**Report On**

**IEEE-CIS Fraud Detection Competition**

**1155092111 ZHONG Yaoxi​**

**1155092065 ZHANG Yufei​**

**1155091863 CUI Tianye​**

**1155092133 SONG Ziwei​**

**1155092023 CHEN Anni**

**1.Introduction**

The IEEE-Fraud Detection Contest is a competition for supervised binary classification problems. The host provides a dataset with around 1100k transactions and 400 variables. 600k of them are the training dataset with known ‘isFraud’ and the other 500k transactions are the testing data. Our goal is to use some statistical techniques to predict whether a transaction in testing dataset is fraudulent or not.

**1.1 Data Description**

**1.1.1 Transaction Table**

The transaction table includes all information related to each transaction, including the time, the card information, the purchase, etc. There are some essential variables for making up user IDs.

card1: Issuer Identification Number. ​card2: bank branch.​ card3: country. D1: time delta (days, rounded down) since the first transaction for one card​. D4: time delta since the first transaction for all cards on the account. Addr1: purchaser's billing region. Vxxx: Vesta engineered rich features, including ranking, counting, and other entity relations. All Vesta features were derived as numerical and some of them are count of orders within a clustering, a time-period or condition, so the value is finite and has ranking.

**1.1.2 Identity Table**

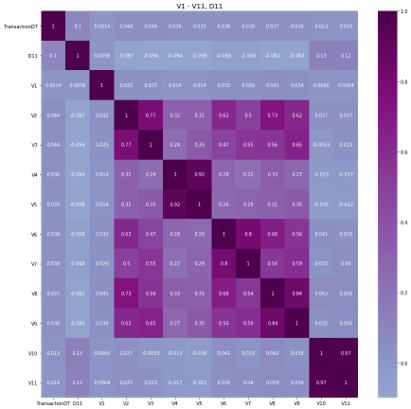
Variables in the identity table are identity information, including network connection information and digital signature associated with transactions.

**2 Methods**

**2.1 Variable Selection**

At the very beginning, we tried to use the simplest method to conduct variable selection: point-biserial test form numerical variables and the Pearson correlation test for others. We finally selected 47 variables (33 numerical variables). Then we used these variables and the models introduced below to make predictions, but the result was not satisfactory. After discussion, we thought the reason may be that there are nearly 600k observations, therefore, the overfitting problem caused by using too many variables in predicting may not be significant, while using only a small number of variables may lose some important information. We have tried several variables selection methods (including several popular kernels in Kaggle), but the result was no better than using all variables.

However, we met another problem: when using all variables, the training process will be quite time-consuming. When we computed the missing rate for all variables, we noticed that many variables have the same missing rate, by small groups of 2-10. For a specific group (for example, id\_15 and id\_16), we compared their missing entries and discovered, as expected, the missing entries are exactly the same. Although the meanings were masked, we believed variables with the same missing rate must have some inner connections. Therefore, an intuitive idea is to group variables with the same missing rate, and we computed the correlation among them (see Figure 1). If several variables are highly correlated (for example, V4 and V5), then we just kept one and dropped the others.

Figure 1

If the group is constructed by categorical variables with very few categories (for example 2-10), one of our methods is to try to encode them by n-ray (binary, ternary) numbers. There are three advantages. First, sometimes, we are delighted to find encoded variables showing essential features. Second, the encoded variables give us more hints on how to assign values to missing entries instead of assigning nonsense value such as -999. Third, model complexity is reduced, so the training process speeds up.

**2.2 Feature Engineering**

**2.2.1 TransactionDT**

The holder provided a very important information on the discussion board: All the transactions are from 1st Dec 2017 to 120 days later. So we are able to extract the TransactionDT to month, week-day and day-hour. By comparing isFraud-rate for each hour, an important feature was shown.

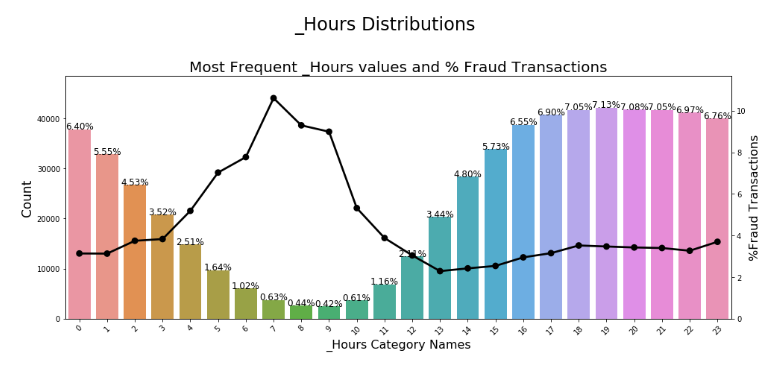


Figure 2

**2.2.2 Email binning**

Each of the purchaser and recipient email domain features has around sixty categories. By exploratory data analysis, we found that some of the email domains from the same company share a similar fraudulent rate. To make the features more robust in our model, we combined the email domains which belong to the same company into a new category. For example, “icloud.com” “mac.com” “me.com” were transformed into a new category named “apple”. In this way, the cardinality of email domains feature was reduced to eleven.

**2.2.3 Frequency encoding**

Since the train and test datasets have different levels in some of the categorical features, we first combined them and then replaced the categories with their counts. To stabilize the training processes, we divided them by the most frequent category to obtain normalized values. The reason for using count encoding was that it helps the tree-based model, like LightGBM, to detect outliers which occur infrequently since LightGBM cannot detect this type of outliers on its own.

**2.2.4 User ID**

In order to further improve our feature engineering methods and auc score, the next essential step is to find a user ID that can help identify each user from millions of transactions. From the Kaggle discussion–board, a key hint has been given by the holder for the competition, saying that once a transaction is detected to be fraud, the corresponding user and his email/billing address will be linked to attributes indicating fraud in the following 120 days.

Taking UID as the secret feature which is constructed of several given features but carries extra information, there are several approaches to form a UID. Obviously, it is better to build a UID telling the account information according to the hint. However, the only factor we can find directly relating to a specific account is D4n (first transection date of the account), since under the same account there might be several cards with different owners (like husband and wife). Another approach is to identify a specific card user. Here in this case we combine the card features, address1, D1n (first transection date of the card) and purchaser email address as our final user ID.

**2.2.5 Data Aggregation**

Data aggregation consolidates vast amounts of detailed data into higher levels of dimension hierarchies which makes it easier to manage data and extract meaningful information. Following the logic behind finding user IDs, it’s intuitive to explore the transactions with the same user IDs. The aggregation is based on the assumption that the transactions with the same userID has the same label. To verify the assumptions, we did analysis on the fraud proportion within the same userID. As we expected, only 0.6% of the training data have a mixture of fraud and nonfraud which might because fraudulent credit cards get terminated instead of reused. ​96.95% of the card are always nonfraud while 2.45% are always fraud (Figure 3). ​In order for easier modelling, we neglected the possibility that the fraud label will change for the same user.

Figure 3Figure 4

The plot showed 87.6% of the cards in the testing dataset first appear while only around 12% data we've already know form the training dataset.​ Therefore, the key for the competition is to build a model that can predict unseen cards.​ Hence, we calculate the average and standard deviation of transaction amount (TransactionAmt​), the distance between the bank and the where he used the card, time delta (D features) and card information (D features). After the calculation of the average and the standard deviation, we discarded the userIDs. ​

**2.2.6 Handling categorical random variables**

If the group is constructed by categorical variables with very few categories (for example 2-10), one of our methods is to try to encode them by n-ray (binary, ternary) numbers. There are three advantages. First, sometimes we are very delighted to find encoded variables showing important features. Second, the encoded variables give us more hints on how to assign values to missing entries instead of assigning nonsense value such as -999. Third, model complexity is reduced, so training process speeds up.

**3. Models**

**3.1 LightGBM**

LightGBM is a gradient boosting framework that uses tree-based learning algorithms. Compared to other level-wise trees, LightGBM grows leaf-wise which can reduce more loss. Since this project involves more than 400 variables with almost a million of observation, LightGBM works well for its high speed and accuracy. The following are the key parameters used in the LightGBM models of our project.

‘application’ is the most important parameter and specifies the application of the model, whether it is a regression problem or classification problem. Obviously for our project, we should indicate the classification problem to LightGBM.

‘boosting’: defines the type of algorithm you want to run. Here we used traditional Gradient Boosting Decision Tree (gdbt).

num\_leaves: number of leaves in full tree. We have tried different values for this parameter, the most commonly used are 197 and 256.

Although the LightGBM works well for our project for its space-saving and high-accuracy characteristics, we should be alert to its probability of overfitting. According to my experience, if the data set is not big enough or one just throw the row data into LightGBM, it will return a almost 0.99 local auc but quit low scores for prediction.

**3.2 XGBoost**

XGBoost is a supervised learning algorithm that implements a process called boosting to yield accurate models. Boosting refers to the ensemble learning technique of building many models sequentially, with each new model attempting to correct for the deficiencies in the previous model. In tree boosting, each new model that is added to the ensemble is a decision tree. XGBoost provides parallel tree boosting (also known as GBDT, GBM) that solves many data science problems in a fast and accurate way.

Since XGBoosts does not accept ‘character’ variable-type as input, we use the one hot encoding and frequency encoding techniques for categorical variables mentioned above to deal with it. For those categorical variables with many categories, the ‘unclass’ function in R is very useful for converting them. Also, we tried different combinations of userID features provided in the last section to find the best single model.

Here, we point out some important parameters. ‘max\_depth’ secify the maximum tree depth. Higher values will make the model more complex and can lead to overfitting. ‘categorical\_encoding’ is a useful option specifies the encoding scheme to use for handling categorical features and the algorithm will automatically perform one-hot encoding. ‘scale\_pos\_weight’ controls the balance of positive and negative weights, useful for unbalanced classes. max\_delta\_step is maximum delta step we allow each tree’s weight estimation to be. If it is set to a positive value, it can help making the update step more conservative.

**3.3 Catboost**

Similar to previous two package, the Catboost also use the gradient boosting on decision trees. Its advantage is that it is ‘smarter’ when dealing with categorical data. Most packages use one-hot encoding. when categorical features don't have a lot of values, one-hot encoding works well. However, in this competition, some variables, such as the email variables, there are more than sixty levels. Instead, the Catboost will convert categorical variables into numerical variables mainly using three statistics: "

CountInClass: how many times the label value exceeded for objects with the current categorical feature value. It only counts objects that already have this value calculated (calculations are made in the order of the objects after shuffling).

TotalCount: the total number of objects (up to the current one) that have a feature value matching the current one.

Prior: a number (constant) defined by the starting parameters.”

# To use it, we have to use python, since it is very complicated to install it in R. Before training, we need to convert all float values to integer and replace missing value with –999(since –999 is far less than normal data, we can guarantee that the model will treat it as a special category in each split. That’s because Catboost use hash table to store the values, and the hash value for “1” is different from the hash value of ‘1.0’, similarly hash value for ‘NA’ and ‘NaN’ are also different. Here I followed the Kaggle public kernel‘Basic EDA’ and ‘Catboost - IEEE fraud’. Then I just used all the categorical features to train the model.

**3.4 Neural network**

**3.4.1 Neural network**

* A neural network is a computational learning system inspired by biological neural networks in the human brain. It uses a network of functions to understand and translate a data input of one form into a desired output. As one of the tools and approaches widely used in machine learning algorithms, neural networks are being applied to many real-life problems today, for example speech and image recognition.
* In this competition, we managed to train a neural network model with 0.85 leaderboard score. This section covers the data preprocessing for NN, model building procedure including model structure and hyperparameter tuning and our reflections on it.

**3.4.2 Data processing**

* Features used in NN model include original features selected previously in the feature selection step, and some engineered features including hour, weekday and uid.
* Data preprocessing is required to transform the data into the format that neural network can understand. For numerical features, log transformation and standardization were performed to reduce the impact of some extreme values and to stabilize the training process. Then missing values were filled by the mean zero. For categorical features, firstly, missing values were filled by “miss” as a new category. After reducing the cardinality of some features (e.g. card1, card3), one hot encoding was applied. Since the engineered feature UIDhas a high cardinality with around eight hundred thousand unique values, count encoding was applied to it for later use.

**3.4.3 Model building**

* A five-fold cross validation scheme was established, and the average auc was calculated. Since UID was a high cardinality feature, it was handled by using embedding layer and concatenating the embedding layer with the last dense layer in NN. To prevent overfitting, dropout=0.2 was used for both hidden layers. The ‘EarlyStopping’ callback was also used to monitor the model performance.
* Despite that in the case of structured data, NN can be much harder to understand and it requires much time finetuning the parameters. It is still manageable and will definitely add some diversity to the results. If given more time, we will further tune the parameters and explore the better structures. Besides, we anticipate a better performance by adding *uid* related aggregation features into the model. Another possible improvement is to blend different NN models with different loss functions. Although we could not make a good use of neural nets in this competition, we did get some insights into it. Hopefully our efforts on it will pay off someday.

**4. Result**

**4.1 Results on single models**

Among all the models, LightGBM gave the best performance not only on accuracy but also on the fast speed. More than five LightGBM models were trained with different feature engineering and parameters. All of them gave at least 0.94 on public leaderboard, which are rather good scores for single models. For final usage, we just picked two of the highest scores to further ensemble.

We also trained 2 XGBoost models. For the choice of parameters, we learned from others’ XGBoost models on the discussion board. For example, at first, we set eta (the learning rate) to be 0.3. The convergence rate of AUC is very fast which led to over-fitting. What people usually did on discussion board is setting eta=0.02-0.05. We also noticed that. After some slight modification considering our computing power and problems of over-fitting, we get our desired results. The public scores of XGBoost models with different UIDs and EDAs are ranging from 0.936-0.947.

As for Catboost, we just used all the categorical features to train the model. As mentioned above, it has a better capability to handle categorical variables. It scored 0.943 in public leaderboard. Concerning the fact that it just made use of a few variables and didn’t sue UID, that is already very promising.

**4.2 Results on blending**

There are many methods to ensemble models. The easiest way for us to improve our models was directly manipulating the prediction result from different single models with different UIDs and EDAs. Simple weighted average led to an upgrade.

public\_score=(0.948,0.942)

model\_type=(LightGBM, XGBoost,).

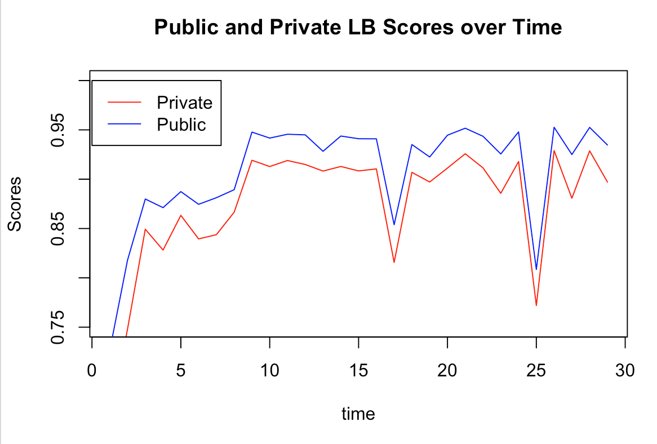
w=(0.5, 0.5)

We first tried simple arithmetic mean. As expected, we had an upgrade on our public score from 0.9477 to 0.9488. Then we added the Catboost’s prediction. Then we got 0.9516. That is an interesting result since the Catboost’s score was not very high, compared with the others. The reason was that the algorithms of Lightgbm and XGBoost are similar, and we found their predictions’ correlations is 0.93. However, the logic of Catboost is different (by its unique way to handle categorical variables), and the correlation is lower (0.87). Therefore, there was a big improvement in our public score. After that, we successfully trained another 2 models with different UIDs. The last day, we had 5 different single model results and adjusted the weight of different models and finally 0.9526 was our best public score, given by:

public\_score=(0.948,0.942,0.945,0.943,0.923)

model\_type=(LightGBM, XGBoost, XGBoost, LightGBM, Catboost)

w=(0.46, 0.26, 0.02, 0.03, 0.23)



**4.3 Final submission**

At the time to make final submissions, we were facing many choices. We only knew the public scores; the private scores remained mysterious. However, we were roughly confident that our model had good generalization ability and there should be a positive correlation between the public and private leaderboard. We decided to choose the best single model and best blending one based on the public score. The private result turned out that we were true: For most submissions their private scores were lower than public scores by 0.025~0.035 and their relative ranks was nearly unchanged.

**5. Limitations**

**5.1 Reflections on Neural Network model**

Despite that in the case of structured data, NN can be much harder to understand and it requires much time finetuning the parameters. It is still manageable and will definitely add some diversity to the results. With more time, we would be able to further tune the parameters and explore better structures. Besides, we anticipate a better performance by adding UID related aggregation features into the model. Another possible improvement is to blend different NN models with different loss functions. Although we could not make a good use of neural networks in this competition, we did get some insights on how to use it on this type of data. For example, transforming the information contained in the data into some images and feeding them to the neural nets could be a potential approach.

**5.2 Hardware limitation**

When using R in our laptops, the training process is very slow due to the file size. However, we can use the Google Cloud Platform to accelerate it. As can be shown in the picture below, we create a R environment with 96v GPUs and 360 GB RAM, which is impossible for personal computers.



**6 Further improvement**

**Adversarial Validation**

As we found that for some factors, there exists huge differences on data distributions and missing rates between training data set and testing data set as well as even between the test public and private, so we have tried one-hot encoding to identify them in order to avoid overfitting to the training data. Unfortunately, one-hot encoding did not improve our models for the reason that most our boosting models did not do well with it and finally we only applied k-fold cross-validation for our model. Since the dataset is heavily skewed, it’s better to use adversarial validation method which can efficiently select training samples that are most similar to test samples to circumvent overfitting. Based on the adversarial validation, we can make use of the AUC of the training data which most similar to test data to predict the performances of our models on the real testing data.

**7. Conclusion**

It is quite challenging for us to finish the competition within three weeks. Although we have struggled a lot, finally we got a valuable opportunity to have a taste of how complex the real-world data analysis can be. We did learn a lot from other kagglers and finally managed to come up with our own ideas to train a better model for fraudulent detection. According to the competition description, Vesta guarantees more than $18 billion dollars in transactions annually. It means that at 3% fraud, $540 million dollars are stolen if the money cannot be reversed. Therefore, even 1% increase of performance means a lot.

**References**

1. Feature Engineering Techniques. Retrieved from: <https://www.kaggle.com/c/ieee-fraud-detection/discussion/108575#latest-6433951>
2. "What is LightGBM, ow to implement it? How to fine tune the parameters?”. Retrieved from:

<https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc>

1. XGBoost Parameters. Retrieved from:

<https://xgboost.readthedocs.io/en/latest/parameter.html>

1. 8th place NN solution Overview. Retrieved from:

<https://www.kaggle.com/c/ieee-fraud-detection/discussion/111476#latest-646755>

1. Catboost document. Retrieved from:

<https://catboost.ai/docs/>

1. CatBoost: unbiased boosting with categorical features. Retrieved from:

<https://arxiv.org/pdf/1706.09516.pdf>

**Appendix**

https://docs.google.com/document/d/1gkpF89FPNm03Y49fHGQvtM4gjhwj4IrMQVpJ354djBY/edit?usp=sharing