KAGGLE COMPETITION

IEEE-CIS FRAUD DETECTION

OCTOBER 15, 2019

AUTHOR： YINGFEI HONG, JIALE LU, JIAHAO LI, MUXI CHEN
INSTRUCTOR： SHUO ZHANG, PH.D., COLUMBIA UNIVERSITY
EMAIL: shuozhang1985@gmail.com

1. Introduction

Upon commencement, we had some experience of R, Python, and the basics of machine learning. During the process, we gained knowledge of various machine learning algorithms, from supervised or unsupervised to reinforcement learning. Now, the time has come to apply them to solving real-world problems and putting what we have learned to the test. Fortunately, we subsequently discovered an ongoing and fascinating competition on kaggle.com, a practical competition whose goal is to detect fraud occurring within financial transactions. The data is derived from Vesta's real-world e-commerce transactions and contains a wide range of features, from device type to product features. Of these features, the training set contains about 590,000, while the test set contains about 500,000. The data is mainly divided into two categories, transaction and identity data.

We considered this type of competition approachable for comparative novices like us due to the detailed explanations of the features provided in full to each participant. This meant that we could more fully seize this invaluable and meaningful opportunity to both practice and hone our skills and increase our practical knowledge of advanced machine learning techniques, such as creative feature engineering and advanced regression techniques like random forest and gradient boosting.

From Aug 16th to Oct 1st, we dedicated ourselves to performing EDA, feature engineering, and ensembling, consistently fine-tuned our parameters consistently, and adjusted our hyper-parameters in order to improve our overall accuracy. With the intention of better stimulating the scoring standard of the competition, our codes are score-orientated. Our best submission to the Kaggle private leaderboard ultimately ranked in the top 19% out of more than 6,300 teams.

2. Workflow

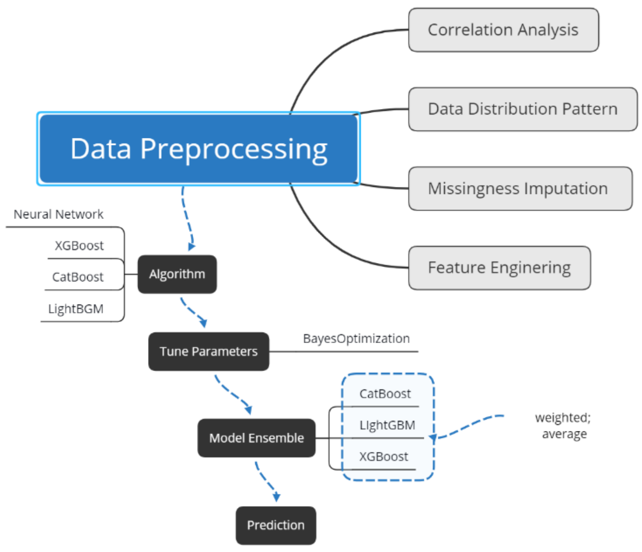


Figure 1. Workflow of the Program

3. Pre-Processing

3.1 Data Distribution Pattern

We aim to predict the probability that an online transaction is fraudulent, thus it is important to determine the distribution of fraud labels.

As shown in the first map below, the shaded blue rectangle represents the non-fraudulent transactions (96.501%), and the shaded green rectangle represents fraudulent transactions (3.499%). It is obvious that the data is highly imbalanced and most of the transactions are non-fraud. If we use this data frame as the base for our predictive models, we might get a lot of errors and our algorithms will probably overfit.

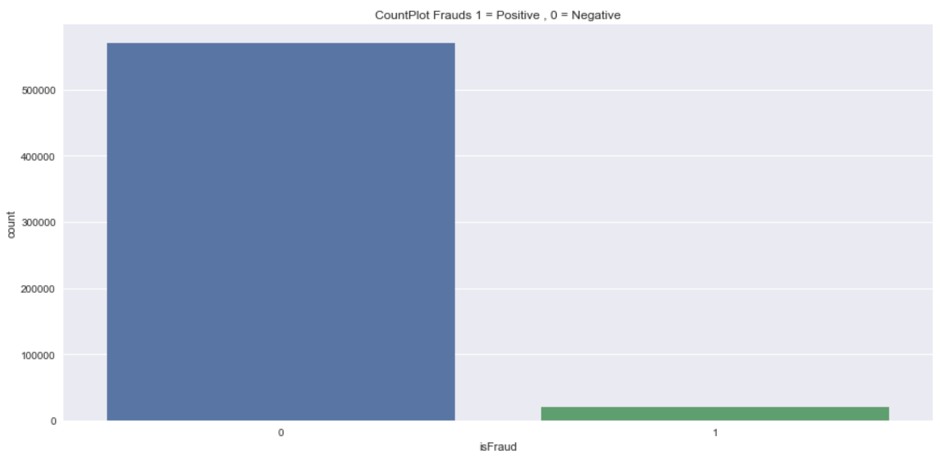
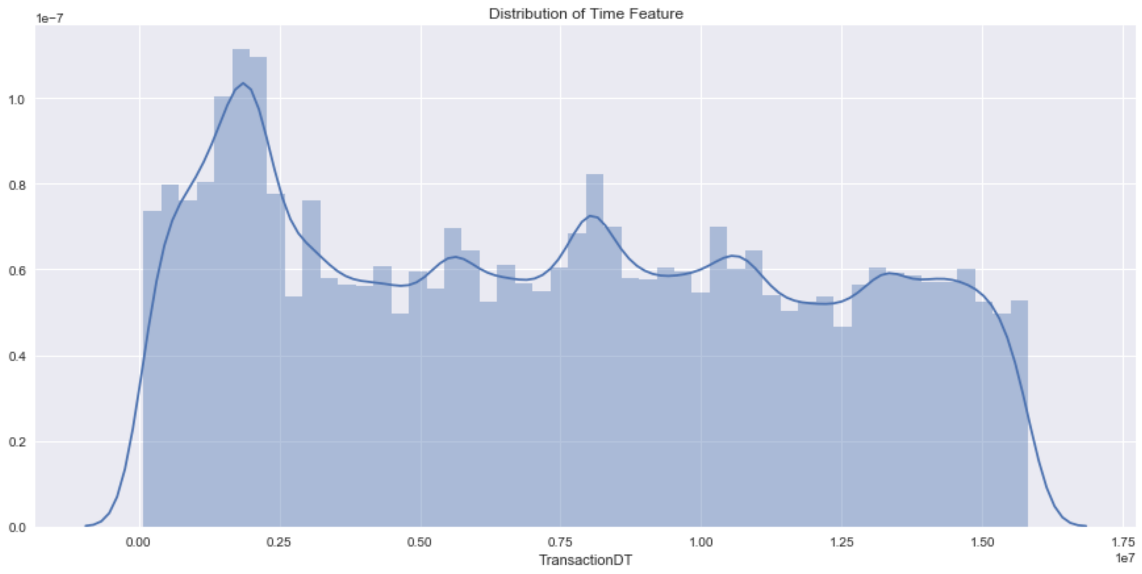


Figure 2. Distribution of Fraudulent Transactions



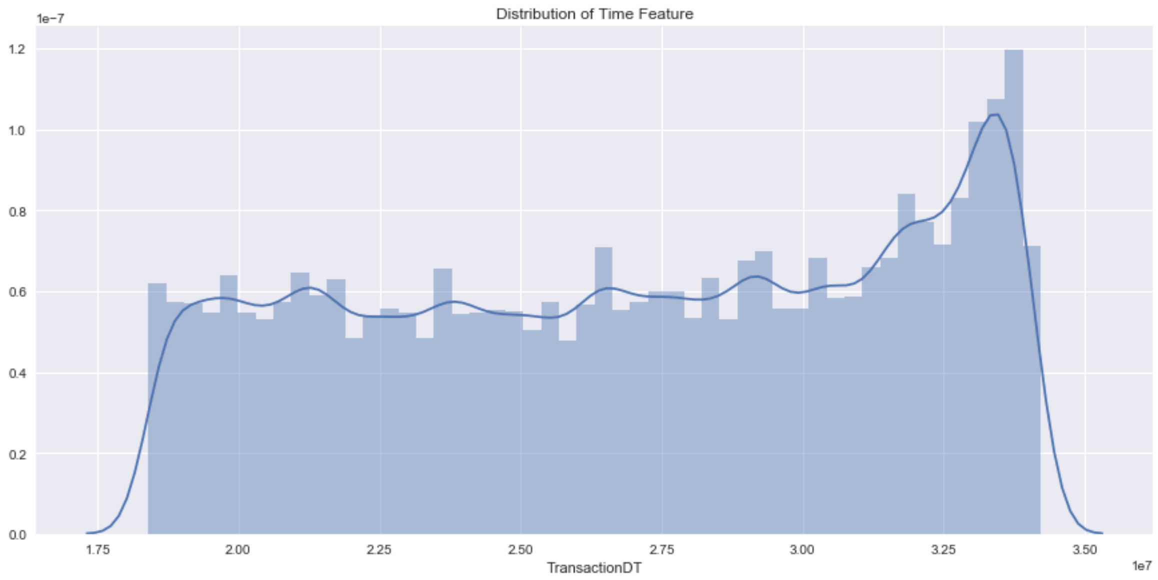
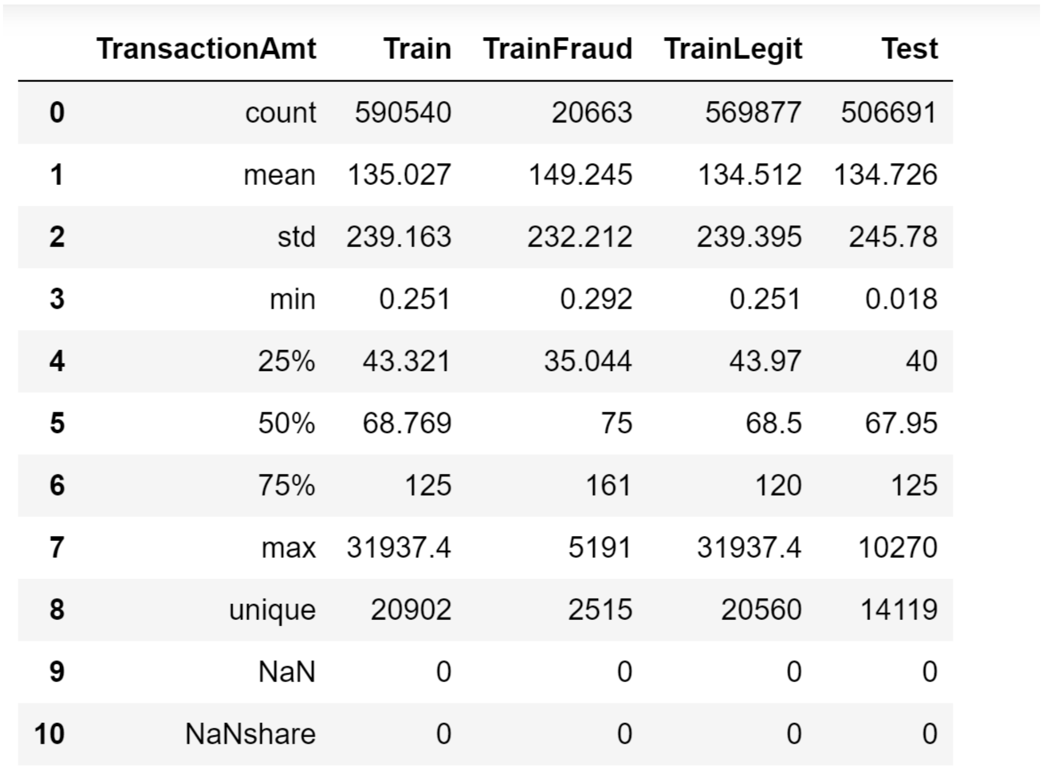


Figure 3. Distribution Plots of Transaction DT

One of the most important features within this dataset is TransactionDT. This is a time-related feature and, as we can see below, the time within the test set is ahead of that used within the train set, and the train transaction dates and the test transaction dates do not overlap. Given this information, it would be prudent to employ a time-based split for the benefit of validation.

Another important feature within this dataset is TransactionAMT. From the table below, we can observe that the least amount is 0.25, while the largest is 31927.4. It is worth noting, however, that most of the transactions are below 125. This feature displays a skewed distribution.



	TransactionAmt	Train	TrainFraud	TrainLegit	Test
0	count	590540	20663	569877	506691
1	mean	135.027	149.245	134.512	134.726
2	std	239.163	232.212	239.395	245.78
3	min	0.251	0.292	0.251	0.018
4	25%	43.321	35.044	43.97	40
5	50%	68.769	75	68.5	67.95
6	75%	125	161	120	125
7	max	31937.4	5191	31937.4	10270
8	unique	20902	2515	20560	14119
9	NaN	0	0	0	0
10	NaNshare	0	0	0	0

Figure 4. Transaction Amount within the Train and Test Datasets

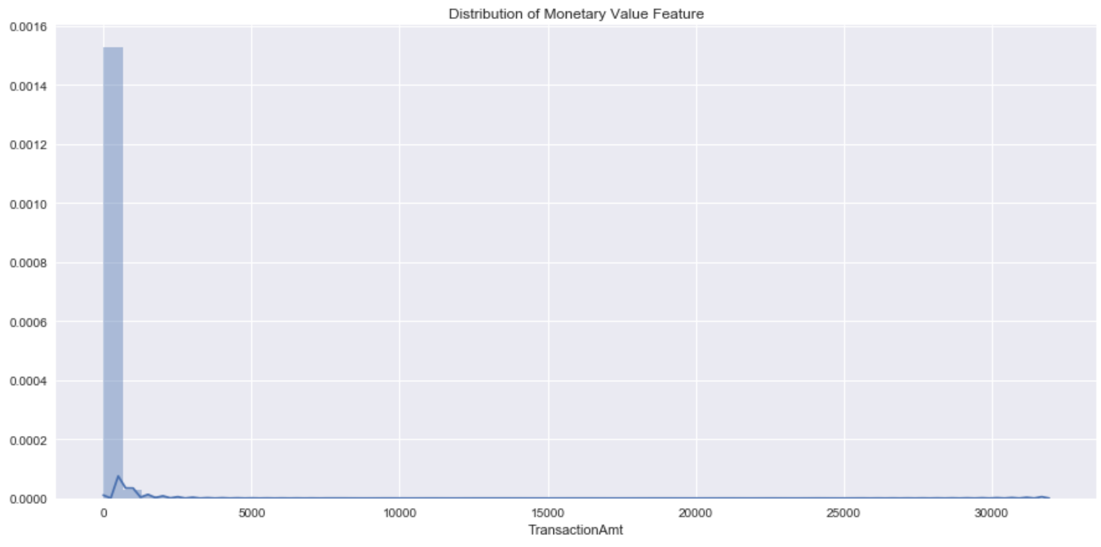


Figure 5. Distribution of Monetary Value Feature

From Figure 6, we can clearly see that whether or not the credit card has been stolen is by no means directly related to the amount of capital changing hands within each transaction. In other words, it is not true that the more people spend, the more likely the card is to be stolen.

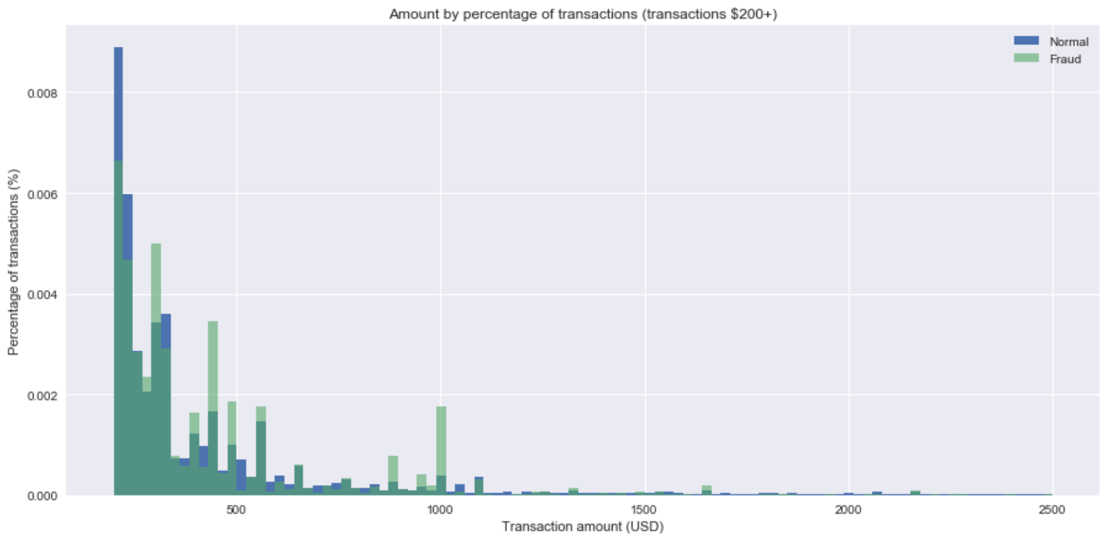


Figure 6. Amount by Percentage of Transactions (Transactions of $200+)

Another feature that deserves notice is the 'Card6' feature. This feature informs us as to whether the transaction occurred using a credit card or a debit card. According to Figure 7, credit card holders experience a higher rate of fraud than their debit card holding counterparts.

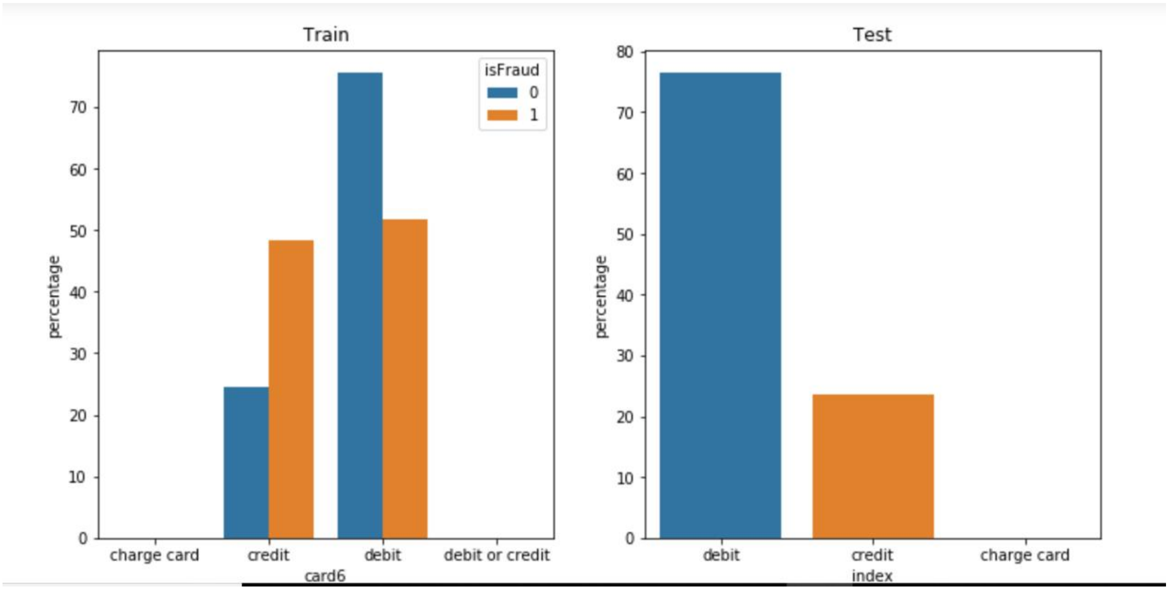


Figure 7. Card6 Feature

3.2 Missing Values

The next important process to be addressed is dealing with missing values. Within certain codes, it is clear that 414 columns contain missing values. The figure below shows the top 20 features displaying the highest missing value percentage. From the figure, we can observe that most features containing high missing value percentages are found in the identity dataset.

	Missing Ratio
id_24	99.196159
id_25	99.130965
id_07	99.127070
id_08	99.127070
id_21	99.126393
id_26	99.125715
id_22	99.124699
id_23	99.124699
id_27	99.124699
dist2	93.628374
D7	93.409930
id_18	92.360721
D13	89.509263
D14	89.469469
D12	89.041047
id_04	88.768923
id_03	88.768923
D6	87.606767
id_33	87.589494
D9	87.312290

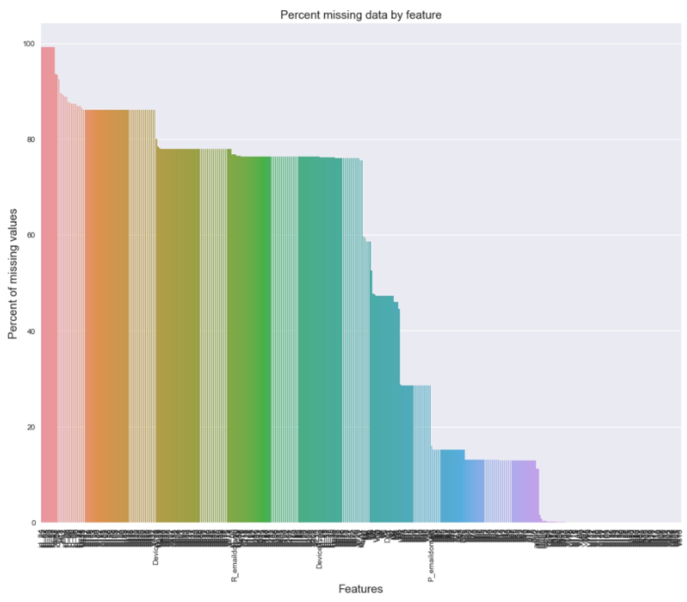


Figure 8. Missing Values

3.3 Correlation Analysis

In order to analyze the correlation between the relevant variables and the isFraud feature, we plotted the correlation heat map of these features. We discovered that C and D display a particularly high internal correlation. For example, 'C1' and 'C2', 'C1' and 'C6', 'C1' and 'C14', 'C6' and 'C14', 'C7' and 'C12', 'D4 and D12', 'D6 and D12', and 'D5 and D7', etc. The same occurs for V, but with some exceptions.

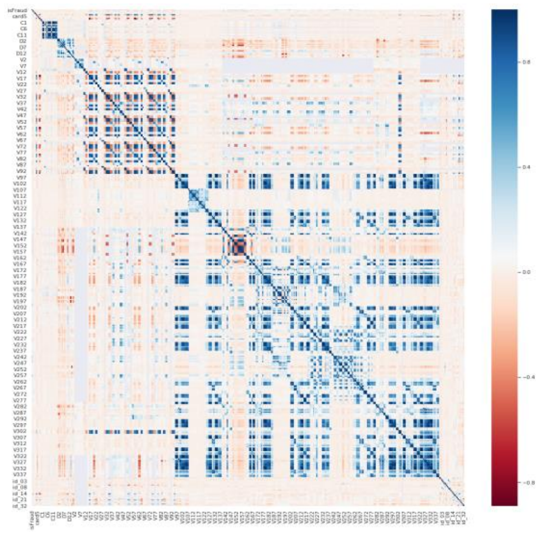
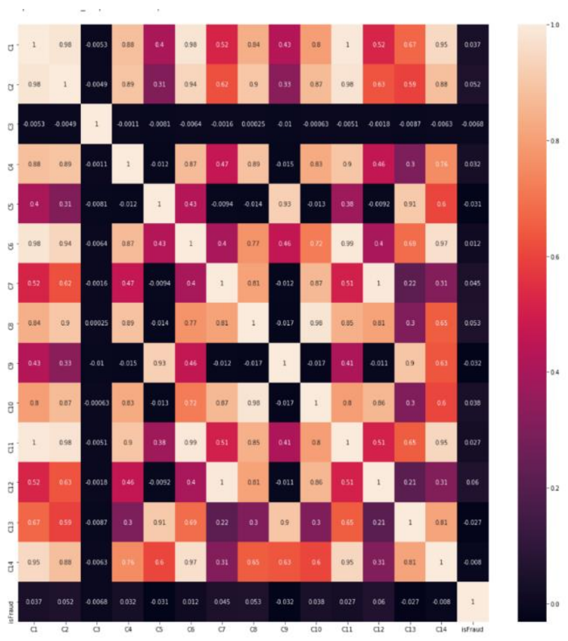


Figure 9. Correlation Matrix



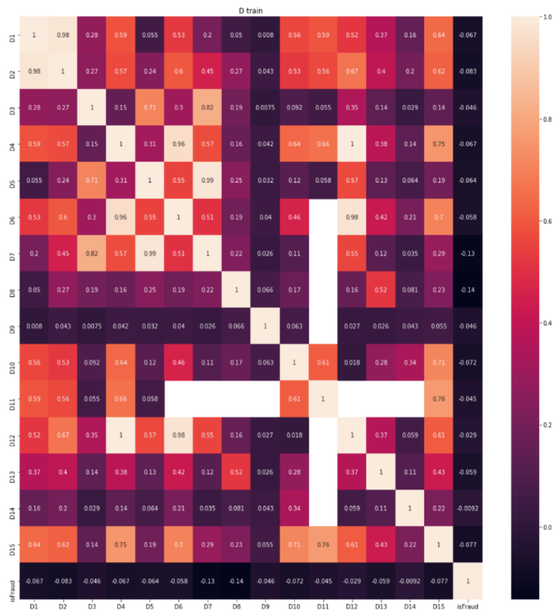


Figure 10. For C Features

Figure 11. For D Features

3.4 Feature Engineering

3.4.1 Transforming Data

Within the next algorithm to be applied, it is important to make sure that the data type be legal, which means that it becomes necessary to transform some of this data. Many unique values have to be combined with other columns in order to achieve the best score boost.

Firstly, in order to reduce data memory, we appropriately adjusted the types of numerical data supplied into 'int16', 'int32', 'int64', 'float16', 'float32', or 'float64'. Additionally, both 'TransactionDT' and 'TransactionAmt' are pure noise, but it remains crucial to combine them with other features and produce aggregated features. Thus, we transformed the date in TransactionDT into Months and another two distinct features - 'is_december' and 'is_holiday'.

3.4.2 Adding New Features

After addition, the total number of features stands at 789.

Process Name	Adding Feature(s)	Method
values_normalization	1) col +'_'+ period+'min_max' 2) col +'_'+ period+std_score	values_normalization: calculate the max col, min col, mean col, std_col in columns, and then calculate the standard deviation and finally adopt formula N=(x-min)(max-min) to normalize these data
frequency_encoding	col+'_fq_enc'	Use values_counts function to calculate the times of each value in each col of the overall columns
timeblock_frequency_encoding	col+'_'+ period +'_proportions	Use values_counts function to calculate the times of each value in each col of the overall columns, and then calculate the proportion of these values taking up the total of each columns
uid_aggregation_and_normalization& check_cor_and_remove	col+'_'+main_column+'_'+agg_type	Combine each col in the uIDs and main_columns to produce new combined features and normalize the produced values. Then calculate the correlation coefficient of each pairwise feature. For those highly correlated features, store them. For those least correlated features, remove them.

temporarily adding features： some kind of client uID	uID: card1+card2 uID2: card1+card2+card3+card5 uID3: card1+card2+card3+card5 +addr1+addr2 uID4: card1+card2+card3+card5 +addr1 +addr2+P_emaildoman uID5: card1+card2+card3+card5 +addr1 +addr2+R_emaildoman	Based on cardID and address columns, we use value_counts to calculate the times of each value in each combined feature. These newly produced features will be very specific to each client, so we need to remove it from the final features. We can use it, however, for aggregations. According to the final outcome, we speculate that card3 and card5 will represent bank country. We will also try to determine the most popular Transaction Hour and the most popular Week Day.

3.4.3 Deleting Features

Some features are pure noise and have nothing to do with the target feature 'isFraud'. So as to avoid our analysis being sullied and interfered with by these features, it's necessary to delete them. We selected the following 6 features for deletion: 'TransactionID', TransactionDT （Transformed into Months and another two new features), P_emaildomain, R_emaildomain, bank_type, and DeviceType

3.4.4 Feature Skewness

Within most of our algorithms, data displaying a Gauss distribution can effectively minimize errors. Thus, we need to determine the skewing feature and then correct it by calculating the relative asymmetry of the data.

We selected TransactionAMT to transform it. There are many unique values and the model doesn't generalize well, however, so after performing some aggregations, we can normalize the data and then log it. The normalization result is illustrated in the following chart.

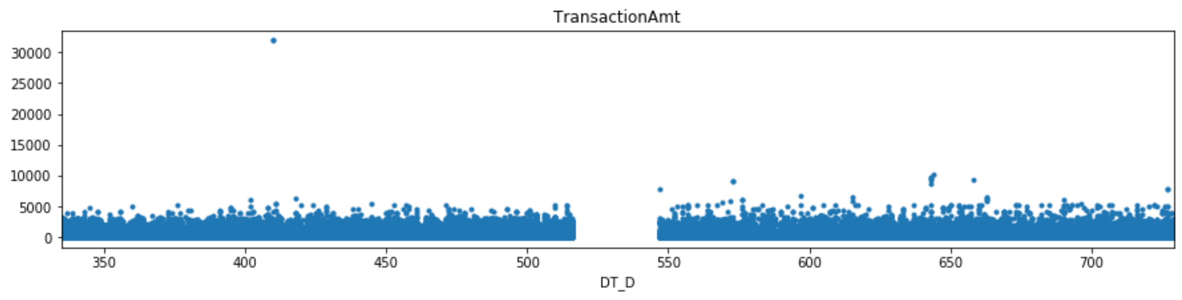


Figure 12. Skewness of the TransactionAmt

4. Models

4.1 LightGBM

The first model to be utilized is the LightGBM model, a gradient boosting framework that implements tree-based learning algorithms. Compared with depth-wise growth, the leaf-wise algorithm can converge much faster. However, the leaf-wise growth may be prone to over-fitting if not used with the appropriate parameters.

There are 8 tuning parameters: num_leaves, bagging_fraction, feature_fraction, min_child_weight, min_data_in_leaf, max_depth, reg_alpha, and reg_lambda. We set the parameters and inserted it into the model. We also subsequently used cross-validation with a group KFold. We set the value of k6 and produced a mean AUC equal to 0.9444 and out-of-folds AUC equal to 0.9431.

Later, we utilized Bayesian optimization to determine the ideal parameters, setting the value of init_points at 5 and the number of iterations at 15. Then we ran the program again with new parameters to see how different the AUC results would be from before. Ultimately, the score improved considerably when our CSV file was submitted to the leaderboard, and we achieved a score of 0.9500.

Table 1. LightGBM

Parameter	Note
Num_leaves:	The main parameter in control of the complexity of the tree model.
Bagging_fraction	The fraction of data to be used within each iteration.
feature fraction	LightGBM will select 80% of parameters randomly within each iteration when building trees.
min_child_weight	Minimum sum of instance weight (hessian) needed in a child.
min_data_in_leaf	The smallest recorded amount of data that a leaf may contain
max_depth	The maximum depth of a tree
reg_alpha	L1 regularization, double type
reg_lambda	L2 regularization, double type

4.2 XGBoost

XGBoost is a supervised learning model that relies on labeled data. It can be implemented to resolve classification and regression problems. In this competition, our objective is to predict the probability that a particular transaction is fraudulent, and thus it is very much a classification problem. As we have mentioned before, the correlations amongst the features are strong, and thus it's better to use tree models.

Decision trees, as base learners, are composed of a series of binary questions and predictions occur within the leaves of the trees. However, there exists severe overfitting problems. To prevent overfitting, XGBoost uses CART: a combination of classification and regression trees. Additionally, XGBoost uses the boosting method to learn the trees. Defining an objective function and optimizing it are always the foundation ideas, and the objective function contains training loss and regularization. The biggest difference between XGBoost and other tree models is that it takes the Taylor expansion of the loss function up to the second order.

We employed GroupKFold as our cross-validation method. It's one of the many scikit-learn cross-validation objects. In GroupKFold, the same group will not appear within two different folds, and the folds are approximately balanced in the sense that the number of distinct groups is approximately the same in each fold.

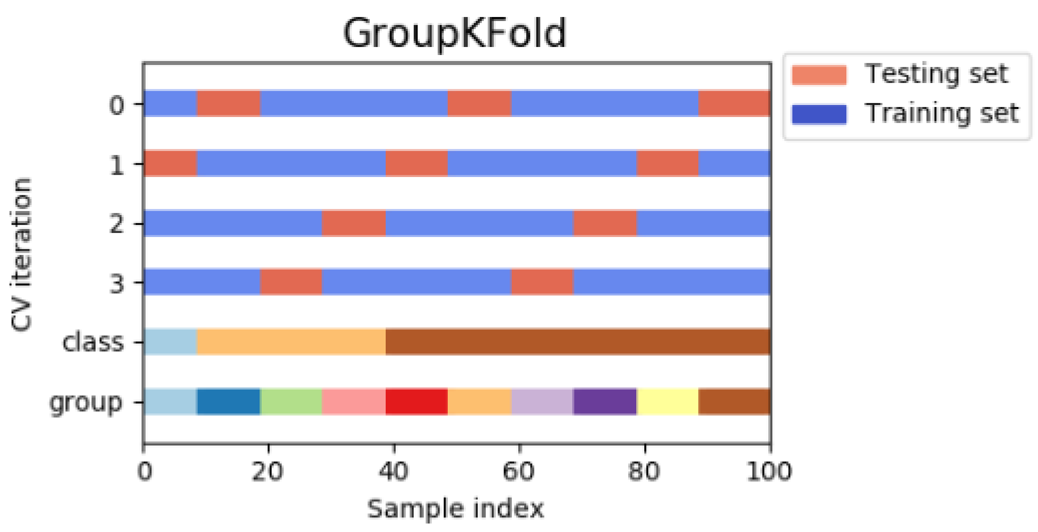


Figure 13. Graphical Explanation of GroupKFold

For XGBoost, there are three types of parameters: general parameters, booster parameters, and task parameters.

Table 2. XGBoost (before tuning)

Parameter	Note
eval_metric	General Parameter. Evaluation metrics for validation data. We choose AUC (area under the curve).
Objective	Learning task parameter. We chose binary: logistic, which means logistic regression for binary classification and output probability.
Booster	General parameter, or which booster to use. We chose gbtree which uses tree-based models.
Nthread	General parameter. Number of parallel threads used to run XGBoost. Our value is 4.
Eta	Parameter for Tree Booster. Step size shrinkage used in an update can prevent overfitting. After each boosting step, we can directly gain the weights of any new features, and eta shrinks the feature weights in order to make the boosting process more conservative. We chose 0.048.
Max_depth	Parameter for Tree Booster. Maximum depth of a tree. Increasing this value will cause the model to become more complex and more likely to overfit. Our max depth is 9.
Subsample	Parameter for Tree Booster. Subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees and this will prevent overfitting. Subsampling will occur once within every boosting iteration. We set it to 0.85.
colsample_bytree	Parameter for Tree Booster. It is the subsample ratio of columns when constructing each tree. Subsampling occurs once every tree constructed. We also set it to 0.85.
Alpha	Parameters for Tree Booster. L1 regularization term on weights. Increasing this value will cause the model to become more conservative. We set it to 0.15.

Lambda	Parameters for Tree Booster. L2 regularization term on weights. Increasing this value will cause the model to become more conservative. We set it to 0.85.
Verbosity	General Parameters. Verbosity of printing messages. Valid values are 0 (silent), 1 (warning), 2 (info), 3 (debug).

In the first round, our mean AUC is 0.934942 and out-of-folds AUC is 0.935065. We decide to list the features' importance and delete those that have low importance. After deleting the features whose importance is lower than 95%, mean AUC improved to 0.935516 and out of folds AUC improved to 0.936867. To further improve the score, we also implement Bayesian Optimization to determine the optimal parameters for XGBoost. For computationally intensive tasks, grid search and random search can be painfully time-consuming, with less success in determining optimal parameters. These methods rely on hardly any information that the model learned during the previous optimizations. Bayesian Optimization on the other hand constantly learns from previous optimizations to find a best-optimized parameter list and also requires fewer samples to learn or derive the best values from. The parameters of Bayes Optimization are listed below.

Table 3. Bayes Optimization

init_points	Number of randomly chosen points to sample the target function before Bayesian Optimization is capable of fitting the Gaussian Process. We set it to 3.
n_iter	Total number of times the Bayesian Optimization is to repeat. We set it to 7.
Acq	Acquisition function type to be used. Can be "ucb", "ei" or "poi". We chose ucb. ucb: GP Upper Confidence Bound ei: Expected Improvement poi: Probability of Improvement

After the optimization, better parameters are listed below:

Table 4. XGBoost(after tuning)

Parameter	Note
max_leaves	General Parameter. Maximum number of nodes to be added. Only relevant when grow_policy=lossguide is set. The optimization value is 525.
min_child_weight	Parameter for Tree Booster. Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning. In a linear regression task, this simply corresponds to the minimum number of instances needed to be in each node. The larger the min_child_weight is, the more conservative the algorithm will be. The optimization value is 0.01.
Subsample	Optimization value is 0.9.
colsample_bytree	Optimization value is 0.9.
Alpha	Optimization value is 0.3.
Max_depth	Optimization value is 15.
reg_lambda	Optimization value is 1.

After using the new optimization parameter, we found that the AUC was indeed higher. Mean AUC is 0.939119 and out-of-folds AUC is 0.938543. The leaderboard performance stands at 0.9451.

4.3 CatBoost

CatBoost (Categorical Boosting) is a gradient boosting model that is particularly adept at processing categorical features. Its learning algorithm is based on GPU while its scoring algorithm is based on CPU.

In gradient boosting algorithms, the most common method is to transform the categorical features into numerical features. Mostly categorical features will change into one or many numerical features. If the base of a categorical feature is low, which means it only contains a few categories, people often use one-hot encoding to transform it (using one-hot encoding for all features with a number of different values less than or equal to the given parameter value).

We can use statistics to transform categorical features into numerical features. For example, we can first calculate the sum of categories necessary for a specific feature that has the value v, and then we can divide the sum by the sample size of features which has the value v. However, this can lead to overfitting, as if v only contains one sample. To prevent overfitting, we need to split the sample into two parts: one specifically for statistics, and one for training, and therefore both the statistics set and the training set can be effectively minimized.

To use all the samples for training, CatBoost first uses random permutations to rank all the samples, and for each example, we compute the average label value with the same category value placed before the given one in the permutation.

Table 5. CatBoost

Parameter	Note
Iterations	The maximum number of trees that can be built when solving machine learning problems. Ours is 20000.
learning_rate	Used for reducing the gradient step. Ours is 0.02. After Bayes Optimization, it turns out to be 0.01.
depth	Depth of the tree. We choose 9. After Bayes Optimization, it turns out to be 15.
l2_leaf_reg	Coefficient at the L2 regularization term of the cost function. We choose 40. After Bayes Optimization, it turns out to be 20.
Bootstrap type	Define the method for sampling the weights of objects. We use Bernoulli.
subsample	Sample rate for bagging. This parameter can be used if one of the following bootstrap types is selected: Poisson and Bernoulli. We choose 0.85 at first. After Bayes Optimization, it turns out to be 0.7211.
loss_function	The metric to use in training. The specified value also determines the machine learning problem to solve. We use logloss.
early_stopping_rounds	Set the overfitting detector type as 'Iter' and stop the training after the specified number of iterations in order to determine the optimal metric value. We choose 100.

After using the optimization parameter, we found that the AUC was also considerably higher. Mean AUC stood at 0.696722 and out-of-folds AUC at 0.697475. The leaderboard performance is thus 0.9447.

4.4 Neural Network

Fundamentally, a neural network is simply a mathematical function that takes a variable in and gives you another variable back, and both of these variables could be vectors. Neural networks are good feature extractors, especially on unstructured data, such as sound, images, and video. However, the data in this contest is structured. So neural network doesn't perform as well as tree models.

The basic components of neural networks are activity, weight, bias, and activation functions. Within the sequential API of Keras, there are some important parameters related to these components.

Table 6. Neural Network

Parameter	Note
Batch_size	To specify a fixed batch size for your inputs (this is useful for stateful recurrent networks). Number of samples per gradient update. We set it to 2048.
epochs	Integer. Number of epochs to train the model. An epoch is an iteration over the entire x and y data provided. We set it to 8.
activation	Activation functions are the functions that decide whether the neuron should fire or not. We chose ReLu as the activation function because it gives an output of x if x is positive and 0 otherwise.

The most successful NN model in the competition added a feature called "uID" and adding uID to the first layer only served to slightly improve the local CV but adding it to the last layer provided a significant boost. However, we didn't find the uID so the score was not that high. We only chose LGB, XGB, and CAT for the ensemble.

Table7. Summary of all Models

Model	Description	Pros	Cons
Catboost	• an algorithm for gradient boosting on decision trees. • a machine learning algorithm that allows users to quickly handle categorical features for a large data set • used to solve regression, classification and ranking problems	• CatBoost boasts flexibility, giving indices of categorical columns so that it can be encoded as one-hot encoding • CatBoost uses an efficient method of encoding for categorical columns that have a unique number of categories greater than one_hot_ max_size	• Slow training speed
LightGBM	• An advanced implementation of gradient boosting. • Employs a more regularized formalization to control over-fitting which encourages a better performance within the model.	• Faster training speed and higher efficiency • Lower memory usage • Superior accuracy • Parallel and GPU learning supported • Capable of handling large-scale data	• Easily over-fitted
XGBOOST	• An advanced implementation of gradient boosting. • Employs a more regularized formalization to control over-fitting which produces a better performance within the model.	• Uses regularization to reduce over-fitting. • Supports parallel processing. • Allows for splits up to the max_depth specified and then starts pruning the tree backwards and removes splits beyond which there is no positive gain.	• Susceptible to outliers. • Lack of interpretability and higher complexity. • Comparatively difficult to tune the parameters. • Slow training speed.
Neural Network	• Its idea is to take a large number of training examples, and then develop a system that can learn from those training examples.	• It's particularly adept at analyzing unstructured data such as images and videos.	• It's too much of a black box, making it difficult to train. • It's no substitute for understanding the problem in-depth.

5. Averaging Based Ensemble Methods

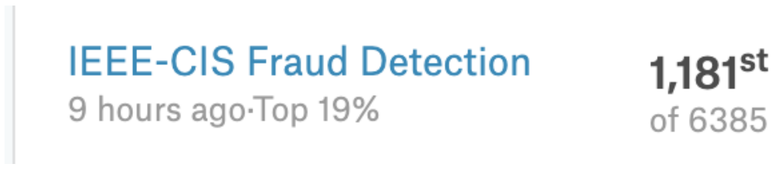
The implementation of only one model cannot independently produce the best score, so we decide to ensemble all relevant and useable models in order to determine the best, most effective, and most efficient way to predict possible outcomes.

Widely known methods of ensembling are voting, stacking, bagging and boosting. We implemented a voting and averaging based ensemble method, or the simple averaging method.

Within the simple averaging method, for every sample within the test dataset, the average predictions are calculated. We chose samples that show the least correlations and from various different models. This method often reduces over-fitting and produces a smoother regression model. Ultimately, the best score is produced by the combined efforts of XGBoost, CATBoost, and LightGBM.

6. Final Score

We achieved a score that ranked in the top 19% of amongst 6,381 international teams.


IEEE-CIS Fraud Detection
9 hours ago·Top 19%
1,181st
of 6385