COMP 440

Machine Learning Tools and Techniques

Assignment 3: Kaggle Competition

IEEE-CIS Fraud Detection

Qiangqiang Li(Aaron)

Supervisor(s): Will Browne

ID: 300422249

Part 1: Core: Exploring and understanding the Kaggle process

- ***Business understanding***

IEEE-CIS partnering with the world's leading payment service company, Vesta Corporation, seeking the best solutions for fraud prevention industry. The data comes from **Vesta's** real-world e-commerce transactions and contains a wide range of features from device type to product features. **The goal** is using Artificial Intelligence methods to impacts payment card fraud detection for improving the efficacy of fraudulent transaction alerts for millions of people around the world.

- ***Data understanding***

The data were predicting the probability that an online transaction is fraudulent, as denoted by the binary target **isFraud(0 or 1)**. The data is broken into two files identity and transaction, which are joined by **TransactionID**. Not all transactions have corresponding identity information. **The first need to merge, become two datasets**: **train_merge.csv and test_merge.csv.**

- ***Data preparation：***

1. Effecting Data Minification (Memory Reducer):

Because the size of the dataset is too big, there are more float64 and int64 variables, which can be compressed, we are trying to make the dataset smaller without losing information. **Reason behind memory Reduction: Int16: 2 bytes; Int32 and int: 4 bytes; Int 64: 8 bytes.** This is an example how different integer types are occupying the memory. In many cases it is not necessary to represent our integer as **int64 and int32** it is just waste of memory. Here I am reducing the memory of the dataset by **66.8%** without losing the data. This will have a great impact while training our model. It will reduce the training time by a very large margin.

```
Mem. usage decreased to 648.22 Mb (66.8% reduction)
Mem. usage decreased to 563.43 Mb (66.3% reduction)
```

2. Exploratory Data Analysis

a. Looking Data type and imbalance data.

Train_merge.csv contains 590540 rows of data, and test_merge.csv contains 506691 rows of data. In the train data, the binary target: isFraud =0 at 96.5%, while the isFraud=1 at 3.5%. this is mean that the data is **imbalance.** Besides, **using info() method** was used to print information about the DataFrame.

```
(590540, 433) (506691, 432)
```

	0	1
isFraud	0.96501	0.03499

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 590540 entries, 2987000 to 3577539
Columns: 433 entries, isFraud to DeviceInfo
dtypes: float16(354), float32(45), int16(1), int32(1), int8(1), object(31)
memory usage: 648.2+ MB
```

b. Missing data

.isnull().sum() can find the number of each column has many NULL ,and using **missingno.bar()** to visualize the distribution of NaN values. The missing data will affect the accuracy of the data analyze. **'TransactionID', 'isFraud', 'TransactionDT', 'TransactionAmt', 'ProductCD', 'C_'** no missing values,**'D_', 'M_' , 'id_'** is relatively sparse. The **'V_'** part has many missing values, and some are sparse, but most of them are missing the same number, such as **V1-V7, V322-V337** believes that they are more correlated with each other.

isFraud	0
TransactionDT	0
TransactionAmt	0
ProductCD	0
card1	0
card2	8933
card3	1565
card4	1577
card5	4259
card6	1571
addr1	65706
addr2	65706
dist1	352271
dist2	552913
P_emaildomain	94456
R_emaildomain	453249

C1	0
C2	0
C3	0
C4	0
C5	0
C6	0
C7	0
C8	0
C9	0
C10	0
C11	0
C12	0
C13	0
C14	0

D1	1269
D2	280797
D3	262878
D4	168922
D5	309841
D6	517353
D7	551623
D8	515614
D9	515614
D10	76022
D11	279287
D12	525823
D13	528588
D14	528353
D15	89113

M1	271100
M2	271100
M3	271100
M4	281444
M5	350482
M6	169360
M7	346265
M8	346252
M9	346252
V1	279287
V2	279287
V3	279287
V4	279287
V5	279287
V6	279287
V7	279287

V322	508189
V323	508189
V324	508189
V325	508189
V326	508189
V327	508189
V328	508189
V329	508189
V330	508189
V331	508189
V332	508189
V333	508189
V334	508189
V335	508189
V336	508189
V337	508189

id_01	446307
id_02	449668
id_03	524216
id_04	524216
id_05	453675
id_06	453675
id_07	585385
id_08	585385
id_09	515614
id_10	515614
id_11	449562
id_12	446307
id_13	463220
id_14	510496
id_15	449555
id_16	461200

c. Statistical Information (Numerical Attributes Exploration)

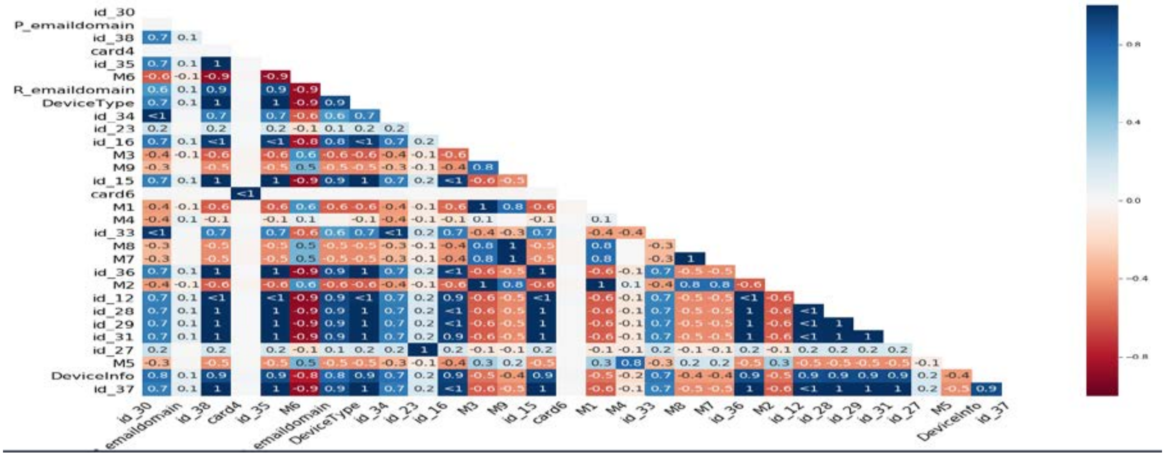
describe() method was used to generate descriptive statistical information of each attributes in the dataset. The table showed **the maximum and minimum** value for each column are out of range of common sense and business understanding, **such as C6, C7**, there are **lot of NaN data**. The min data and the max data have t**he huge different**. **Outliers values** can use the average to replace the NA or very lager value. Because **Outliers values will affect the correct information.**

	isFraud	TransactionDT	TransactionAmt	card1	card2
count	590540.000000	5.905400e+05	590540.000000	590540.000000	581607.0
mean	0.034990	7.372311e+06	NaN	9898.734658	NaN
std	0.183755	4.617224e+06	NaN	4901.170153	NaN
min	0.000000	8.640000e+04	0.250977	1000.000000	100.0
25%	0.000000	3.027058e+06	43.312500	6019.000000	214.0
50%	0.000000	7.306528e+06	68.750000	9678.000000	361.0
75%	0.000000	1.124662e+07	125.000000	14184.000000	512.0
max	1.000000	1.581113e+07	31936.000000	18396.000000	600.0

C5	C6	C7
590540.0	590540.0	590540.0
NaN	NaN	NaN
NaN	NaN	NaN
0.0	0.0	0.0
0.0	1.0	0.0
0.0	1.0	0.0
1.0	2.0	0.0
349.0	2252.0	2256.0

- ***3. Feature interaction---correlation.***

Using missingno.heatmap(). The heatmap tells the correlation between the missing nature of the data. Like the below picture: **Id_38 and id_35, DeveiceType** have a same relationship in the correlation map. **M7 and M8** have a same relationship in the correlation map. **Id_28, id_29 and id_31** have a same relationship in the correlation map.

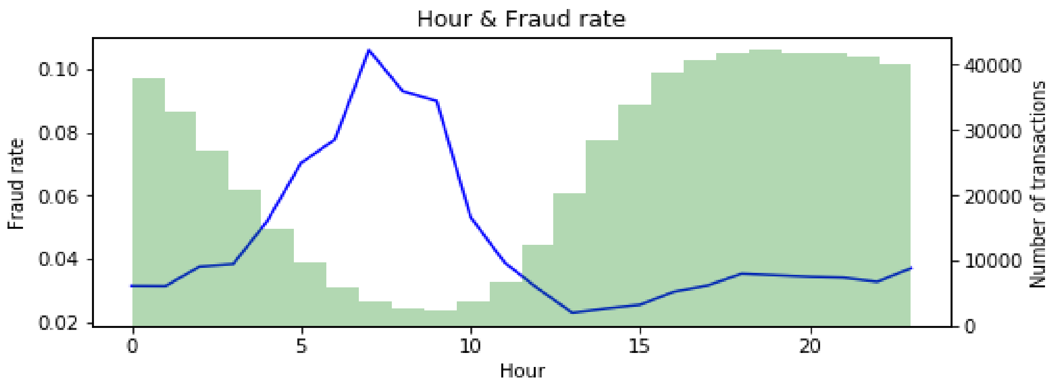


- ***4.Modeling***

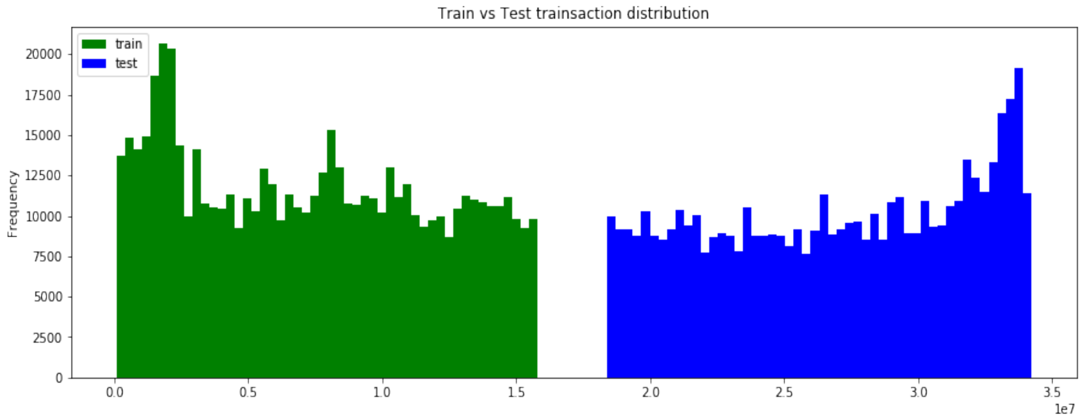
XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. This system gives us a good understanding of how the knowledge flow been created. Then loaded the training set into the model and use **XGBoost** to gain the predicting model. The lots of parameters to tune can improve the accuracy.

- ***5. Important findings:***

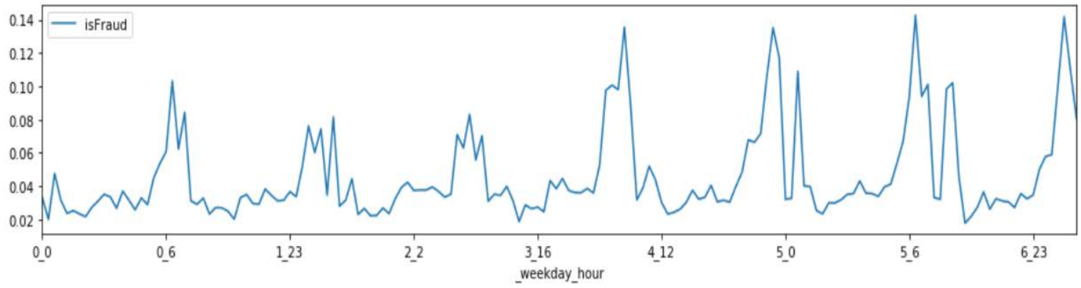
A . Exploring the impact of datetime variables on fraud. The **TransactionDT** feature is a time data from a given reference datetime. the data can use the fraud transaction rate by **weekday, time, and days.** In this figure (**Hour & Fraud rat**e), it shows the fraud ratio and transaction volume at different times, and the time from 5am to 10am is the trough period of normal trading, but it is the peak period of fraudulent transactions.



In the below figure (**Train vs Test trainsaction distribution**), for the distribution of transactions over time, it can be known that there is no intersection between the training set and the test set in same time.

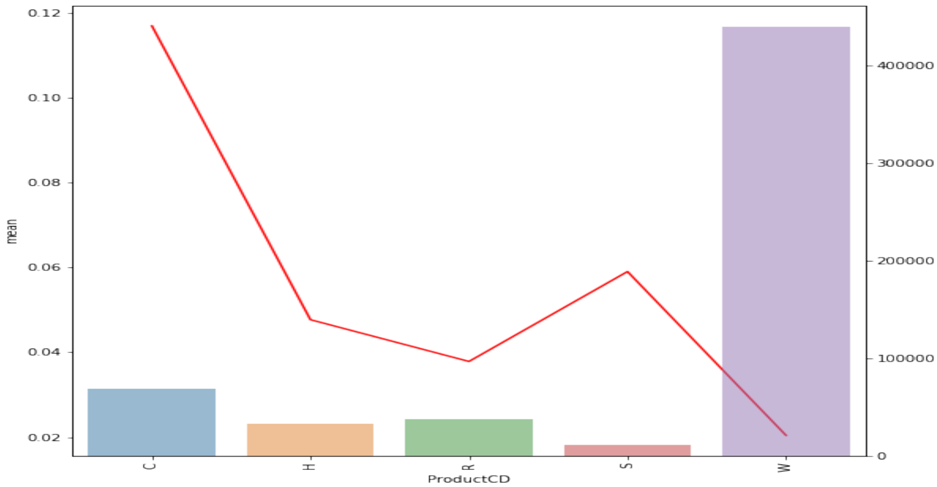


In the below figure, it was easy to find the evening and midnight of the end week (3,4,5,6) have high fraud transaction.

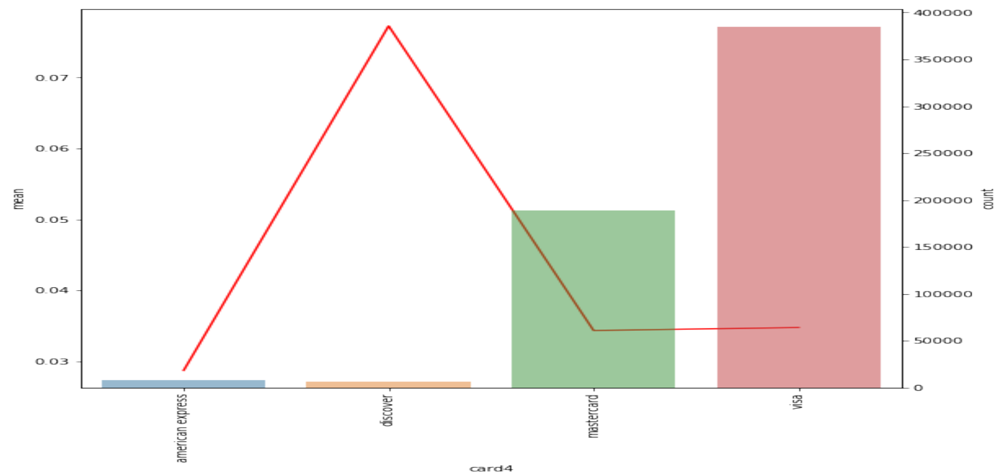


B. Exploring the impact of import variables on fraud.

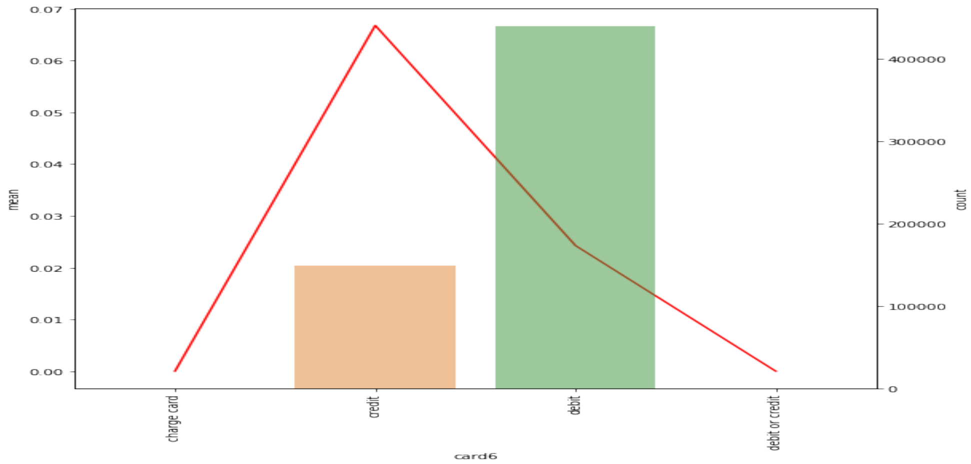
1. **ProductCD.** ProductCD W's transactions accounted for a relatively high proportion, but ProductCD C's fraud accounted for a relatively high proportion. In the case of ProductCD's C transactions, about 11.69% were fraudulent transactions, and ProductCD C accounted for 10.62% of the total transactions, but its fraud. Trading accounted for approximately 38.76% of total fraudulent transactions



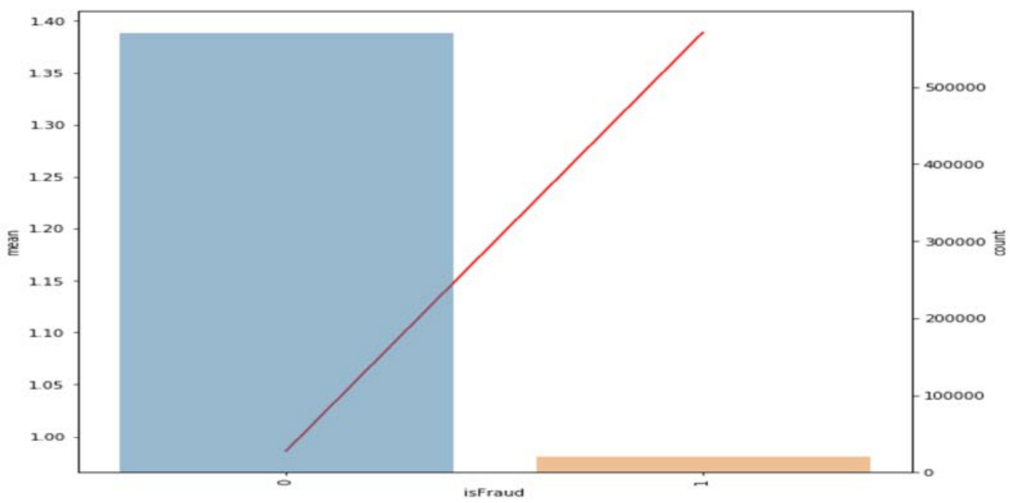
2. **the type of credit card(Card4).** In the below figure, VISA and master card have high count of fraud , especially VISA, but their fraud is not high, but the fraud is relatively high, ae is the lowest.



3. **the type of card(card6)**. In the below figure, **Credit card** fraud accounts for a higher debit card. The other two card products are under-utilized and the fraud ratio is 0.



4. **The price of each transaction.** By finding the average price of each traded product, the price per time is divided by the average price, and the ratio of each price to the average price is obtained, and then the summary operation is performed. As shown in the above figure, the price of the fraudulent transaction is relatively normal. The price is about 40% higher



2.2 *Completion: Developing and testing your machine learning system*

In this part, data cleaning, pre-processing, feature engineering and modelling process were designed and optimized to predict the fare amount in the testset.csv. In the beginning, **XGBoost** was selected and the performance of the initial design on leaderboard was only around 0.9312 (top 62%). After a lot of work in feature engineering, a new algorithm (**Light Gradient Boosting Machine**) was chosen the score was improved to 0.9372 (top 58%). However, the score was no longer improved after that, even attempted for a lot of times. Finally, the final score on the leader board is 0.9419 (top 48%).

- ***1. Discuss the initial design of your system, i.e. before you have submitted any predictions to the Kaggle competition. Justify each decision you made in its design, e.g. reference insight you gained in the Core part.***

Step 1 - Load Data (merge data): The data is broken into two files identity and transaction, which are joined by **TransactionID**. **The first need to merge, become two datasets**: **train_merge.csv and test_merge.csv.**

Step 2 - Initial Data Analysis: The data have huge data, Train_merge.csv contains 590540 rows of data, and test_merge.csv contains 506691 rows of data, the attributes includes 433 and 432. The data includes many missing values. So the ***Memory Reducer need to do.***

```
Mem. usage decreased to 648.22 Mb (66.8% reduction)
Mem. usage decreased to 563.43 Mb (66.3% reduction)
```

Attributes	mean
TransactionDT	timedetla at a certain point in time, not a specific timestamp.
TransactionAMT	US dollar settlement transaction amount
ProductCD	Product code for each transaction
Card1 - card6	Card information paid, it may be card products, issuing banks, issuing organizations,etc
Addr& Dist	Address& distance
P_ & R__ emaildomain	Purchase and receiving mailbox service providers
C1-C14	Some statistics, specifically unknown
D1-D15	It is said to be a timedelta with the previous transaction, but unknown
M1-M9 (except M4)	The value is T, F, the specific meaning is unknown, it is said that the matching result of some materials (name, address, etc.)
M4	M4: The value is M0, M1, M2. The specific meaning is unknown. It may be a delay.

Vxxx	The characteristics created by Vesta, may be customer rankings, etc., specifically unknown.

- **Step 3.1 - Data Pre-processing (Data Cleansing):** In this step, some instances with unreasonable values or outliers in certain features were removed. The rationality of the conditions was explored in later steps. However, after finalizing the conditions, it is memory-efficient to put this step here, especially for big data. (1. drop the index column,2. clean the outliers). The result is show that the attributes is **433, So After data cleansing, 82** attributes **were removed.**

```
one_value_cols = [col for col in train.columns if train[col].nunique() <= 1]
one_value_cols_test = [col for col in test.columns if test[col].nunique() <= 1]

many_null_cols = [col for col in train.columns if train[col].isnull().sum() / train.shape[0] > 0.9]
many_null_cols_test = [col for col in test.columns if test[col].isnull().sum() / test.shape[0] > 0.9]

big_top_value_cols = [col for col in train.columns if train[col].value_counts(dropna=False, normalize=True).values[0] > 0.9]
big_top_value_cols_test = [col for col in test.columns if test[col].value_counts(dropna=False, normalize=True).values[0] > 0.9]
```

- **Step 3.2 - Data Pre-processing (Categorical Feature Quantification):** There are columns of features in the raw data were categorical, which are showed in the below figure. In this step, the original data were replaced by number.

```
Categoric Columns: ['id_23', 'card4', 'M9', 'R_emaildomain', 'M8', 'id_37', 'M7', 'id_28', 'id_15', 'M1', 'ProductCD', 'M4', 'id_27', 'id_38', '_ymd', 'id_36', 'id_12', 'id_34', 'id_30', 'DeviceInfo', 'Date', 'id_29', 'id_33', 'id_35', 'DeviceType', 'P_emaildomain', 'card6', 'id_31', 'M3', '_year_month', 'id_16', 'M5', 'M6', 'M2']
```

```
df['ProductCD'] = df['ProductCD'].map({'W' :5, 'H' :4, 'C' :3, 'S' :2, 'R' :1})
df['card4'] = df['card4'].map({'discover' :4, 'mastercard' :3, 'visa' :2, 'american express' :1})
df['card6'] = df['card6'].map({'credit' :4, 'debit' :3, 'debit or credit' :2, 'charge card' :1})
df['ProductCD'] = df['ProductCD'].map({'credit' :4, 'debit' :3, 'debit or credit' :2, 'charge card' :1})

df['id_34'] = df['id_34'].map({ 'F' :2, 'T' :1})
df['id_35'] = df['id_35'].map({ 'F' :2, 'T' :1})
df['id_36'] = df['id_36'].map({ 'F' :2, 'T' :1})
df['id_37'] = df['id_37'].map({ 'F' :2, 'T' :1})
df['id_38'] = df['id_38'].map({ 'F' :2, 'T' :1})
df['M1'] = df['M1'].map({ 'F' :2, 'T' :1})
df['M2'] = df['M2'].map({ 'F' :2, 'T' :1})
df['M3'] = df['M3'].map({ 'F' :2, 'T' :1})
df['M5'] = df['M5'].map({ 'F' :2, 'T' :1})
df['M6'] = df['M6'].map({ 'F' :2, 'T' :1})
df['M7'] = df['M7'].map({ 'F' :2, 'T' :1})
df['M8'] = df['M8'].map({ 'F' :2, 'T' :1})
df['M9'] = df['M9'].map({ 'F' :2, 'T' :1})
df['M4'] = df['M4'].map({ 'MO' :1, 'M1' :2, 'M2' :3})
df['id_12'] = df['id_12'].map({ 'NotFound' :2, 'Found' :1})
df['id_16'] = df['id_16'].map({ 'NotFound' :2, 'Found' :1})
df['id_29'] = df['id_29'].map({ 'NotFound' :2, 'Found' :1})
df['id_15'] = df['id_15'].map({ 'Unknown' :3 ,'New' :2, 'Found' :1})
df['id_28'] = df['id_28'].map({'New' :2, 'Found' :1})
```

- **Step 3.3 - Data Pre-processing (Data Standardization): T**he train and test were standardized by the mean and STD values. After data cleansing and standardization, the distributions are plotted below. It can be found clearly that the outliers were removed by comparing the plots below and the plots above.

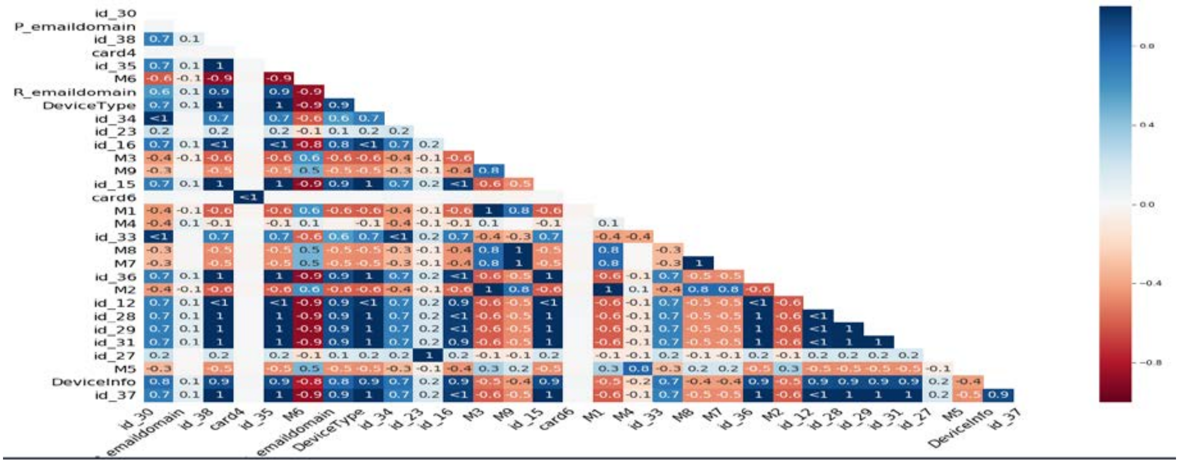
```
train['id_02_to_mean_card1'] = train['id_02'] / train.groupby(['card1'])['id_02'].transform('mean')
train['id_02_to_mean_card4'] = train['id_02'] / train.groupby(['card4'])['id_02'].transform('mean')
train['id_02_to_std_card1'] = train['id_02'] / train.groupby(['card1'])['id_02'].transform('std')
train['id_02_to_std_card4'] = train['id_02'] / train.groupby(['card4'])['id_02'].transform('std')

test['id_02_to_mean_card1'] = test['id_02'] / test.groupby(['card1'])['id_02'].transform('mean')
test['id_02_to_mean_card4'] = test['id_02'] / test.groupby(['card4'])['id_02'].transform('mean')
test['id_02_to_std_card1'] = test['id_02'] / test.groupby(['card1'])['id_02'].transform('std')
test['id_02_to_std_card4'] = test['id_02'] / test.groupby(['card4'])['id_02'].transform('std')

train['D15_to_mean_card1'] = train['D15'] / train.groupby(['card1'])['D15'].transform('mean')
train['D15_to_mean_card4'] = train['D15'] / train.groupby(['card4'])['D15'].transform('mean')
train['D15_to_std_card1'] = train['D15'] / train.groupby(['card1'])['D15'].transform('std')
train['D15_to_std_card4'] = train['D15'] / train.groupby(['card4'])['D15'].transform('std')

test['D15_to_mean_card1'] = test['D15'] / test.groupby(['card1'])['D15'].transform('mean')
test['D15_to_mean_card4'] = test['D15'] / test.groupby(['card4'])['D15'].transform('mean')
test['D15_to_std_card1'] = test['D15'] / test.groupby(['card1'])['D15'].transform('std')
test['D15_to_std_card4'] = test['D15'] / test.groupby(['card4'])['D15'].transform('std')
```

- **Step 4 - Exploratory Data Analysis:** The correlation between the features was explored. From the result, we can see many features have a strong correlation with the isFraud and the result of correlation between the isFraud and other columns is shown below:
- Using missingno.heatmap(). The heatmap tells the correlation between the missing nature of the data. Like the below picture: **Id_38 and id_35, DeveiceType** have a same relationship in the correlation map. **M7 and M8** have a same relationship in the correlation map. **Id_28, id_29 and id_31** have a same relationship in the correlation map.



- **Step 5 - Modelling:**
- **XGBoost** is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. This system gives us a good understanding of how the knowledge flow been created. Then loaded the training set into the model and use **XGBoost** to gain the predicting model.

```
clf = xgb.XGBClassifier(n_estimators=500,
                        n_jobs=4,
                        max_depth=9,
                        learning_rate=0.05,
                        subsample=0.9,
                        colsample_bytree=0.9,
                        missing=-999)
```

paramers	mean	purpose
n_estimators	Maximum number of decision trees; high value can lead to overfitting	Parameters for controlling speed
n_jobs		Important parameters which control overfitting
max_depth	The maximum depth of a tree, same as GBM.	Important parameters which control overfitting
learning_rate	Optimal values lie between 0.01-0.2	Important parameters which control overfitting
subsample	Subsample ratio of the training instance	Parameters for controlling speed
colsample_bytree	Subsample ratio of columns	Parameters for controlling speed
missing	Missing value	

submission.csv 0.9312

a month ago by Aaron

Splitting to train and validation We will now split the train dataset into train and validation set. We will keep 20% of data for validation.

- ***2. Discuss the design of one or more of your intermediary systems. Justify the changes you made to the previous design based on its performance on the leaderboard, and from any other additional investigation you performed.***

  Reply: **Feature Engineering**

1. ***Add New Features:The time is very important, so add _hour, _weekday, _day. T***he **TransactionDT** feature is a time data from a given reference datetime **.*So we can gave new attributes include weekday, hour, day.***

```
train['Date'] = train['TransactionDT'].apply(lambda x: (startdate + datetime.timedelta(seconds=x)))
train['_ymd'] = train['Date'].dt.year.astype(str) + '-' + train['Date'].dt.month.astype(str) + '-' + train['Date'].dt.day.astype(str)
train['_year_month'] = train['Date'].dt.year.astype(str) + '-' + train['Date'].dt.month.astype(str)
train['_weekday'] = train['Date'].dt.dayofweek
train['_hour'] = train['Date'].dt.hour
train['_day'] = train['Date'].dt.day
```

2. **Handle Categorical Feature Quantification:**

A. Device Info: many information include the same mean, Like 'SM','SAMSUNG' all mean the company of Samsung. So the data information will provide the noise and the data missing mean, we define the function is setDevcie().

```
def setDevice(df):
    df['DeviceInfo'] = df['DeviceInfo'].fillna('unknown_device').str.lower()
    df['device_name'] = df['DeviceInfo'].str.split('/', expand=True)[0]

    df.loc[df['device_name'].str.contains('SM', na=False), 'device_name'] = 'Samsung'
    df.loc[df['device_name'].str.contains('SAMSUNG', na=False), 'device_name'] = 'Samsung'
    df.loc[df['device_name'].str.contains('GT-', na=False), 'device_name'] = 'Samsung'
    df.loc[df['device_name'].str.contains('Moto G', na=False), 'device_name'] = 'Motorola'
    df.loc[df['device_name'].str.contains('Moto', na=False), 'device_name'] = 'Motorola'
    df.loc[df['device_name'].str.contains('moto', na=False), 'device_name'] = 'Motorola'
```

B. Handle P Email Domain and R Email Domain. The proportion of fraud in the two mailboxes is similar, and the fraud of proton mail is relatively high, especially R_**emaildomain**, which accounts for nearly 100% of fraud. So we change the two email domain like the below figure.

```
for c in ['P_emaildomain', 'R_emaildomain']:
    train[c + '_bin'] = train[c].map(emails)
    test[c + '_bin'] = test[c].map(emails)

    train[c + '_suffix'] = train[c].map(lambda x: str(x).split('.')[-1])
    test[c + '_suffix'] = test[c].map(lambda x: str(x).split('.')[-1])

    train[c + '_suffix'] = train[c + '_suffix'].map(lambda x: x if str(x) not in us_emails else 'us')
    test[c + '_suffix'] = test[c + '_suffix'].map(lambda x: x if str(x) not in us_emails else 'us')
```

C. Handle Browser Version

the proportion of fraud in many browser versions, So, we can relation with id_31 and set the browser version all at 1 , like the below figure.

```
train["lastest_browser"] = np.zeros(train.shape[0])
test["lastest_browser"] = np.zeros(test.shape[0])

def setBrowser(df):
    df.loc[df["id_31"]=="samsung browser 7.0",'lastest_browser']=1
    df.loc[df["id_31"]=="opera 53.0",'lastest_browser']=1
    df.loc[df["id_31"]=="mobile safari 10.0",'lastest_browser']=1
    df.loc[df["id_31"]=="google search application 49.0",'lastest_browser']=1
    df.loc[df["id_31"]=="firefox 60.0",'lastest_browser']=1
    df.loc[df["id_31"]=="edge 17.0",'lastest_browser']=1
    df.loc[df["id_31"]=="chrome 69.0",'lastest_browser']=1
    df.loc[df["id_31"]=="chrome 67.0 for android",'lastest_browser']=1
    df.loc[df["id_31"]=="chrome 63.0 for android",'lastest_browser']=1
    df.loc[df["id_31"]=="chrome 63.0 for ios",'lastest_browser']=1
    df.loc[df["id_31"]=="chrome 64.0",'lastest_browser']=1
```

D. the other Categorical Feature: Like ProductCD, card4, card6, id_34,…,we can find the define the different value.

```
def ChangeToNum(df):
    df['ProductCD'] = df['ProductCD'].map({'W' :5, 'H' :4, 'C' :3, 'S' :2, 'R' :1})
    df['card4'] = df['card4'].map({'discover' :4, 'mastercard' :3, 'visa' :2, 'american express' :1})
    df['card6'] = df['card6'].map({'credit' :4, 'debit' :3, 'debit or credit' :2, 'charge card' :1})
    df['ProductCD'] = df['ProductCD'].map({'credit' :4, 'debit' :3, 'debit or credit' :2, 'charge card' :1})

    df['id_34'] = df['id_34'].map({ 'F' :2, 'T' :1})
    df['id_35'] = df['id_35'].map({ 'F' :2, 'T' :1})
    df['id_36'] = df['id_36'].map({ 'F' :2, 'T' :1})
    df['id_37'] = df['id_37'].map({ 'F' :2, 'T' :1})
    df['id_38'] = df['id_38'].map({ 'F' :2, 'T' :1})
    df['M1'] = df['M1'].map({ 'F' :2, 'T' :1})
    df['M2'] = df['M2'].map({ 'F' :2, 'T' :1})
    df['M3'] = df['M3'].map({ 'F' :2, 'T' :1})
    df['M5'] = df['M5'].map({ 'F' :2, 'T' :1})
    df['M6'] = df['M6'].map({ 'F' :2, 'T' :1})
    df['M7'] = df['M7'].map({ 'F' :2, 'T' :1})
    df['M8'] = df['M8'].map({ 'F' :2, 'T' :1})
    df['M9'] = df['M9'].map({ 'F' :2, 'T' :1})
    df['M4'] = df['M4'].map({ 'M0' :1, 'M1' :2, 'M2' :3})
    df['id_12'] = df['id_12'].map({ 'NotFound' :2, 'Found' :1})
    df['id_16'] = df['id_16'].map({ 'NotFound' :2, 'Found' :1})
    df['id_29'] = df['id_29'].map({ 'NotFound' :2, 'Found' :1})
    df['id_15'] = df['id_15'].map({ 'Unknown' :3 ,'New' :2, 'Found' :1})
    df['id_28'] = df['id_28'].map({'New' :2, 'Found' :1})
```

3. Data Preprocessing

Delete the many missing value and stand the useful value. Firstly, clearing too many null attributes, the null rate >0.9; Then, clearing the many repeated attributes, like the rate >0.9. This is delete the many not mean data.

```
def get_too_many_null_attr(data):
    many_null_cols = [col for col in data.columns if data[col].isnull().sum() / data.shape
[0] > 0.9]
    return many_null_cols

def get_too_many_repeated_val(data):
    big_top_value_cols = [col for col in train.columns if train[col].value_counts(dropna=Fal
se, normalize=True).values[0] > 0.9]
    return big_top_value_cols

def get_useless_columns(data):
    too_many_null = get_too_many_null_attr(data)
    print("More than 90% null: " + str(len(too_many_null)))
    too_many_repeated = get_too_many_repeated_val(data)
    print("More than 90% repeated value: " + str(len(too_many_repeated)))
    cols_to_drop = list(set(too_many_null + too_many_repeated))
    #cols_to_drop.remove('isFraud')
    return cols_to_drop
```

Label encoding the each column, is import the LabelEncoder class from the sklearn library, fit and transform the each column of the data, and then replace the existing text data with the new encoded data.

```
# Label Encoding
for f in train.columns:
    if train[f].dtype.name =='object' or test[f].dtype.name =='object':
        lbl = preprocessing.LabelEncoder()
        lbl.fit(list(train[f].values) + list(test[f].values))
        train[f] = lbl.transform(list(train[f].values))
        test[f] = lbl.transform(list(test[f].values))
```

4. Change a new model

Light GBM is a gradient boosting framework that uses tree-based learning algorithm. It focuses on accuracy of results. LGBM also supports GPU learning and thus data scientists are widely using LGBM for data science application development. the result will improve

```
params = {'num_leaves': 546,
          'min_child_weight': 0.03454472573214212,
          'feature_fraction': 0.1797454081646243,
          'bagging_fraction': 0.2181193142567742,
          'min_data_in_leaf': 106,
          'objective': 'binary',
          'max_depth': -1,
          'learning_rate': 0.005883242363721497,
          "boosting_type": "gbdt",
          "bagging_seed": 11,
          "metric": 'auc',
          "verbosity": -1,
          'reg_alpha': 0.3299927210061127,
          'reg_lambda': 0.3885237330340494,
          'random_state': 42,
}
```

The table present the import paramers to tune that can improve the accuracy.

paramers	mean	purpose
Num_leaves	**The number of leaves in a tree**	**Important parameters which control overfitting**
feature_fraction	**Fraction of features to be taken for each iteration.**	**Parameters for controlling speed**
bagging_fraction	**Data to be used for each iteration and is generally used to speed up the training and avoid overfitting**	**Parameters for controlling speed**
min_data_in_leaf	**Default=20, alias=min_data,min_child_samples**	**Important parameters which control overfitting**
max_depth	**Important to note that tree still grows leaf-wise. Hence it is important to tune.**	**Important parameters which control overfitting**

5. **K-Folds cross-validator**: Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often

called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model. In this case, such as k=5 becoming 5-fold cross-validation.

```
folds = TimeSeriesSplit(n_splits=5)

aucs = list()
feature_importances = pd.DataFrame()
feature_importances['feature'] = X.columns

training_start_time = time()
for fold, (trn_idx, test_idx) in enumerate(folds.split(X, y)):
    start_time = time()
    print('Training on fold {}'.format(fold + 1))

    trn_data = lgb.Dataset(X.iloc[trn_idx], label=y.iloc[trn_idx])
    val_data = lgb.Dataset(X.iloc[test_idx], label=y.iloc[test_idx])
    clf = lgb.train(params, trn_data, 10000, valid_sets=[trn_data, val_data], verbose_eval=1000, early_stopping_rounds=500)

    feature_importances['fold_{}'.format(fold + 1)] = clf.feature_importance()
    aucs.append(clf.best_score['valid_1']['auc'])
```

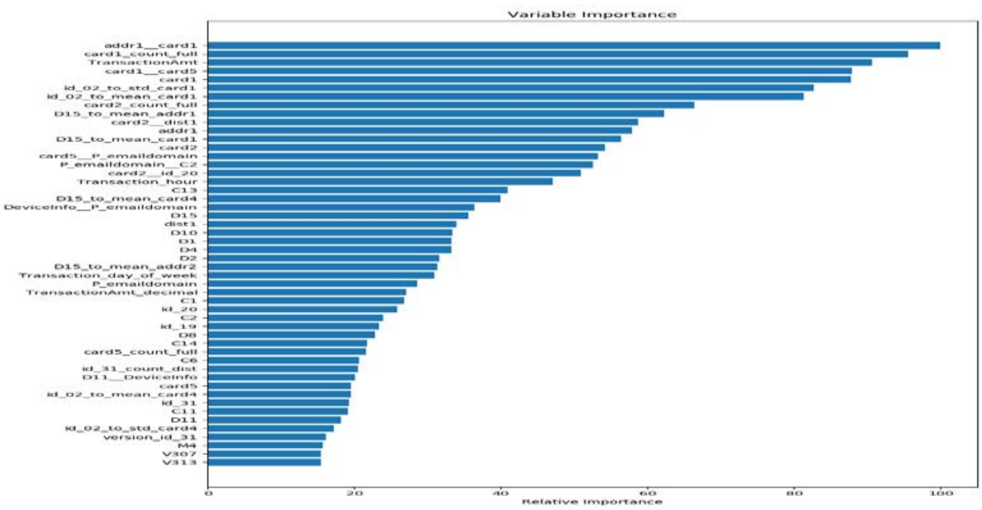
6. *Summit the csv file*

Finally, the important feature was created feature_important.csv file, and test_set_predict was transferred to DataFrame type and saved as .csv format for submission. The code like the below figure and the result showed the end figure.

```
feature_importances['average'] = feature_importances[['fold_{}'.format(fold + 1) for fold in range(folds.n_splits)]].mean(axis=1)
feature_importances.to_csv('feature_importances.csv')

plt.figure(figsize=(16, 16))
sns.barplot(data=feature_importances.sort_values(by='average', ascending=False).head(50), x='average', y='feature');
plt.title('50 TOP feature importance over {} folds average'.format(folds.n_splits));
best_iter = clf.best_iteration
clf = lgb.LGBMClassifier(**params, num_boost_round=best_iter)
clf.fit(X, y)

sample_submission['isFraud'] = clf.predict_proba(test)[:, 1]
sample_submission.to_csv('submission_1909.csv', index=False)
```



submission_1909.csv 0.9372

13 days ago by Aaron

a new one for the data propose

The next system is only change the Configuration ***for the model, and the result was a little different, the score was improved by 0.004.***

```
params = {'num_leaves': 491,
          'min_child_weight': 0.03454472573214212,
          'feature_fraction': 0.3797454081646243,
          'bagging_fraction': 0.4181193142567742,
          'min_data_in_leaf': 106,
          'objective': 'binary',
          'max_depth': -1,
          'learning_rate': 0.006883242363721497,
          "boosting_type": "gbdt",
          "bagging_seed": 11,
          "metric": 'auc',
          "verbosity": -1,
          'reg_alpha': 0.3899927210061127,
          'reg_lambda': 0.6485237330340494,
          'random_state': 47
         }
```

sample_submission3009.csv
5 hours ago by Aaron
a new one

0.9419

- ***3. Use your judgement to choose the best system you have developed — this may not necessarily be the most accurate system on the leaderboard.***

Reply : The Kaggle website is **https://www.kaggle.com/aaron207/kernel210cac8a76**. And found the interesting website about the different of the popular model: https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db.

Function	XGBoost	Light GBM
Important parameters which control overfitting	1. **learning_rate or eta** – optimal values lie between 0.01-0.2 2. **max_depth** 3. **min_child_weight:** similar to min_child leaf; default is 1	1. **learning_rate** 2. **max_depth**: default is 20. Important to note that tree still grows leaf-wise. Hence it is important to tune **num_leaves** (number of leaves in a tree) which should be smaller than 2^(max_depth). It is a very important parameter for LGBM 3. **min_data_in_leaf**: default=20, alias= min_data, min_child_samples
Parameters for categorical values	Not Available	1. **categorical_feature:** specify the categorical features we want to use for training our model
Parameters for controlling speed	1. **colsample_bytree:** subsample ratio of columns 2. **subsample:** subsample ratio of the training instance 3. **n_estimators:** maximum number of decision trees; high value can lead to overfitting	1. **feature_fraction:** fraction of features to be taken for each iteration 2. **bagging_fraction:** data to be used for each iteration and is generally used to speed up the training and avoid overfitting 3. **num_iterations:** number of boosting iterations to be performed; default=100

- The different model provided the different result, an important thing to note here is that it performed poorly in terms of both speed and accuracy when XGBoot is used. the system is used by Light GBM. Besides,
- The processing of system includes the different attributes to compare. The figures were present the relation with each feature, built new columns to find the feature interaction, like the year, week_day,_hour.
- The septs of Data Standardization to reduce the impact of missing values and clearing the outliers data.
- To tune the parameters of Light GBM model, The result got the Mean AUC has the highest score, and Mean AUC is 0.9253039274404873. and the total time is below one hour. The speed of running system was improved fast.

- ***3. Challenge: Reflecting on your findings***

How was the team formed? What benefits were bought into the team by having different approaches? What were the major differences before and after the team formation to the model?

The team function on Kaggle allows us data miners from all over the world to discuss and develop the best solution together. Many approaches from different members in the team can be amalgamated. Team members can share their ideas, methods, algorithms and code; also, they can share their hardware resources. For example, if anyone in the team has spare GPUs resources or Cloud Machine Learning Engine for training the model, the team can share the cost together.

As a team leader in Kaggle, I can see all the submission history from any members in the team by merging with their submission history. We were made a meeting for discuss every three days, and we can find some important information for improve the score by sent email to each other. After that, I can change the code on the result for team’s discussion. The “My Submission” page became like a list of submission history from all the members in the team. The scores for each submission can be seen and compared. I can select up to 2 submissions to be used to count towards the final leaderboard score.

Cjfbfmh and AMY61 are the other member in my team. After more than 2 weeks struggling with the ***LGBM*** algorithm, and data analyze. They provide nice useful features to find, and the many parameters designed in the initial model (**XGBoost). In the nearly the half month to finfish the competition,** he ***LGBM*** algorithm was be designed, and used in the new system. After using the LGBM algorithm, I found it was worth trying it. Then I combined and modified the code from Cjfbfmh, to make it fit in my data processing pipeline with my original data reading, visualization, cleansing, feature manipulation part of code. And the system after combination worked well. Finally, after this team work, I got 0.01 improvement in Kaggle score, Cjfbfmh got 0.003 improvement.
Our final Kaggle score is 0.9419, which ranks 2991th out of 6303 contestants. (Top 48%).