a specified cumulative feature



Above the heatmap indicate the correlation greater than 0.97 between features. It is based on the Pearson's correlation coefficient[4] .

As you can see, FLAG_EMP_PHONE feature has weaker correlation with other features and also OBS_30_CNT_SOCIAL_CIRCLE is recorded weakness. These features on the X - axis will be removed in the application_train/test set. because FeatureSelector works by finding feature importance using a gradient boosting machine algorithm in the LightGBM[5] library.

[4] https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

[5] https://lightgbm.readthedocs.io/en/latest/

Algorithms and Techniques

This problem of Home Credit Default Risk is one of supervised learning and one of the binary classification. there are great algorithms available for training a classifier to learn such as SVM, Decision Tree, Random forest, and Logistic regressor. Logistic regressor is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 or 0. I choose the this algorithm to the baseline model for these datasets. because it is a widely used technique and very efficient. besides, it does not require too many computational resources.

After preprocessing all dataset, I tested this classifier and submitted the result. Logistic regressor's kaggle submission score was 62.3%. Of course, it is not powerful enough to classify the client into 'will repay' or 'have difficulty' and I can be satisfied with this classifier's score. I needed other more powerful algorithms like ensemble methods.

1. GBM

To convert weak learners to strong learner, we have to combine the prediction of each weak learner using methods like average/weighted average and considering prediction has the higher vote. There are many boosting ensemble algorithm types like Adaboost, GBM, and XGBoost. boosting algorithm plays a vital role to improve the accuracy of these models with bias & variance trade-off. Unlike Bagging algorithm, boosting controls both bias and variance and it will be a more efficient way.

Boosting not only combine a set of the weak learner but also delivers improved prediction accuracy. If there are any instances, boosting model outcomes are weighed based on the outcomes of the previous instance -1 and its outcomes predicted correctly are given a lower weight. In contrast, miss-classified predictions are weighted higher like penalty.

- Class : sklearn's ensemble model - sklearn.ensemble.GradientBoostingClassifier[6]
- Defualt parameters selected for modeling

    a) ***learning_rate*** : Controls the contribution of weak learners in the final combination. Lower values make the model robust to the specific tree and allowing it to generalize well however it requires the higher number of trees thus will be computationally expensive while model learning.

[6] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

b) ***n_estimators*** : Controls the number of weak learners. Higher values are robust but it can overfit at a point, therefore, this value should be tuned using cross-validation.
(There is a trade-ff between n_estimators and learning_rate)

c) ***max_depth*** : The maximum depth of a tree it used to control over-fitting as higher depth. it can be tuned using CV.

d) ***max_leaf_nodes*** : The maximum number of terminal nodes it can be defined in place of max_depth.

2. *XGBoost*

The XGBoost is favorite in machine learning competitions and sometimes we called GBM killer. It has become great tool for winning competitions. this method is a highly sophisticated. Also it is very powerful to deal with the anomaly dataset and can be used parallel processing. furthermore, supports implementation on Hadoop. But it requires lots of data and take a long time to train. it is a really stressful one.

GBM divides the optimization problem into two parts by first determining a direction of step then optimizing the step length. Contrastively, XGBoost method determine the step directly by solving. Trees in the XGBoost model have a number of terminal nodes and leave weights of the trees that are calculated with less information. The extra randomization parameter can be used to reduce the correlation between the trees. finally, the lesser the correlation among classifiers, the better our ensemble of classifiers will return. One of the XGBoost's advantage is allowing the user to run a cross-validation at each iteration of the boosting process and it is easy to get the exact optimum number of boosting iterations and users to define custom optimization objectives or evaluation criteria.

- Class : import xgboost as xgb[7] in python
- Default parameters selected for modeling

a) ***booster*** : Select the type of model to run at each iteration. gbtree is the tree-based models, gblinear is the linear models.

b) ***max_depth*** : The maximum depth of a tree, same as GBM. This should be between 3-10.

c) ***gamma*** : Default is 0. this is minimum loss reduction required to make a further partition on a leaf node of the tree and it can vary depending on the loss function and it should be tuned.

[7] https://xgboost.readthedocs.io/en/latest/

3. *LightGBM*

LightGBM algorithm is really fast and has high-performance. It used for ranking, classification, and many other machine learning problems. this method is based on Tree based learning algorithms like XGBoost, it splits the tree leaf-wise with the best fit whereas other boosting algorithms split the tree depth or level-wise rather than leaf. Also, this algorithm require lower memory usage because it replaces continuous values to discrete values which result in lower memory usage. below is LightGBM's advantage from LightGBM page: [8]

I. Faster training speed and higher efficiency
II. Lower memory usage
III. Better accuracy
IV. Parallel and GPU learning supported
V. Capable of handling large-scale data

Many gradient algorithm frameworks like GBDT, GBRT, and GBM approach of splitting vertically on the features to distribute on worker nodes, is quietly computation overhead and communication of splits among the nodes. If we have small dataset then lightGBM distributes complete data to all the nodes and the communication cost is reduced as all the nodes are aware of the split. If we have the larger dataset, this method merge a different histogram built on a different set of data available on worker nodes.

- Class : import lightgbm as lib in python
- Default parameters selected for modeling

a) **application:** This is the most important parameter and specifies the application of model, whether it is a regression problem or classification problem. LightGBM will by default consider model as a regression model.

b) ***max_depth*** : It describes the maximum depth of tree. This is used to deal with overfitting.

c) ***min_data_in_leaf*** : Min number of data in one leaf.

d) ***feature_fraction*** : Default is 1. 0.8 feature fraction means LightGBM will select 80% of parameters randomly in each iteration for building trees.

e) ***num_iterations*** : Number of boosting iterations to be performed. default is 100 and Int type.

[8] https://lightgbm.readthedocs.io/en/latest/index.html

Benchmark

The first submission with the logistic regressor was score was 0.623 with default parameters and the training dataset has 307511 of observation and 121 of features. this score is calculated with approximately 20% of the test data.

Evaluation is on area under the ROC curve between the predicted probability and the observed target. At present, the top ranking score is 0.812. Deadline remains 10 days. So, with ROC auc score of 0.8 ~ 0.9 will be considered an excellent level while a score of 0.75 ~ 0.8 is a good result.

3. Methodology

Data Preprocessing

These dataset have lots of missing data and categorical value. To succesful data management, data cleaning and feature selection are a necessary process. But it can be a tedious process. So, we need tools like sklearn's libraries and FeatureSelector class or helpful open-source libraries.

- To handle categorical variable, I used the one-hot encoding library such as get-dummies function from sklearn and pandas.

- To handle low importance features, (correlation threshold 90%) I dropped 22 features in the application_train / test set using FeatureSelector class.

- For Missing value imputation, Usually median is used with numeric columns and mode with non-numeric columns. I decided to fill null values using the median from sklearn's imputer class. this class fills missing values with some statistics (e.g. mean, median, ...) of the data.

- To grouping all dataset, I group a data by a column. In this case, I grouped by the SK_ID_CURR column mostly.

- To calculate data's statistics information like mean, max, min, and sum between columns, I call the aggregate function of pandas.

- As you know, this project has 7 different datasets so after preprocessing all data, we need to merge the main application train/test data with others. Thanks to the pandas, I used pd.merge function. it was not hard and it was pretty fast.

Implementation

Once datasets preprocessed for three models, I defined a function which will create classifiers and trained each model with the train set which has 3,017,511 of row and 1,912 columns and initial parameter was default parameters. Because without any tuning, I can choose the best score model. Prediction performed using the prepared from testing data. Making predictions involves metric used to score the classifiers is an area under the ROC curve. To implement this, ROC auc socre's function defined in the process. I've already recorded these performances:

	GBM	XGBoost	LightGBM

Default Parameters	learning_rate=0.1
	n_estimators=100
	max_depth=3.
	min_samples_split=2.
	min_samples_leaf=1
	subsample=1.0.
		learning_rate=0.1,
		max_depth=3,
		min_child_weight=1,
		n_estimators=100,
		subsample=1
			learning_rate=0.1,
			max_depth=-1,
			min_child_samples=20
			min_child_weight=0.01,
			n_estimators=100,

Training AUC score	0.783	0.781	0.825
Submission (Testing) AUC score	0.771	0.769	0.781
Training Time	3425.0 sec	2195.5 sec	106.5 sec
Prediction Time	1.5 sec	4.3 sec	1.7 sec

For evaluating models with default parameters, the performance of models in terms of both speed and accuracy is LightGBM comes out as the winner maximum accuracy on train set and test set. Score were 0.825 and 0.781. but at the overfitting, both train and test accuracy have gap than other models.
Training time & predict time was minimum. as you know, LightGBM's one of the advantages is very very fast. Since it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than others. This algorithm has shown better results and has outperformed other boosting algorithms as well as it is very time-saving than XGBoost and GBM.
Next performer was GBM which generally works well. Its accuracy was quite close to LightGBM. Submission score was 0.771 and training accuracy is 0.783. In this case, GBM records the minimum overfitting. both train and test accuracy are close. However, one of

a problem with GBM is that its algorithm did not fast. Training time with data set was 3425 second. This time took nearly 30 times than Light GBM's training time.

The last place goes to XGBoost. it achieved similar accuracy with much faster speed compared to GBM. The accuracy was 0.781 of training and 0.769 of the testing set. but we know that XGBoost takes a long time to train data. It is too slow especially If we tune its parameters with GridSearchCV, It will take a very long time. The better way is to tune parameters separately rather than using GridSearchCV.

Lastly, I decided to choose the LightGBM model with the submission score and training time. This algorithm has shown far better results and has outperformed existing boosting algorithms. I'll strongly recommend you to implement Light GBM over the other boosting algorithms.

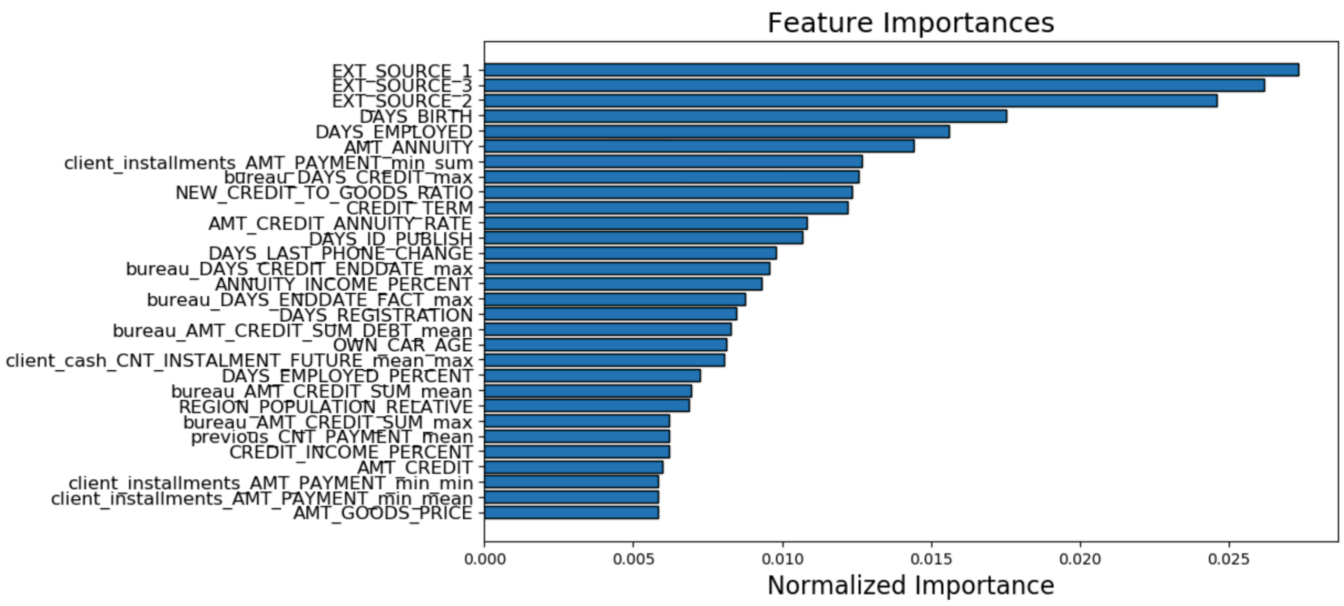
Refinement

The LightGBM classifier was selected for further refinement. This refinement step will be to dig down into parameters in order to fine-tune parameters to produce the best score model. Below the table is additional parameters for searching. It will take a longer amount of time.
There are two types of the parameter to be tuned. one type is tree based and another is boosting parameters. There are no optimum values for learning rate as low values always work better, given that we train on a sufficient number of trees but a high number for learning_rate can lead to overfitting in another manner I will reduce the learning rate and increase trees.

PARAMETERS	VALUES
learning_rate	0.003, 0.005, 0,007, 0.1
max_depth	-1, 3, 5, 7
min_child_weight	2, 5, 7
sub_sample	0.5, 0.7, 0.9
colsample_bytree	0.5, 0.7, 0.9
max_bin	255, 280
num_iteration	100, 200, 300

Once parameters were passed to the GridSearchCV function, this function returns from its search. The classifier will then fit with training data using fit( ) on the classifier. And

LightGBM is very similar to XGBoost's tuning, I used a gradient boosted tree approach.[9] Below is the most important 30 features selected using FeatureSelector for Feature Selection process. It will be used later after removing zero importance.



4. Results

Model Evaluation and Validation

LightGBM	Training Time (s)	Testing Time (s)	Train ROC score	Submission ROC score
Default parameters	1142.2	2.5	0.825	0.781
Tuned parameters	120.4	3.25	0.825	0.783
Feature Selection	50.5	1.1	0.78	0.747

This table is the final LightGBM score for models. There is a maximum score of 0.783 and this score was improved by 0.002 than default parameters. For that reason, I reduced the feature dimension using FeatureSelector. however, it was unexpected that training on the dataset (307511 of observation and 614 of features) without low importance would result in a much lower scores both train set and submission. It performed poorly in terms of both feature selection and parameter tune. Important parameters of LightGBM for better accuracy are max_bin, max_depth, num_leaves and small learning_rate with large

[9] https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

num_iterations. But a small improvement which probably shows that the parameter tuning do not help in this model.

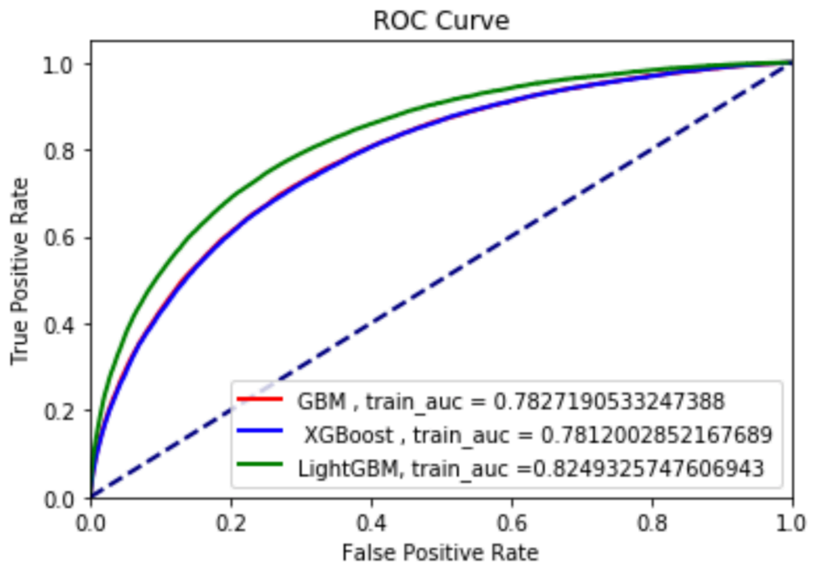
Justification

All of the hard work translates to a tiny improvement of 0.002 ROC auc over the original testing data. Removing the zero importance features decreases performance. Nonetheless, this algorithm has shown far better results and has outperformed existing boosting algorithms. Especially, training and testing time is constantly decreased. this algorithm has faster training speed and higher efficiency. XGBoost algorithm is the ultimate weapon of many ML engineer. It's a highly sophisticated algorithm, powerful enough to deal with all sorts of irregularities of data and has higher accuracy. However, the only problem is too slow. It was really frustrating to tune its parameters.

About parameter tuning, when creating the model we don't know what the optimal model parameter should be for a given data. I used the GridsearchCV for tuning. it is the most basic parameter tuning method. I defined a list of values to try for both min_child_weight and max_depth and a grid search would build a model for each possible combination. With this technique, I simply build a model for each possible combination of core parameters but it was tedious to evaluating and selecting the optimal value which produces the best result.

5. Conclusion

Free-Form Visualization



This graph is ROC curve to compare differential three models used to develop a solution to the problem. Having a visual representation of the score with train data helps to determine which model is best to improve the result.

Reflection

This capstone project began with a statistic analysis of the dataset and determined data cleaning processes like finding missing value and outlier data. It was decided to remove and impute missing value before training and testing. And Handle categorical columns so that it can be encoded as one-hot encoding using pandas class. Once the preprocessing was done, I implemented feature selection with the main application train/test and 6 other data. After feature engineering, the main training and testing dataset were merged with other data files.

Next, GBM, XGBoost, and Lightgbm were trained with merged dataset the results of these classifiers on the test data set were records and compared using ROC auc score with training time and testing time. From these scores, it was determined the best accuracy model to use for this problem.

Finally, the LightGBM classifier model was refined by adding additional search parameters using Gridsearch and K-fold cross-validation and re-tunning. the hardest work was to determine which parameters to add to the search space for the grid search, as it is simply not feasible to add all of the available parameters. it was necessary to come up with a set of possible value of the parameters. Unexpectedly, sometimes the re-tuning process drop the ROC auc score and core parameters for the better accuracy were close to LightGBM's default values mostly. So it was difficult to select a good range.

While I failed a lot, the LightGBM model accuracy was not bad. I think it was pretty good but not excellent. However, by attempting to understand how our models make decisions, I can try to improve them or examine the mistakes in order to correct the errors.

Improvement

A small improvement of LightGBM model over the dataset, I consider a different method for feature engineering. I can say that some of the features I removed are the most important. So, Attempt different strategies for outlier detection and removal will improve results. I guess, the better performance of Machine learning model is won by feature engineering. People who create the most useful features out of the data will winner in competitions or real problems.
There are many techniques we can use to both create features and select features like using Domain knowledge, Polynomial engineering, and Automated Feature Engineering.

or Study the further underlying relationship between features than others. Reasonably, Choosing the right model and optimal settings with feature selection are important.
The other way to improve the result is using Neural Network. Neural Nets are powerful algorithms. This method is complex and it requires lots of a good idea for this problem. Thought, since neural nets are more powerful, It may help to solve this real problem.