Assignment 2: Practical Machine Learning Project

Project Title: Santander Customer Transaction Prediction

Student Name: Yang LIU, Yuhao SHI

Student ID: 13174420, 13338239

1. Introduction

1.1 Business Understanding:

The mission in Santander bank is to help people and businesses thrive. Santander bank provided a challenge which is predicting whether customers will generate transaction in the future. In this way, Santander not only can be provided services that their customers might need but also help the company make profits.

1.2 Kaggle Competition:

The Santander transaction competition on kaggle was the most intense competition of the past few years, for more than one month, this competition has nearly 9,000 data science teams from all over the world participated and submitted effective results.
In this competition, the task of the organizer of Santander was to use anonymous data to predict which customers will conduct specific transactions in the future, thus helping Santander to find methods to help customers could help them achieve their monetary goals. Although this competition is a binary problem with serious unbalance data, due to the data privacy protection consideration of Santander, the released competition data sets are all desensitized data after anonymization that means the features provided was masked with string. Therefore, many ideas for feature engineering based on prior knowledge are not available in this competition, which has become the biggest challenge in this game. The good thing is that all the features were numerical.

1.3 Objective of the Project:

In this project, we want to use the methods which are based on machine learning to predict the Santander customer whether conduct transaction in the future. Specifically, we processed the data which includes feature engineer and handled the unbalanced data. From the algorithms level, we tuned and compared many algorithms, including Logistic Regression/Random Forest/Light GMB. Compared with other algorithms, LGBM can get the best performance. In addition to comparing different kinds of algorithms, we also mentioned some areas of the current improvements and future works. In terms of ethics, we discussed the bank using data can be cause the impact on the client privacy.

1.4 Measure of Success:

In this project, we need to predict the target which is the customers of Santander whether conduct transaction in the future. The accuracy of the predict result will be the main factor in measuring the success of this project. According to the average performance of the teams on kaggle, we think that if our prediction accuracy is higher than 0.9, the project is successful.

1.5 Code Link :

https://github.com/13174420yang/UTS_ML2019_ID13174420

2. Data Overview and Exploration

2.1 Missing Value:

Figure 1: The percentage of missing values in each attribute

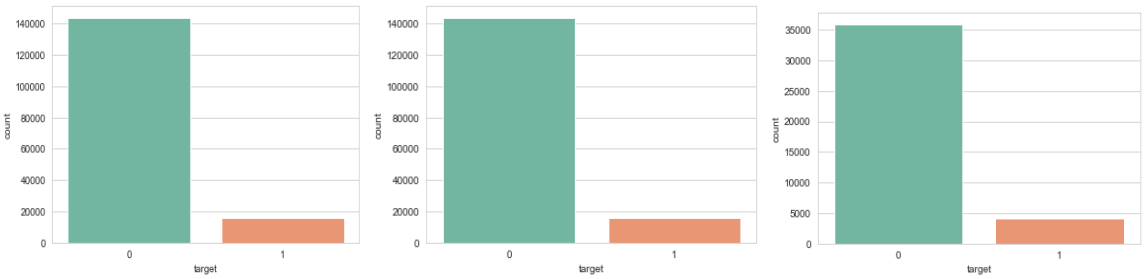
```
Out[3]:
         ID_code  target  var_0  var_1  ...  var_96  var_97  var_98  var_99
total        0.0     NaN    0.0    0.0  ...     0.0     0.0     0.0     0.0
percent      0.0     0.0    0.0    0.0  ...     0.0     0.0     0.0     0.0

[2 rows x 202 columns]
```

From the figure 1, we can see the output of Python code that there are no missing values in each attribute, so we do not need to consider the handling of the missing value.

2.2 Unbalanced Data:

Figure 2: The distribution of target values in whole dataset (a), training dataset (b) and validation dataset (c).

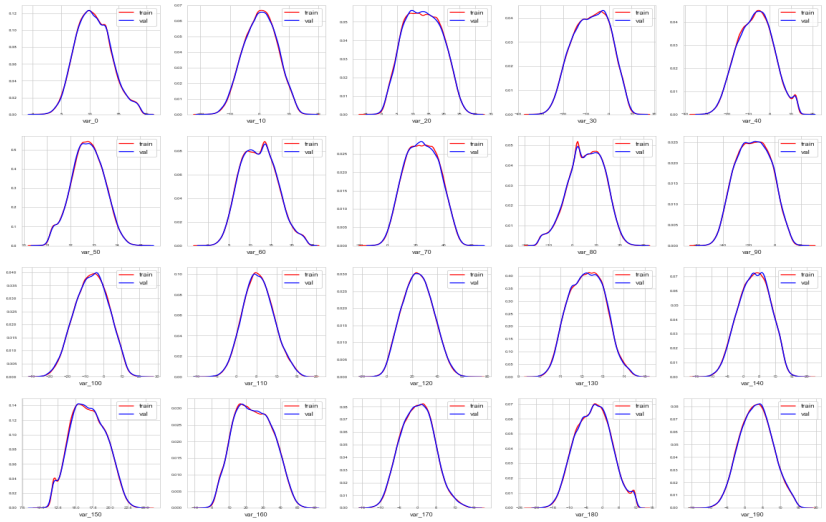
a) *b)* *c)*



From the three figure 2, we realize that there is a huge unbalanced in the target values. Although the values of the target column in training data is binary, the percentage of 0 value and 1 value is nearly 9:1, which performs the same in both validation data and the whole data set. In the feature engineering part, we need to solve the unbalanced issue to improve our models performance.

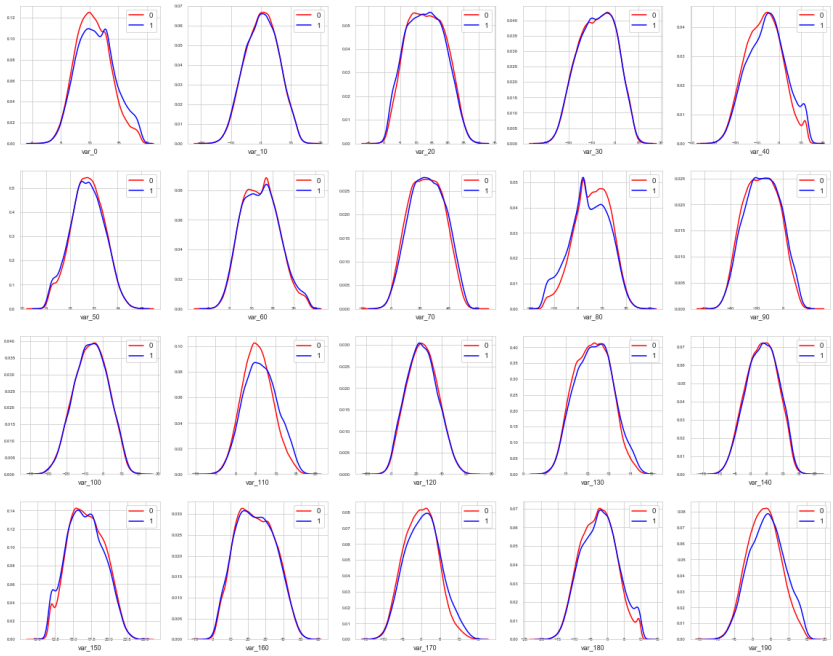
2.3 The Distribution of Different Attribute:

Figure 3: The distribution of training data and validation data in each attributes.



We selected some examples from the dataset which can be seen from the figure 3. The blue line and red line nearly overlapping that means each attributes almost have the same distribution in training dataset and validation dataset which seems to be well balanced. Training datasets and validation datasets are nearly consistency.

Figure 4: The proportion of positive and negative set in different attributes



The performance with different labs may have different behavior. In order to analysis different behaviors, we select 20 attributes to compare the proportion of positive sample (value=1) and negative sample (value=0) and the results have been shown in figure 4. Take the var_70 and var_140 for instance, the distribution of positive sample and negative sample almost same. On the other hand, var_80 has different distributions for the two target values'. we will take these finds into consideration in the future section, such as selection of the features for the prediction model.

Figure 5: The distribution of max values per rows and columns

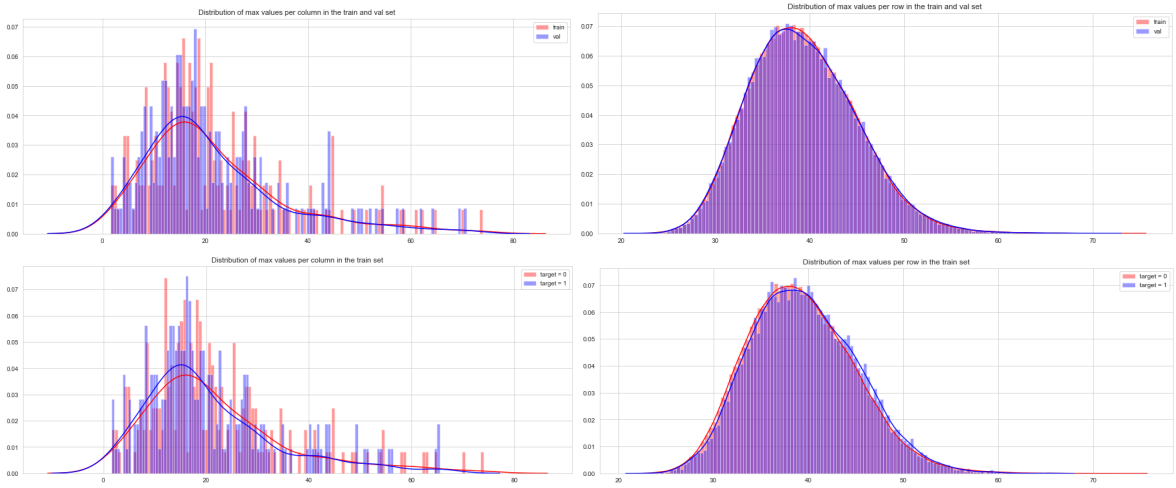


Figure 6: The distribution of min values per rows and columns

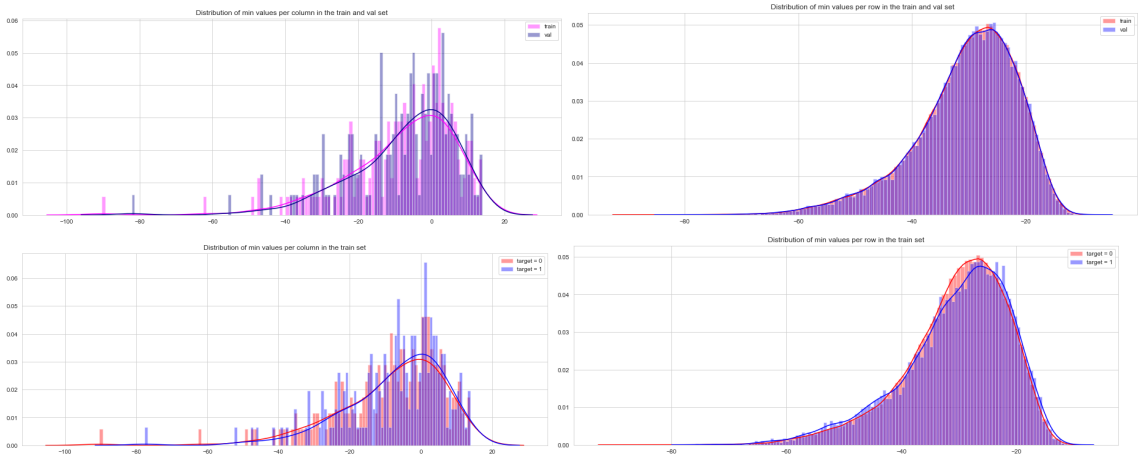


Figure 7: The distribution of mean values per rows and columns

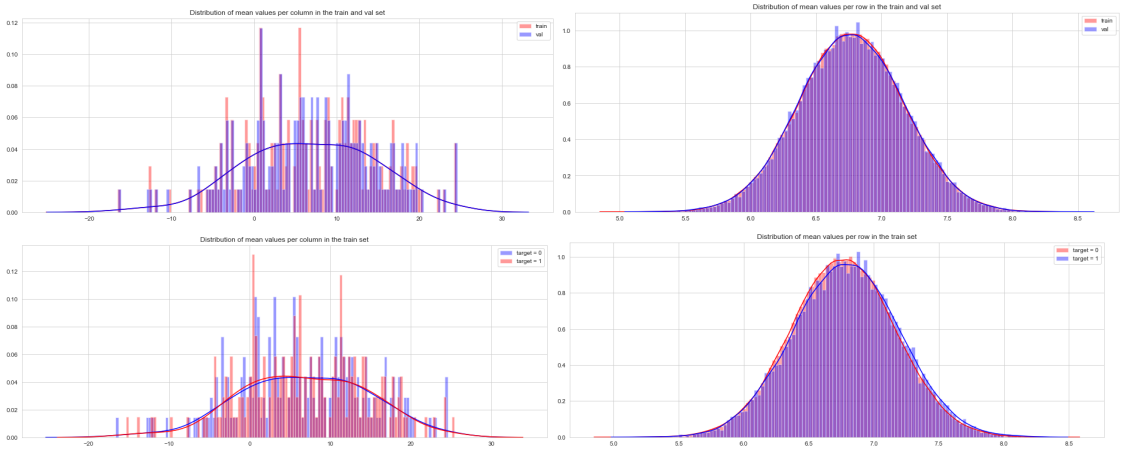


Figure 8: The distribution of std values per rows and columns

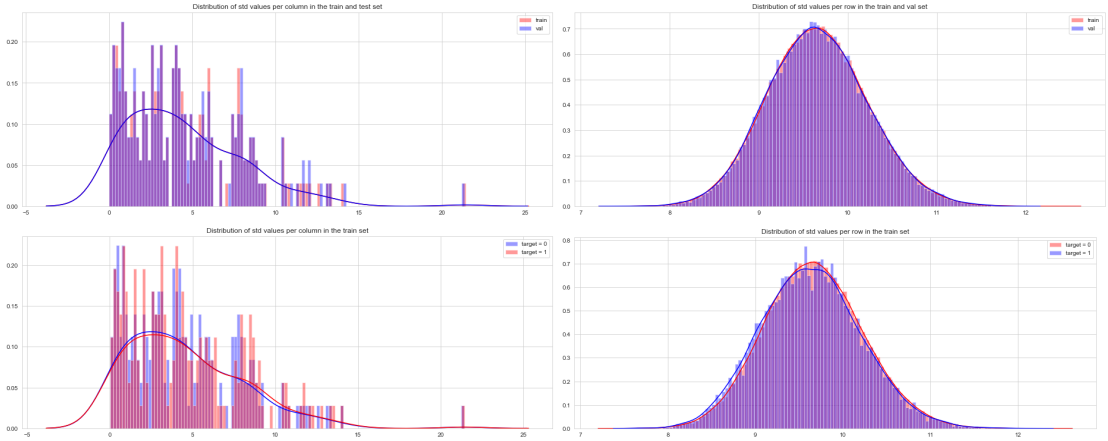


Figure 9: The distribution of skew values per rows and columns

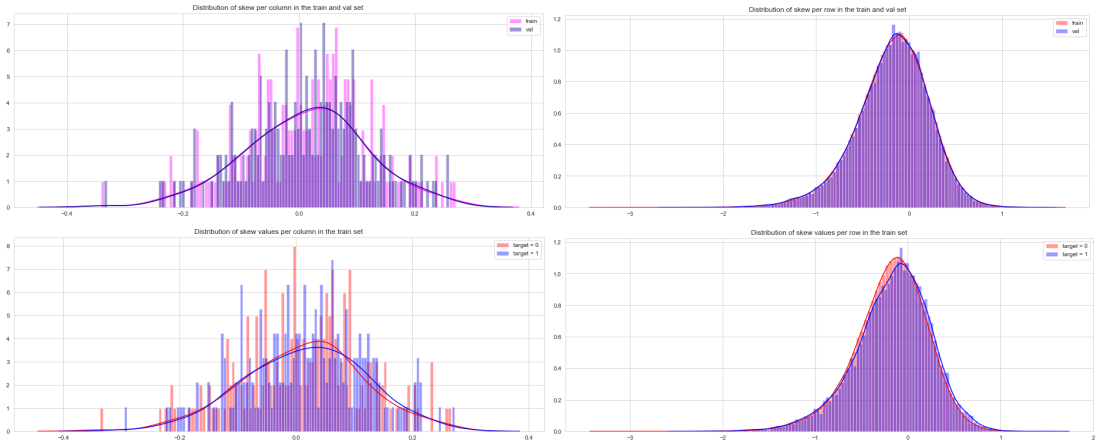
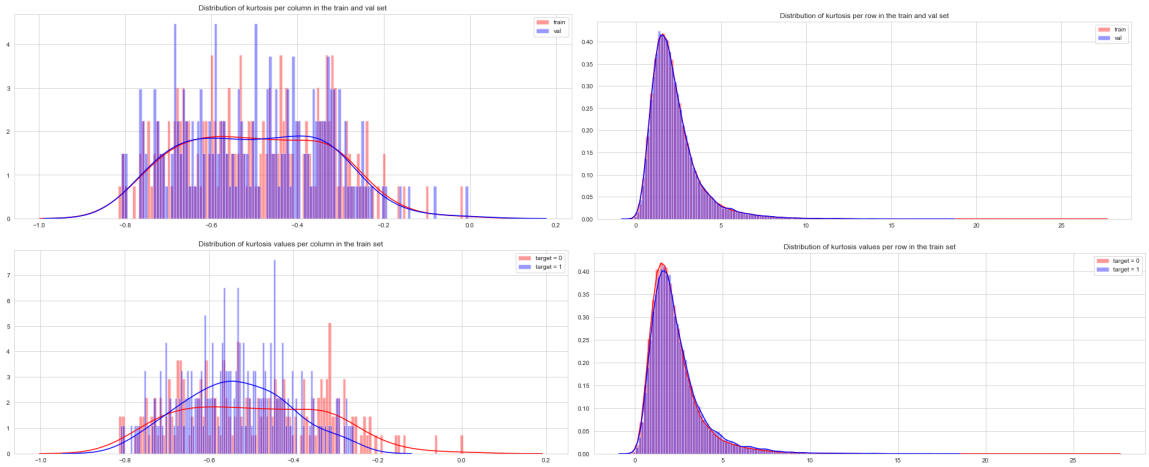


Figure 10: The distribution of kurtosis values in rows and columns



From figure 5-10, We draw different histograms showing the distributions of different mathematical statics values, such as maximum, minimum, mean, standard deviation, skew and kurtosis, in order to gain the insight view of the large dataset. We can see from those pictures that there is a similar trend among them. Firstly, comparing them from the vertical view, whether it is training dataset or validation set, or the target is 0 or 1, the distribution of data on columns is almost different but the distributions on rows are nearly coincide, we believe that the coincided distribution is with less help to our prediction model, we decide to add those statistic values on columns as the new attributes in the feature engineering part. Secondly, although the distribution of columns and rows are different, all the trend of the distributions conform to the characteristics of a normal distribution.

2.4 Feature Correlation:

Figure 11: The top 5 and last 5 of the correlation between each attribute

```
In [81]: correlations.head(5)
Out[81]:
   level_0  level_1            0
0   var_75  var_191  2.703975e-08
1  var_191   var_75  2.703975e-08
2  var_173    var_6  5.942735e-08
3    var_6  var_173  5.942735e-08
4  var_126  var_109  1.313947e-07
```

```
In [80]: correlations.tail(5)
Out[80]:
       level_0  level_1         0
39795  var_165   var_81  0.009714
39796   var_53  var_148  0.009788
39797  var_148   var_53  0.009788
39798   var_26  var_139  0.009844
39799  var_139   var_26  0.009844
```

From figure 11, it can be seen that we calculate the correlations between each attribute, and we know that the larger the absolute value of correlation is, the more relevant the two attributes are, also we can replace the highly relevant attributes. We can see from the two tables that the largest value is nearly 9.07 and the smallest value is -0.009844, so all the attributes are less relevant.

2.5 Duplicate Values:

Figure 12: The top five number of duplicate values in attributes

```
Out[74]:
                    68        108       126       12        91
Feature         var_68   var_108   var_126   var_12   var_91
Max duplicates    1084       313       305       203       66
Value           5.0214   14.1999   11.5356   13.5545   6.9785
```

We recognize that some attributes have some duplicate values, if the number of duplicate values is large, the role of such attributes is quite less in the prediction model. From the figure 12, we can see that the maximum number of duplicates is 1084, but the data set has 200000 value, the percentage is small and the impact of those duplicates is not so obvious, thus, we decided not to delete those values in the preprocessing.

3. Data Preprocessing

https://github.com/13174420yang/UTS_ML2019_ID13174420/blob/master/assignment2/006_data_preprocessing_directly_version2.ipynb

3.1 Feature Engineering:

After the data overview processment, we believe that some features like maximum, minimum can be added into the dataset as the new attributes, so that it can help us improve the performance of the prediction model, the added new features shown in the table below.

Table 1: The Description of New Features

New Feature	**Description**
var_sum	sum on column
var_min	min value on column
var_max	max value on column
var_mean	mean value on column
var_std	std value on column
var_skew	skew value on column
var_kurt	kurt value on column
var_med	med value on column
var_round_1	one decimal place every column
var_round_2	two decimal places every column

The code can be shown here:

```
# add some basic columns
def basic_preprocessing(df):
    columns = df.columns
    df['sum'] = df[columns].sum(axis=1)
    df['min'] = df[columns].min(axis=1)
    df['max'] = df[columns].max(axis=1)
    df['mean'] = df[columns].mean(axis=1)
    df['std'] = df[columns].std(axis=1)
    df['skew'] = df[columns].skew(axis=1)
    df['kurt'] = df[columns].kurtosis(axis=1)
    df['med'] = df[columns].median(axis=1)

    # add round features
    for column in columns:
        df['r1_' + column] = np.round(df[column], 1)
        df['r2_' + column] = np.round(df[column], 2)
    return df
```

```
df_data_prep = df_data.copy()
df_data_prep = basic_preprocessing(df_data_prep)
print(df_data_prep.shape)
df_data_prep.head()
```

```
(200000, 608)
```

3.2 PCA:

As there are as many as 608 features, it would be favorable to reduce the dimension to handle curse of dimensionality and speed up the computing process. In this section we apply PCA to reduce the dimension to 300, 200, 150, 100 and 80. However the initial experiment shows poor performance compared with dataset with all features. Thus in the next sections we will still use the dataset without PCA.

3.3 Split Train and Validation Dataset:

The real test set does not contain labels that we can only get the result via submission. For simplicity, we split the train dataset as training set and validation set and evaluate the performance of our algorithms on validation set. Specifically, we use the last 20% data as validation set. Following table shows the basic statistics of these two datasets

Table 2: The Basic Statistics of Training Dataset and Validation Dataset

Name	Size	1s	0s
Train set	160000	16049	143951
Validation set	40000	4049	35951

We can see that the proportion of 1s and 0s are similar thus it's suitable to train on train set while evaluating the performance on validation set. There can be other methods to split the dataset such as building validation set via sampling or we can use K-fold cross validation,

which can be used in the future work. In the next sections we will only modify train set while keeping validation set unchanged.
The code of this part can be shown here:

20% validation data

```
def split_train_val(df, train_path=None, val_path=None):
    df_train, df_val = df[:160000], df[160000:]
    print(df_train.shape, df_val.shape)
    df_train.to_csv(train_path)
    df_val.to_csv(val_path)
    return df_train, df_val
```

```
train_path = "train_data.csv"
val_path = "val_data.csv"
train_data, val_data = split_train_val(df_data_prep, train_path, val_path)
train_path = "train_label.csv"
val_path = "val_label.csv"
train_label, val_label = split_train_val(df_label, train_path, val_path)
```

```
(160000, 608) (40000, 608)
(160000, 1) (40000, 1)
```

3.4 Handle Unbalanced Samples:

As we shown before, the dataset is unbalanced that there are about only 10% of the samples with target 1, meaning that only a small proportion of users will perform transaction finally. In order to make the classifier easier to split the two classes, in this section we will balance the train dataset. Generally, there can be three ways to handle unbalanced samples: under sampling, oversampling and the combination of under sampling and oversampling. Under sampling is to just sample a subset from the larger classes and then combine it with the smaller classes. This is simple to achieve, however, as the size of dataset decreases, some crucial information may be lost, which may be harmful for the performance. In contrast, oversampling tries to sample some data repeatedly from the smaller classes, thus the size of resulted smaller classes will be increased. Instead of sampling, another oversampling method maybe SMOTE, which generates new data based on nearest neighbor. Although SMOTE is suitable for smaller dataset with low dimensions, in our project we need to handle a large dataset with 160000 samples and 600+ dimensions, which makes it time consuming to find the neighbors. In our project, we combine the method of under sampling and oversampling. In terms of under sampling, we sample 75% data from 0s. In terms of oversampling, we sample repeatedly to increase 1s from 16049 to 64196. After combining the two sets together and shuffling, we obtain a dataset with 172159 samples, with more balanced proportion of 1s and 0s.

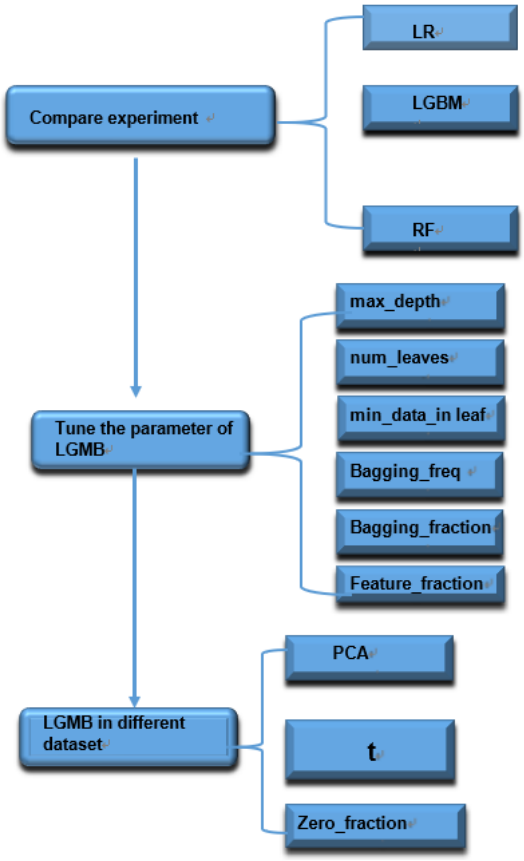
4. Methodology and Algorithm

There are many algorithms for classification including nearest neighbor, logistic regression, naive bayesian classifier, decision trees, support vector machine and some ensemble methods such as bagging, boosting and random forest. Due to time and computing power limit, in this project we will only apply some of them. Specifically, for non-ensemble methods, we will only use logistic regression. As all of the features are continuous and there's no explicit way to discretize them semantically like 'age' in Titanic dataset, decision tree will not be applied. As the size and dimension of dataset are high, KNN and SVM will not be applied for that they are too computational expensive. Neural network may be suitable for our project for that it could extract key features effectively, and it has shown superior performance in other tasks with huge data size and high feature dimension. This would be served as a part of future work.

In this report we will use random forest, logistic regression and light GMB.
Compared with single models such Logics Regression (LR) and Decision Tree (DT), the ensemble methods like Random Forest (RF) and Light GBM (LGBM) are more widely used in Kaggle because of them effective performance. The reason is that the single decision tree is more likely to over fitting. However, the random forest can be served as a meta estimator that suits multiple decision tree classifiers. On randomly selected subsamples of the dataset, through averaging the 'decisions', the prediction accuracy can be improved while the problem of over fitting also can alleviate.
The another tree-based ensemble method is Light GBM which is an extension of gradient boosted way like XGBoost and AdaBoost. By comparing the open source kernels, we found that LGBM is nearly essential for this project, thus we would pay more attention on this algorithm to tune the parameters. However, due to "no free lunch theorem", LR and RF will be used as well for comparison.

5. Experiment

Figure 13: Flow Chart of the Experiment Part.



In this section, we firstly compared three algorithms which are LR, LGBM and RF, and then we tune the parameters of LGBM model to improve the performance. Finally, we tuned the dataset by changing the different proportion of training dataset and validation dataset to see if we can get better results. The following parts explains each experiment in details.

5.1 Comparison Experiments:

In the first step, we will choose which algorithm has the best performance from the LR, RF and LGBM.

Figure 14: The Final Parameters of RF

```
estimator = RandomForestClassifier(n_estimators=1400,
                                   random_state=0,
                                   max_features='log2',
                                   max_depth=60,
                                   min_samples_split=25,
                                   min_samples_leaf=1,
                                   n_jobs=-1)
```

From the figure 14, it can be seen that the final parameters selected for random forest. By tuning parameters n_estmators, max_feature, max_depth, min_sample_split, min_samples_leaf, the random forest can improve the performance, get the better result (auroc_train 1.0000|auroc_validation 0.8669). The validation result is low but the training result can get 1.0 that means this model is overfitting. As a result, the random forest cannot get the effective result.

Figure 15: The Final Parameters of LR (c=0.1, c=1.0 has the same result)

```
estimator = LogisticRegression(random_state=0, solver='lbfgs', max_iter=10000, C=1.0)
estimator = LogisticRegression(random_state=0, solver='lbfgs', max_iter=10000, C=0.1)
```

Figure 15 shows the best parameter tuned for the logistics regression, while the other parameters are default value. It can be seen that we tuned C=0.1 and C=1.0 has the same result. Logistics regression has the worst performance which is train result is 0.8989 and validation result is 0.8650. Both the train and validation results are low but similar that means this model is underfitting. Another problem of logistics regression is that the tuning process is too slow. Both the logistics regression and random forest can not get the effective result. The parameters tuning result of LR and RF will be shown in the appendix.

Figure 16: The Basic Parameters of LGBM

```
param = {
    'bagging_freq': 5,
    'bagging_fraction': 0.4,
    'boost_from_average':'false',
    'boost': 'gbdt',
    'feature_fraction': 0.05,
    'learning_rate': 0.01,
    'max_depth': -1,
    'metric':'auc',
    'min_data_in_leaf': 80,
    'min_sum_hessian_in_leaf': 10.0,
    'num_leaves': 13,
    'num_threads': 22,
    'tree_learner': 'serial',
    'objective': 'binary',
    'verbosity': 1
}
```

Compared the logistics regression and random forest, the light GBM show the attractive performance. The figure 16 shows the basic LGMB model, this result is better than the tuned LR and RF. The final result of this version is train 0.9875 and validation 0.9021. Based on the above, comparison experiment we will tune the parameters for LGMB in the next section.

5.2 LGBM parameters:

The link below shows how do we obtain the parameters of LGBM (parameter tuning) in Python codes. And then we show some figures about the results.
https://github.com/13174420yang/UTS_ML2019_ID13174420/blob/master/assignment2/007_algo_lgbm.ipynb

Figure 17: All results of round 1: tune max_depth parameter

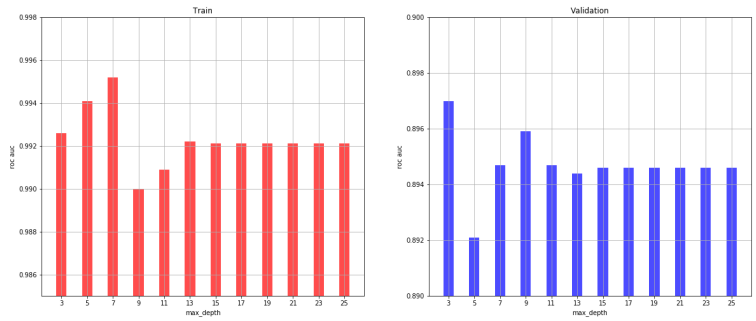


Figure 18: All results of round 2: num_leaves

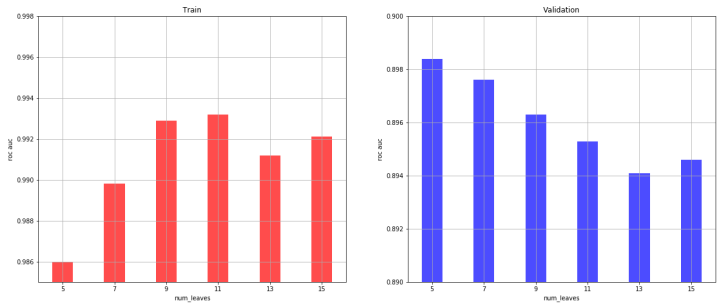


Figure 19: All results of round 3: min_data_in leaf

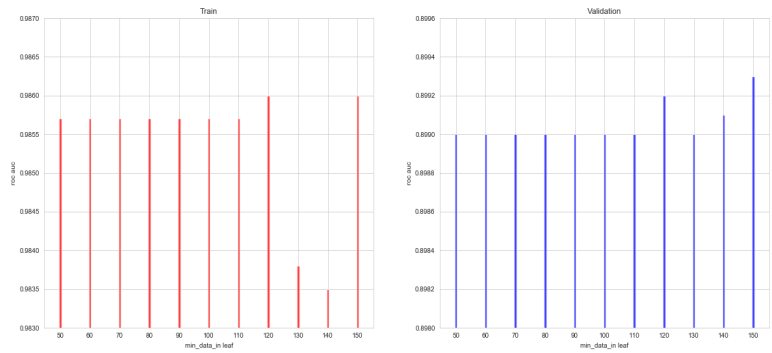


Figure 20: All results of round 4: bagging_freq

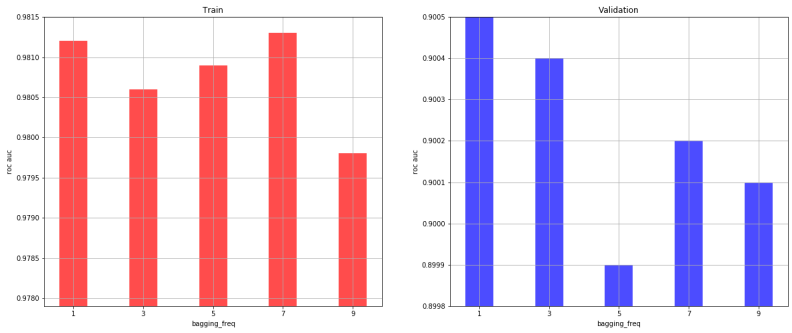


Figure 21: All results of round 5: bagging_fraction

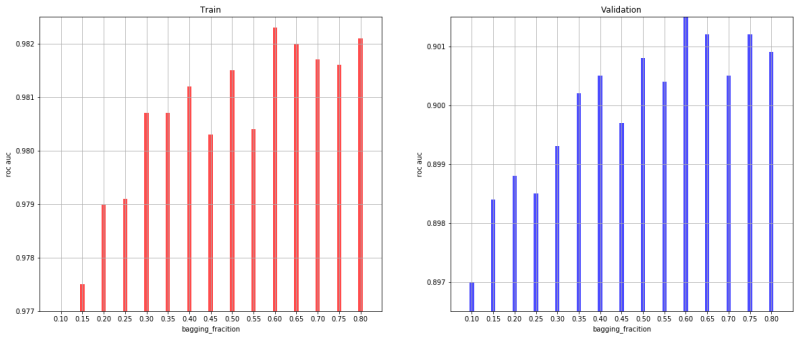
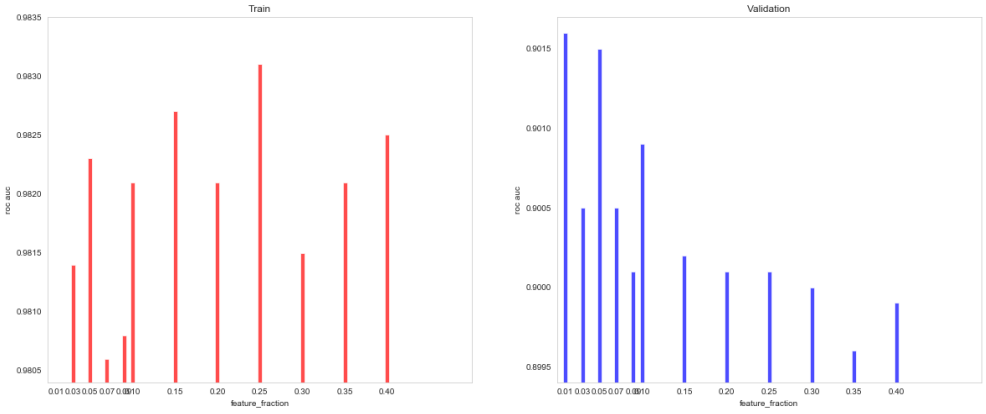


Figure 22: All results of round 6: feature_fraction



Compared with other algorithms, LGMB has a number of parameters need to tune, in this section, we tuned parameters including, max_depth (see figure 17), num_leaves (see figure 18), min_data_inleaf (see figure 19), bagging_freq (see figure 20), bagging_fraction (see figure 21), feature_fraction (see figure 22). The final version of the parameters will shown in the figure 23 and best result of each parameter can be seen in table 3.

Table 3: The best result of each parameter for LGMB parameters

Parameter	Optimal option	Train result	Val result
max_depth	15	0.9921	0.8946
num_leaves	5	0.9860	0.8984
min_data_in leaf	150	0.9860	0.8992
bagging_freq	1	0.9812	0.9005
bagging_fraction	0.6	0.9823	0.9015
feature_fraction	0.05	0.9823	0.9015

Figure 23: The final version

```
param_base = {
    'boosting_type': 'gbdt',
    'boost_from_average': False,
    'objective': 'binary',
    'tree_learner': 'serial',
    'verbosity': 1,
    'learning_rate': 0.01,
    'num_threads': 22,
    'metric':'auc',

    'num_leaves': 5,
    'max_depth': 15,
    'min_data_in_leaf': 150,
    'min_sum_hessian_in_leaf': 10,

    'bagging_freq': 1,
    'bagging_fraction': 0.6,
    'feature_fraction': 0.05
    }
```

Based on the parameter performance tuned in the last section. The final model chose the parameters have been shown in figure 23. Compared with other models, it can be seen that LGBM can predict target more accuracy. (train 0.9830|validation 0.9026).

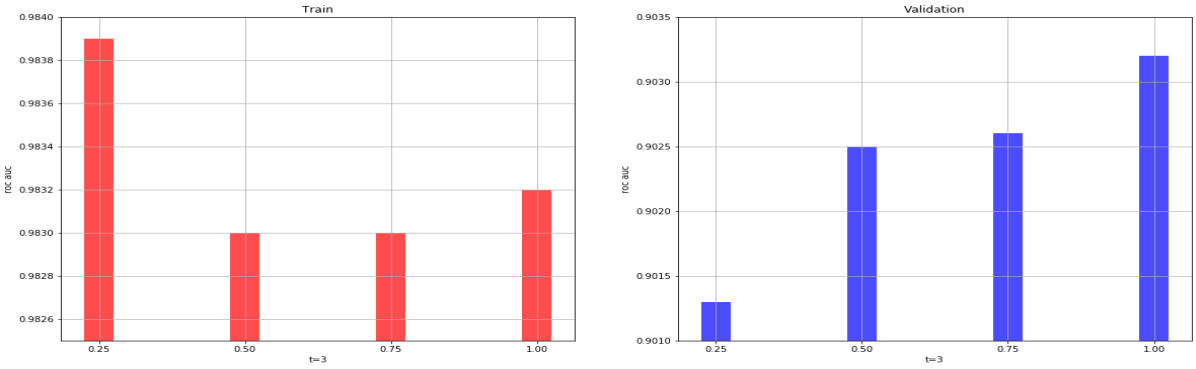
5.3 Datasets Experiments:

In last sections, we got lgbm's parameters on a training set and achieved good performance on the validation set. In this section, we will build other forms of training set, and check whether the performance on validation set can be better. Specifically, we will change the proportion of resampling and augmentation, and check whether the performance on validation set can be better.

5.3.1 Freeze the parameters of LGBM, compare its performance on different datasets

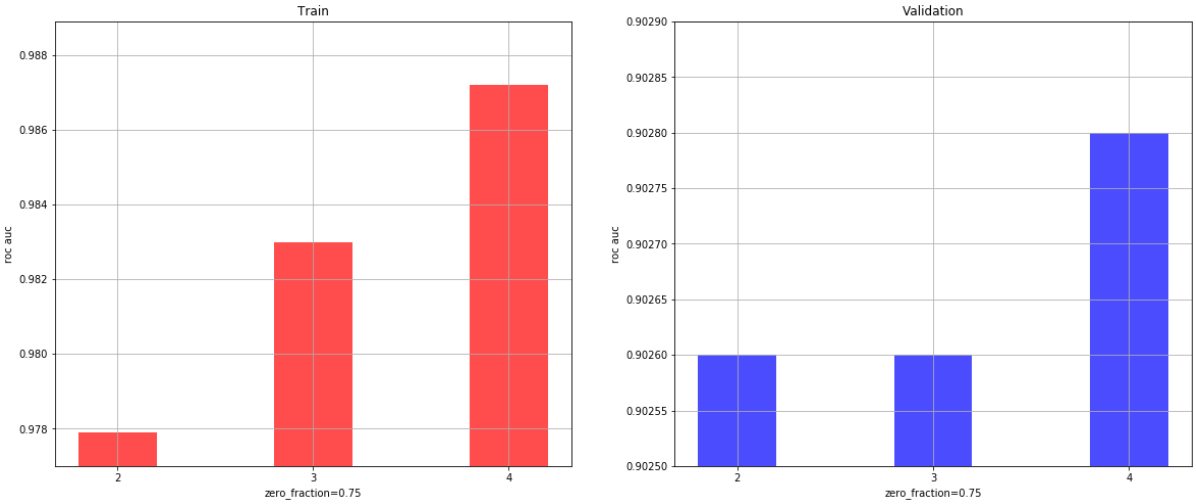
In this part, we will use resample and augmentation to show different proportion of 1s and 0s.

Figure 24: when t=3, compare different zero_fraction (0.25,0.50,0.75,1)



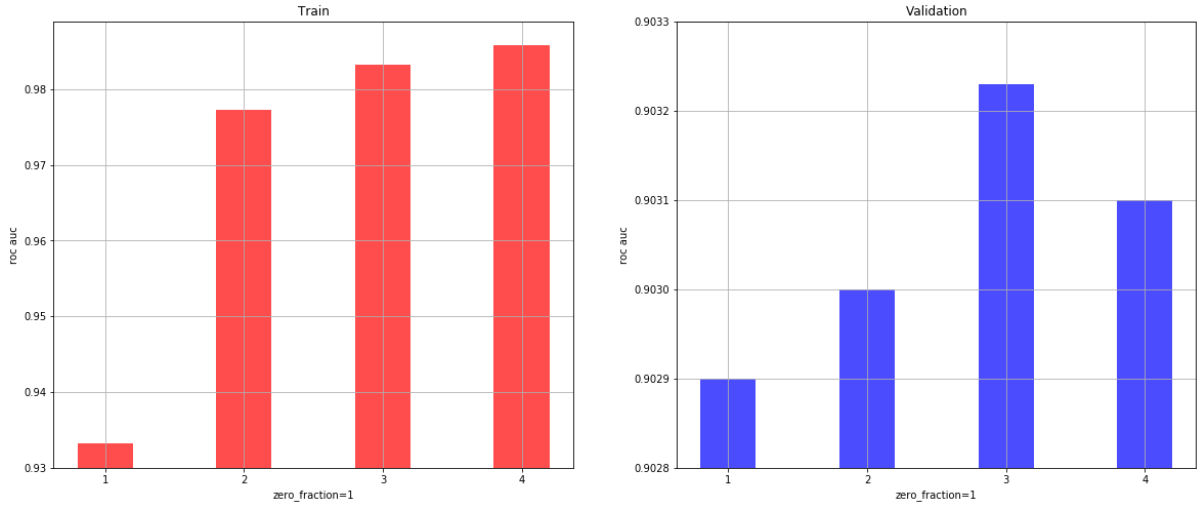
From this comparison, we recognize that when tripling the number of target data which the value is 1, the percentage of data we get from value 0 will have a huge impact on the final result. In the training data, when the percentage is 0.25, we can get the highest accuracy, however, in the validation data, it performs much worse than other proportions. When t=3, zero_fraction=1, the validation result reaches the highest level, which is 0.90323.

Figure 25: when zero_fraction=0.75, compare t=2, 3,4



When zero_fraction = 0.75, we compare the results under the different conditions of t values, which are 2,3,4. In these conditions, we can see that when t equals 4, it has the highest accuracy in both training set and validation set, which is 0.9872 and 0.9028 respectively.

Figure 26: when zero_fraction=1, compare t=1,2,3,4



In this figure, we can realize that when zero_fraction and t are both equal 1, the results are the bad in both training set and validation set. Moreover, the highest level in training data is 0.9858, when t = 4, and in validation data is 0.90323, when t = 3.
In conclusion, we compared the results with different parameters, we believe that when we add more 2 times datasets to target 1 and do not delete any data in target 0, the model will perform best, and the validation result is 0.90323.

The code can be shown here:

```
# this function is modified from https://www.kaggle.com/jiweiliu/lgb-2-leaves-augment
def augment(df, features, t):
    x = df.iloc[:,:-1].values
    y = df['target'].values

    xs,xn = [],[]
    for i in range(t):
        mask = y>0
        x1 = x[mask].copy()
        ids = np.arange(x1.shape[0])
        for c in range(x1.shape[1]):
            np.random.shuffle(ids)
            x1[:,c] = x1[ids][:,c]
        xs.append(x1)

    for i in range(t//2):
        mask = y==0
        x1 = x[mask].copy()
        ids = np.arange(x1.shape[0])
        for c in range(x1.shape[1]):
            np.random.shuffle(ids)
            x1[:,c] = x1[ids][:,c]
        xn.append(x1)

    xs = np.vstack(xs)
    xn = np.vstack(xn)
    ys = np.ones(xs.shape[0])
    yn = np.zeros(xn.shape[0])
    x = np.vstack([x,xs,xn])
    y = np.concatenate([y,ys,yn]).astype(np.uint64)
```

```
    features = pd.DataFrame(x, columns=features)
    labels = pd.DataFrame(y, columns=['target'])
    combined = pd.concat([features, labels], axis=1)
    return combined
```

```
def resample_and_augment(df_data, df_label, t=3, zero_fraction=0.75):
    """
    we will sample only 1/4 data from 0s, then combine them with 1s
    remember to combine features with label before feeding into this function
    """
    # combine data and target
    features = df_data.columns
    df_data = pd.concat([df_data, df_label], axis=1)
    df_ones = df_data[df_data['target'] == 1]
    df_zeros = df_data[df_data['target'] == 0]
    print("Original 1s {}, 0s {}".format(df_ones.shape[0], df_zeros.shape[0]))
    # augment 1s
    aug_ones = augment(df_ones, features, t)
    print("Now we have {} 1s".format(aug_ones.shape[0]))
    df_zeros_part = df_zeros.sample(frac=zero_fraction)
    print("part of 0s: {}".format(df_zeros_part.shape[0]))
    # combine and shuffle
    df_combine = pd.concat([df_zeros_part, aug_ones]).sample(frac=1)
    print("Combined: {}".format(df_combine.shape))
    train_data_aug,  = df_combine.iloc[:,:-1]
    train_label_aug = df_combine.loc[:,'target']
    return train_data_aug, train_label_aug
```

```
train_data_aug, train_label_aug = resample_and_augment(train_data, train_label, t=3, zero_fract
train_data_aug.shape
```

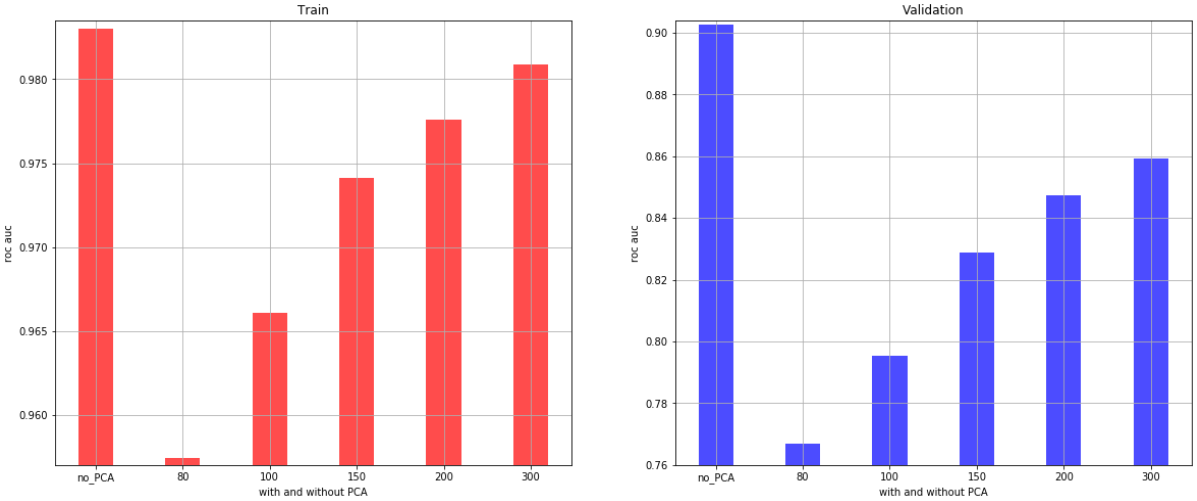
```
Original 1s 16049, 0s 143951
Now we have 64196 1s
part of 0s: 107963
Combined: (172159, 609)

(172159, 608)
```

5.4 PCA

5.4.1 With / Without PCA:

In previous sections, without dimension reduction, the basic result is training 0.9830 and validation 0.9026. Now we will use PCA to reduce dimension.

Figure 27: Results of dataset without PCA, without PCA and different parameters



From the two histograms above, it is obvious that the dataset without PCA operation can performs much better than the dataset operated by PCA model. More interestingly, for those dataset which handled by PCA, the accuracy increases with the growth of the value of the parameter called dim. When dim= 300 can get the better result (train 0.9809|validation 0.8592), but it cannot exceed the result which is without the PCA (train0.9830| validation 0.9026). After this comparison experiment, we decided to choose the dataset without PCA operation.

The code can be shown here:

Here I reduce the dimension to 60

```
def dimension_reduction(df, n_components=200):
    pca = PCA(n_components=n_components)
    df_processed = pca.fit_transform(df)
    df_processed = pd.DataFrame(df_processed,
                                columns = ['var_pca_{}'.format(i) for i in range(n_components)])
    return df_processed
```

```
df_data_pca = dimension_reduction(df_data_prep)
print(df_data_pca.shape)
df_data_pca.head()
```

```
(200000, 200)
```

6. Discussion

6.1 Summary of Conclusion from Experiment:

In this project we aim to predict the transaction from the consumer dataset of Santander bank. This is a two-class-prediction task where the proportion of classes are unbalanced and the samples are with high dimensional, continuous features. During preprocessing, we performed feature engineering to add some new features, and data resampling to handle imbalanced classes. Instead of submitting the prediction of testing set, for simplification we splitted the original dataset as training set and validation set and only evaluate the performance on the validation set. Three algorithms, logistic regression, random forest and light GBM, were used to perform prediction. The result shows that the performance of LGBM is much better than the other two methods, whose auc roc score exceeds 0.90. To study the influence of dataset, we varied the resampling proportion on training set. The result shows that compared with the base training set (t=3, zero_fraction=0.75), there exists more optimal proportion that leads to better performance. We also studied whether dimension reduction is helpful for our project, while the data with reduced features shows worse result.

6.2 Limitations and Future Work:

Although we achieved promising results via LGBM, there can still be some improvements which may lead to better results. First, we just performed basic feature engineering and parameter tuning, that more careful feature selection may further boost the performance.

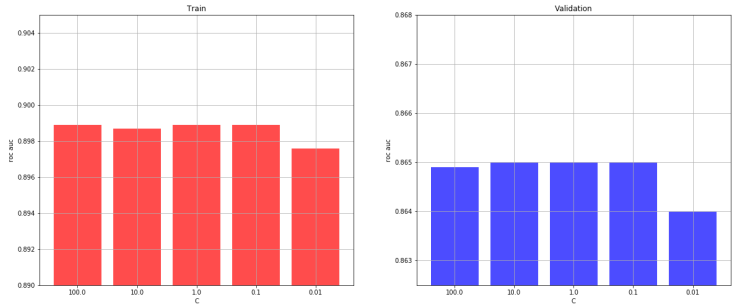
From the discussion and experience of other competitors, we found that there might be magic tricks helpful for prediction, such as serving the count of same values as feature, or trying to predict and remove fake data. These could be done as part of future work. Second, due to time and device limit we did not try some other powerful algorithms such as neural network, which could be suitable to handle large data size and high dimensional features. In terms of engineering, stacking different models may achieve better performance, while on the other hand, requires more intensive parameter tuning. Third, we found that by further changing the training set, the performance could be even better. In other words, it could be an iterative process to tune the parameters and modify dataset, which requires more trials. Last but not the least, in the future we can go beyond this binary prediction competition to dig more interesting information from the dataset, such as performing unsupervised clustering to find the implicit characteristic of consumers.

6.3 Ethnic Considerations:

In this competition, the Santander looks forward to predicting its clients potential trading behavior to produce better products and services, so the Santander bank uploaded 200000 clients' data to Kaggle for the data analysts to use, so the ethical of private customer data and identity should remain private is extremely necessary. First and most importantly, consider that the data is highly related to its clients' funds privacy, all the data sets that Santander uploaded is anonymous, which can avoid the disclosure of clients' information, such as the occupation of clients, the balance of clients' accounts. This kind of information is not only extremely privacy, also it may have a direct influence on the results of the targets, which means the trading behavior. Secondly, this competition is not only related to data analysis, but also some other fields and industries, including financial, policy and market environment, so for some teams or individual competitors, in order to achieve a high rank, they may have some cheat behavior, which is just focus on the result on the Leaderboard of kaggle but not to improve the model prediction. Although this kind of process can help achieve a good prediction, the Santander wishes all the competitors can output the results based on all the attributes, so all the attributes are also nameless. Thus, the Santander have followed the ethical of data privacy protection to guarantee the interests of its clients and avoid cheating in this competition.

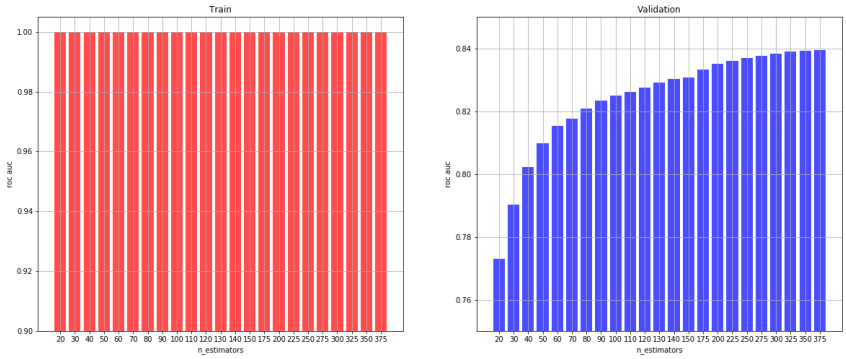
7 Appendix

7.1. The results of the LR model when parameters change

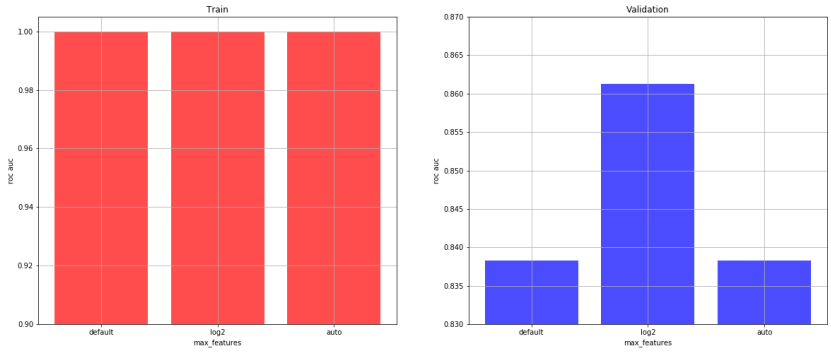


7.2 The results of the RF model when parameters change

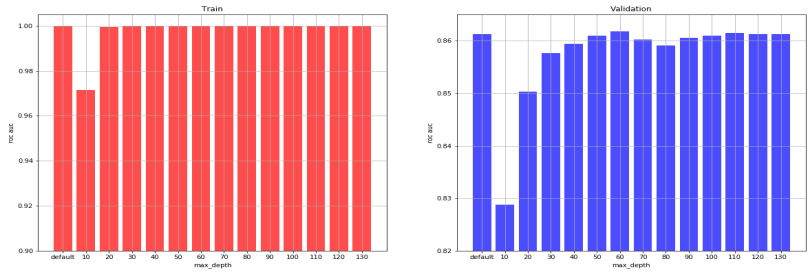
1) Round 1: n_estimator



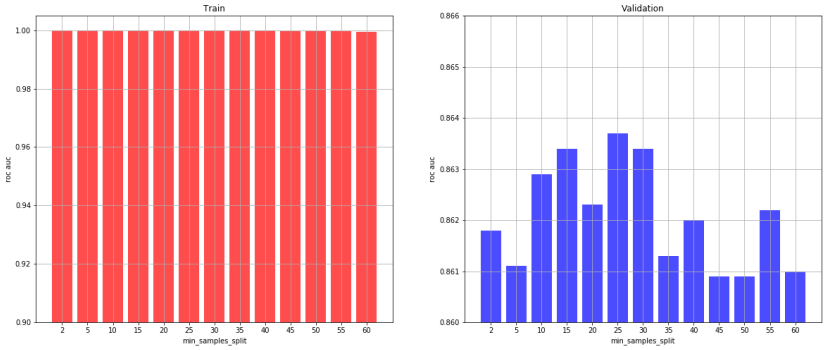
2) Round 2: max_features



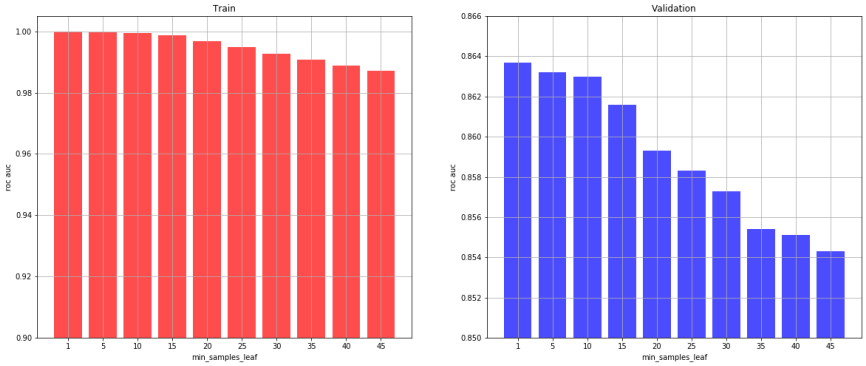
3) Round 3: max_depth



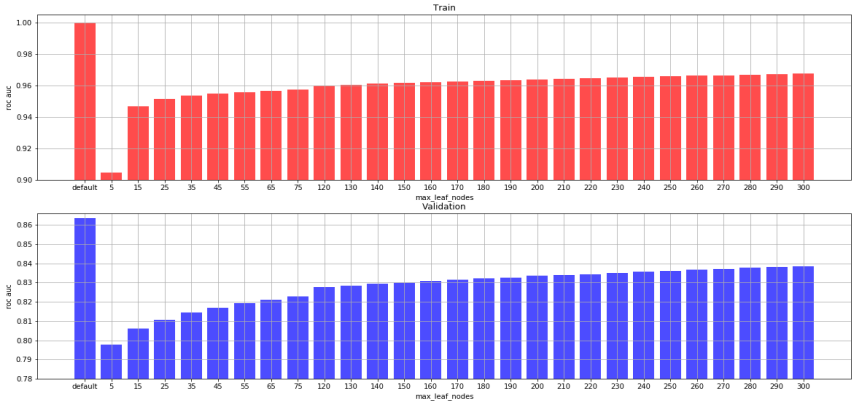
4) Round 4: min_samples_split



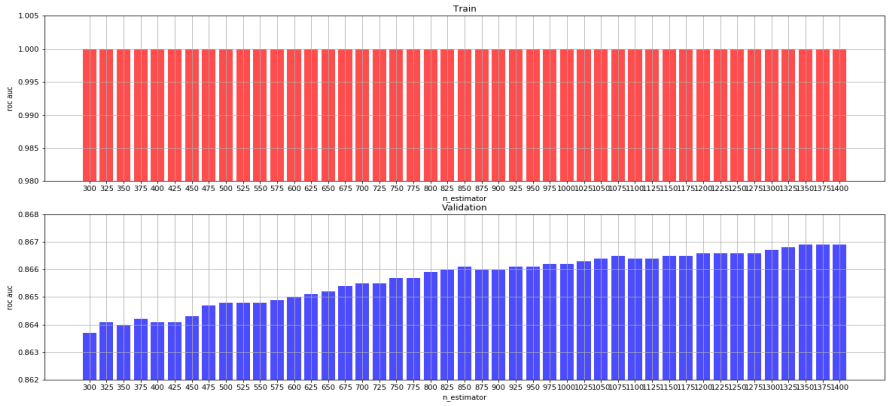
5) Round 5: min_samples_leaf



6) Round 6: max_leaf_nodes



7) Round 7: n_estimator



8) Round 8: The change from round 1 to round 7