ECE 681 Final Project

Wenyan Yao (wy48), Ye Tao (yt114), Zike Qin (zq21)

August 8, 2018

Contents

1 Introduction

1.1 Problem Background

In this project, we aim to tackle the problem, TalkingData AdTracking Fraud Detection Challenge, published in a kaggle competition (available at https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection). This competition aims at develop classification algorithm to detect click fraud based on available data to the services provider.

Click fraud refers to behavior of people taking advantage of computer program to automatically click certain online advertisement without real interest in the advertised item. By doing so, they gain illegal profits as the own of the website are paid according to the amount of clicks on the advertisement [1]. In this case, the TalkingData, which is a Chinese platform receives more than 3 billion clicks per day, initial this competition to reduce the mislead click rate and the resulted wasted money

1.2 Dataset description

The dataset provided by the TalkingData contains more than 100 million examples which are collected in four day. In this very project, we only take advantage of 3 million examples out of it, which are picked from the training dataset, considering the computation feasibility. Each example consists of 7 features which are listed below.

1. ip: "ip address of click."
2. app: "app id for marketing."
3. device: "device type id of user mobile phone (e.g., iphone 6 plus, iphone 7, huawei mate 7, etc.)"
4. os: "os version id of user mobile phone"
5. channel: "channel id of mobile ad publisher"
6. click_time: "time stamp of click (UTC)"
7. attributed_time: "if user download the app for after clicking an ad, this is the time of the app download" [2]

where the target/label of is provided as "is_attributed: the target that is to be predicted, indicating the app was downloaded". [2]

It is notable that `ip,app,channel,device,os` are categorical data. In this case, the reasonable way is to transform them into one-hot vectors in which different element indicates different class. However, this requires extra memory to store, process and compute the data, which make the whole problem computational infeasible considering our computation resources are limited to our PC. In this case, instead of using one-hot encoding, we use cumulative encoding which means transform first category to 1, second category to 2 and so on. Thus we can encode the original value into numeric value. For the date feature `click\_time` of which type is second-hours-day, we split the `click\_time` into three features which are day, hour and second of which type is int.

The dataset is chosen as it has the following advantages: First, the problem is one cutting-edge problem and the dataset is complex enough so that we can apply various learning algorithm learned in class and expect accuracy varying largely from each other. This serves as a good opportunity for us to explore the property of different algorithm lectured in class and thus enhance our understanding of them. Second, this dataset is highly imbalanced, around 0.264% examples are true (which are fraudulent activities). However, to the best of our knowledge, common machine learning algorithms do not consider imbalance data during design. This provides us a unique opportunity to modify existing algorithm to tackle this problem. Third, since there are only 7 features, the selected portion of data which is more than 3 million can be reasonably assumed to be representative.

1.3 Problem description

This problem is regarded as a binary classification problem to discriminate the fraudulent clicks. The input is the given data points of seven features and the output is the label which is a boolean variable represented by 1 as true and 0 as false. Meanwhile, since the accuracy of a highly imbalanced data is not informative as balanced data, we also apply the AUC (Area Under Curve) . The reason why AUC is a good substitute of accuracy is fully illustrated in Section.3.2.1 along with the effect of accuracy and AUC on logistic regression model.

In this project, we first construct more features based on the importance and correlation of the generic features. Then we apply three prevalent classification algorithm-logistic regression, random forest, and boosting. Their AUC achieves up to $91\%, 97\%, 98\%$ separately.

Furthermore, we try to use some ensemble methods such as voting, stacking to improve our model. And, in Section.6, we discuss three different methods we tried and analyze its advantage and disadvantage related to their performance on our CV sets. Specifically, hard voting achieves 92.9%AUC, soft voting achieves 93.3% AUC, and stacking achieves 95% AUC. Finally, we try to improve our model using stacking with LGB model which got 95% AUC.

2 Date exploration

2.1 Feature exploration

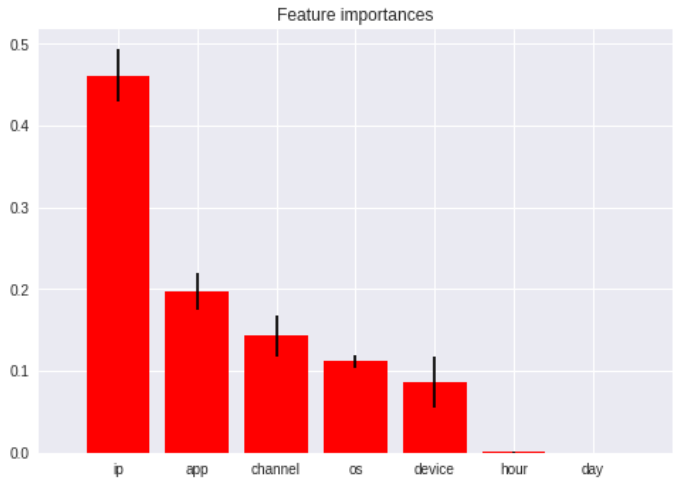

Figure 1: feature Importance

In the preprocessing stage, we first split the click_time into second, hour and day whereas the other features remain the same. The good of this stage of feature selection is to identify good features more relevant to the target. We first drop the seconds of a click time arguing that it is repetitive every hours and thus less related to whether a click is fraudulent. Meanwhile, since the rage of second is relative large, ranging from 1 to 60, the models we applied later still try to fit it whereas intuitively they are irrelevant. In that case, although fitting an irrelevant feature may improve training accuracy, it results in bad generalization in general.

Then, we start to explore the importance of each features using the random forest algorithm. The Random forest algorithm is described in detail in Section.4.1. Here, we briefly show the procedure to compute feature importance based on Random forest algorithm.

1. Goal: measure the importance of feature j.
2. Randomly select partial data to construct tree T and computer error called $error_T$.
3. Then construct a new tree using same data exception shuffle the values of all examples' feature j. Compute the error called $error_{T,permuted}$.
4. Repeat these process up to N times.

5. The importance of feature j is

$$\frac{1}{N}\sum_{T=1}^{N}(error_T - error_{T,permuted}). \tag{1}$$

In this project, we use the `tree.feature_importances_` provided by sklearn package [3] to compute the importance of each feature.

From Fig.1, it is noticed that the ip feature share the most importance. We think it is reasonable as the fraudulent clicker tends to own some specific ip address. In addition, we notice that the importance of day and hour and relative low. This is reasonable as intuitively the click time of fraudulent tends to spread across a whole day. And thus, the click time may have little impact on the accuracy of our machine learning model. Also, the range of hour is only 24 in this dataset and thus they may not be representative in this case. Also, we discover the hour's importance because there are only 3 day data. And thus we can expect the feature day is little representative.

	ip	app	device	os	channel	is_attributed	hour	day
ip	1.000000	0.006168	0.003141	0.003322	-0.004766	0.081353	0.001995	NaN
app	0.006168	1.000000	0.340599	0.329379	0.002912	0.058260	-0.003792	NaN
device	0.003141	0.340599	1.000000	0.915566	0.031558	0.004747	-0.000722	NaN
os	0.003322	0.329379	0.915566	1.000000	0.028950	0.003375	-0.000684	NaN
channel	-0.004766	0.002912	0.031558	0.028950	1.000000	-0.015653	0.001641	NaN
is_attributed	0.081353	0.058260	0.004747	0.003375	-0.015653	1.000000	0.000434	NaN
hour	0.001995	-0.003792	-0.000722	-0.000684	0.001641	0.000434	1.000000	NaN
day	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 2: feature correlation

To further illustrate the inter-relation between features, we also compute the correlation between features after cumulative encoding mention above in Fig.2. The correlation is calculated by calling `pandas.DataFrame.corr` of which output is the pairwise correlation of columns [4]. Drawn from Fig.2, we discover that os and device have the highest correlation which is 0.91. This shows that these two feature may share most of common information and thus one of them is enough when we construct new features.

In addition, we notice that correlations of day feature are NaN in this case. This is because the data we select are contiguous and the data are mostly the same day. The reason why we only use contiguous data is because that the total data is too large, more than 2G bytes, to fit in our personal computer's memory which makes the random select of example impossible. However, we argues that the day of click time may have little effect on the accuracy of our model as assumption that fraudulent happens with equal possibility every day is reasonable.

In summary, we explore the importance and correlation of features in this section which provides basic information regarding our dataset for our next feature engineer phase and the following algorithm sections. Through this section, we discover that ip has the highest importance among all features. Meanwhile, os and device have the largest correlation indicating that they share a lot of common information. In addition, we found that in general, the click time owns the lowest feature time which imply less computation is possible by dropping parts of this features.

2.2 Feature engineering

In this section, we create more features based on the correlation between the generic features as good construct features generally improve the performance of machine learning model. The feature constructing strategy we applies in this project is that examples which shares same value of one or more selected features are classified into different groups . Then we compute some statistical properties within each group such as the sum of examples in a group or the unique number of one

feature other than those used to classify the groups and then we assign these statistical results as new features to those example.

First, We construct the group as the following

1. $['ip','day','hour']$
2. $['ip','app']$
3. $['ip','app','os']$
4. $['ip']$

initially, we use ip, $['ip','app']$ as they have the highest importance than others. Then we can select either the third important feature *device* or the fourth *os* to form a group along with ip, appas their correlation are high, 0.9, in this case. Then we pick the time feature, *hour*, as it is least correlated to $['ip','app']$.

Next, we calculate the following statistical results for different features. First, compute the average of numeric features, $hour, day$, within each group. Second, count the unique number of categorical features within each groups other than those used to group the examples. For example, count the number of unique channels happens in a group consists of all examples share same ip and app. Then, we compute the ratio between the unique number of categorical features and the number of examples in a group.

Another feature, next click time of one ip, our construction is based on life experience that one person may click ads conforming to a certain pattern and thus the interleaving time also has certain pattern. This is done by searching two consecutive click of one ip and then append the second click time to the first example. Due to this click time has a property that if no one click certain link one more time which means the next click time is infinity thus we try to use a mapping strategy to map all click time into $[0,2^{32}]$. Thus we have equation like :

$$\Delta_t = min(T_{n+1} - T_n, 2^{32}) \tag{2}$$

$$\tag{3}$$

One can find that the most cases are either one click this link in a short time or will not click it twice. Thus this feature will have concentrate on head and tail of the intervals like Figure.4.

3 Logistic Regression

In this project, we first apply the logistic regression mode to the given dataset described above. Logistic Regression is a binary classifier that maps a linear regression to a logistic function within range $[0, 1]$, which serves a good starting point for binary classification problems. This is due to its model simplicity which makes the computation less time-consuming compared with other prevailing learning algorithm such as the random forest and boosting algorithm we will apply latter.

3.1 Model Description

Logistic regression model is a simple linear model which implement the below formula to predict categorical outcomes

$$y = sign(w^T x), \tag{4}$$

where w is the weight vector and x is the examples. Output y is binary value, $0, 1$.

The core idea of logistic regression is to try to predict the probability that a given example belongs to the "1" class versus the probability that it belongs to the "0" class by applying logistic function to the linear function [5]:

$$\begin{cases} p(y=1|x) = \frac{1}{1+\exp(-w^T x+b)} = h_w(x) \\ p(y=0|x) = 1 - h_w(x). \end{cases} \tag{5}$$

Given the above formulas, we can show the process to train the weight vector w to minimize the cost function which is show below

$$L(\theta) = -\sum_i \big(y_i \log(h_w(x_i)) + (1 - y_i) \log(1 - h_w(x_i))\big) + C\|w\|_2^2, \tag{6}$$

where C is a hyper-parameter regularizing the model complexity. In this case we apply the sklearn.linear_model.LogisticRegression from sklearn package which implement a gradient descent algorithm to compute the optimal w which minimize the loss function [6].

3.2 Data processing based on logistic regression model

In this section, we perform the data pre-processing by two means. First discuss the issues of using accuracy as evaluation metrics when the dataset is highly imbalanced. Then we discuss possible improvement brought by data normalization.

3.2.1 Imbalanced Data

Recall that the dataset is highly imbalanced with 2992058 entries of no-click and only 7942 clicks, where un-clicked sample (class 0) makes up 99.74% of the dataset and clicked sample (class 1) makes up only 0.26% of the dataset. When dealing with this type of data, many small changes to the model has to be made to make sure it is trained to achieve the goal.

Since that the given dataset is highly imbalanced and thus accuracy is still high when all data are classified as one class. In this case, we can use AUC to show the performance of our classifier. This is because that AUC computes the area of ROC curve which plots True positive rate $\frac{\#Truepositive}{\#positive}$ versus false positive rate $\frac{\#Falsepositive}{\#Negative}$. By apply the ratio, AUC can eliminate the numeric issues brought by accuracy. For example, given a data set contain 99 positive and 1 negative. When the classifier class all data as positive, the accuracy of the classifier is 99% whereas the false positive rate is 100%. This high false rate apparently indicate that the model severely mis classifies the dataset.

In addition, we need to pay extra care to the split of training dataset when we apply the cross-validation to optimize the parameters. In this project, we use a 3-fold cross-validation method to evaluate the model. Since the data is highly imbalanced, if we only adopt random number generator to split the data, it is possible that one portion of data does not have any class 1 sample in it. To avoid that, when splitting training and validation set, use `StratifiedKFold` to make sure that the ratio of the two classes stays the same as the original dataset [3].

In this section, we also performs the weight procedure to tackle the imbalanced data problem. This is a common method applied to rare events problem of which core idea is to assign an extra weights to data during training. The weight of one sample is inversely proportional to its class frequencies

$$weight_i = \frac{\#sample}{\#class \times \#(samples)_{class=i}}, \tag{7}$$

where weight i is the weight assigned to examples belong to class i and it is the number of total samples divided by number of class and number of samples in class i. In practice, this procedure is done by setting the `sklearn.linear_model.LogisticRegression`'s parameter `class_weight = 'balance'` [3].

3.2.2 Feature selection

After setting AUC as out main evaluation matrices, we start to perform the feature selection phase. In this phase, we first evaluate the AUC of logistic regression model based on the generic features of the original dataset. Then, we step to evaluate the effect of individual new constructed features on the performance of the model. For the sake of time, we only plot the changes of performances by adding features consecutively. This is shown in Fig.3.

Fig.3 plots the AUC versus the number of features we used. Drawn from this figure, we notice that the overall performance of the model become better as we inputing more and more features to the model. This shows that our ideas of constructing new features are valid. This only exception happens when we append the last feature which is the next click time. In the next subsection, we explore the property of this feature.

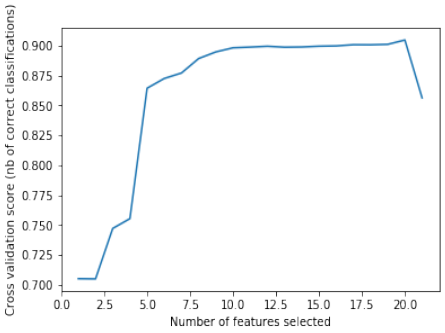

Figure 3: Feature Selection

3.3 data normalization

In order to explore why the feature worsen the performance, we first plot the distribution of this feature shown in Fig.4.

Fig.4, plot the distribution of the data among `next_click` for both labels (Fig 4) From the fig, we notice the feature `next_click` has extremely wide range, from 0 to 641306200. Recall that is due to we desired to hash the click time into a large range. Meanwhile the label has a bimodal distribution, and the scale of the values will make it difficult for the linear model to deal with.

This large range and imbalanced distribution impose numeric difficulty to training process. To be specific, weight of logistic regression model for `next_click` is around 1e-6, which rescale the range of $w^T x$. However, this worsen the performance shown in last section.

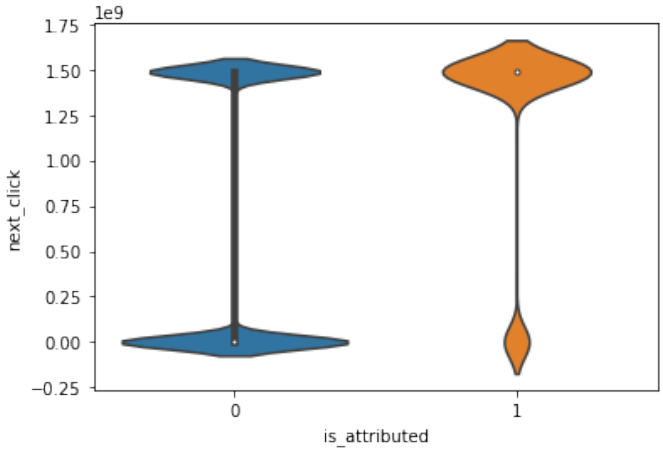

Figure 4: NextClick feature distribution

In order to take advantage of `next_click` feature, it needs to be preprocessed to reduce its variance. In this project, we explore three possible ways to address the problem: 1. Rescale the feature to range [0,20]. 2. Apply function $f(x) = \log(1 + x)$ to the feature to decrease the range of `next_click` feature. 3. Remove the feature as a process of feature selection.

In examine their effect, we plot the results when apply these three methods to logistic model shown in Fig.5.

Drawn from the figure, among the three method, a linear normalization may have slightly worse performance than logarithm because log function changes faster for relatively small values which takes up the majority of the data, hence easier to separate. Dropping this feature also works, but compared to the other two methods, the information would be completely lost in this way. The result (Fig 5 and Table 1) affirmed the analysis, where the log normalization achieves the highest results. Therefore, choose only logged data to compare with previous results.

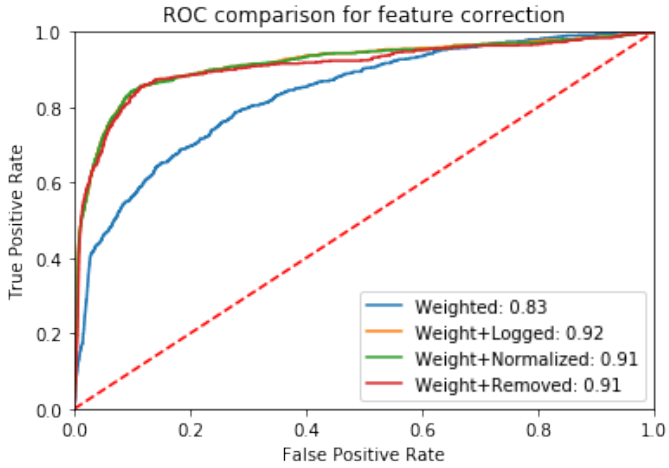

Figure 5: ROC comparison for the three methods

Table 1: AUC score comparison for the three methods

Dataset	AUC Score
Weighted Data	0.833367
Log-Corrected	0.915872
Normalized	0.914684
Feature Removed	0.907995

3.3.1 experimental results and discussion

In this section, we validate our assumption that AUC is a better matrices than accuracy when the dataset is imbalanced by showing results computed by logistic regression model shown in 2 and Table. 3.

Table 2: accuracy score for different data

Dataset	Training	Testing
Original Data	0.997353	0.997353
Weighted Data	0.716394	0.715957
Log-Corrected	0.860994	0.86054

Table 3: Cross-validated mean AUC score for different data

Dataset	AUC Score
Original Data	0.217184
Weighted Data	0.833367
Log-Corrected	0.915872

For the scoring metrics, choose AUC score instead of accuracy score. The reason for this choice is that, the accuracy ($\frac{\#TruePositive+TrueNegative}{\#NumOfSamples}$) can not be misleading and extremely high even if the classifier simply predict 0 for all data. See the comparison of two methods in Table 2 and Table 3 on three datasets: Original data, weighted data, and log-corrected data(weighted data with $f(x) = \log(1+x)$ function applied to `next_click` feature).

As shown in the tables, the accuracy achieved 99.74% on the original dataset. However, this does not mean that the model is good, because it simply classifies all samples as 0. On the other hand, the original data got an extremely low AUC score, because it does a poor job with regard to FPR($\frac{\#FalsePositive}{\#Negative}$) and TPR($\frac{\#TruePositive}{\#Positive}$), i.e., the classifier cannot balance between having high TPR and low FPR. (Fig 6)

In summary, we illustrate that AUC is a better matrices to evaluate our algorithm as it shows significantly difference between pre-processed dataset, apply weighting procedure and log normal-

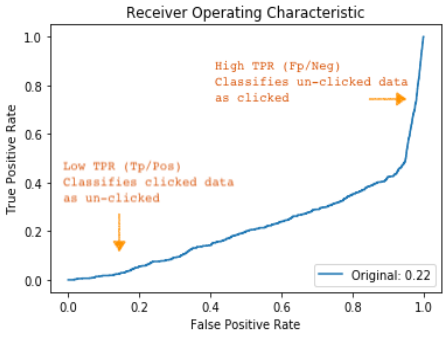

Figure 6: ROC analysis

ization procedure, as well as the original dataset. On the other hand, the increased AUC after weight and log normalization proves the effectiveness of our data processing methods. After data pre-processing, we begin to fine-tune the parameters of the logistic model in order to further improve the accuracy. Note for the rest of the project, we stick to the dataset processed by weight and log normalization methods unless explicitly stated.

3.4 fine-tuning parameters of logistic regression model.

Recall that the logistic regression model has only one parameter to tune which is the C parameter in Eq.6. By increasing the value of the C parameters, we allow the model to be more complex in order to fit the training dataset well. Recall that this may impose overfitting problem as the C become large. In order to avoid this problem, we apply the cross validation methods to approximate observe the overfitting of the model. Recall that in this case overfitting happens when the AUC keep going up on training dataset whereas the AUC decreases on the testing dataset.

This fine-tuning process is done by doing a grid search for C from [0.001, 0.01, 0.1, 1, 10, 100]. We here take advantage of the `sklearn.model_selection.GridSearchCV` function provided by sklearn package [3] to do grid search. The result is shown below.

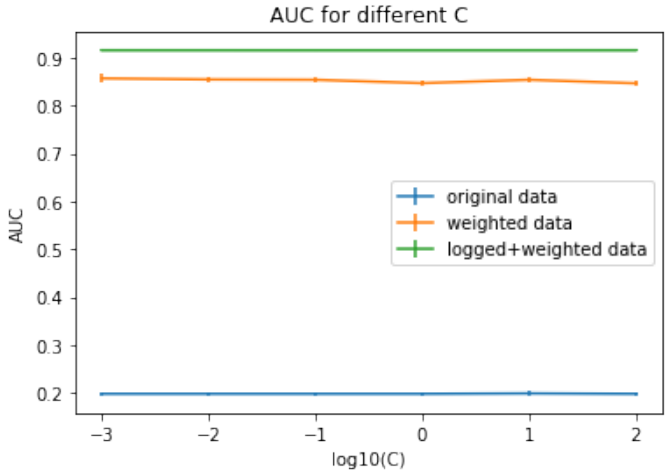

Figure 7: AUC score v.s. C

Fig.7 plots the mean AUC versus the C parameter on three different dataset. Recall that we perform cross-validation on three folder split from training dataset. The mean AUC is calculated from the average of three folders. In this figure, the blue line plots the AUC on the original dataset containg the generic seven features and newly constructed features described in Section.2.2. The orange line plots the AUC after we impose the weight methods described in Section.3.2.1. The green line represents the AUC after we perform log normalization on the click_next features as well as the weight methods. The mean AUC of the three line are around 0.91, 0.87, 0.22 separately.

Drawn from Fig. 7, we notice that the these three line are all flat line which means the change

of C has little impact on the the overall performance of the model. Meanwhile, we note that the mean AUC which represents the AUC on testing set doesn't show the turning point. This means that the model is still under-fitting even we increase the C up to 100. This phenomenon can be explain as the logistic regression model is still a linear model. That means the dataset is separated by a linear hyperplane. Under such circumstance, increasing the C only increases the norm of such hyperplane whereas its shape is still remain the same.

3.5 Results and discussion

In this section, we show the tuned logistic regression model of which ROC is plot in Fig. 8. We found that for imbalanced data LR get a unacceptable AUC of 0.22, we think the possible reason is that LR is a linear model and are very sensitive to imbalanced data. We also discover that increasing the C parameter means increasing the norm of hyperplane vector can not increase the overall performance of the model, which means the model is still under-fitting. Therefore, we can try to improve the the performance by increasing the complexity of models to achieve better performance by switching to more complex model such as Random forest and boosting algorithm.

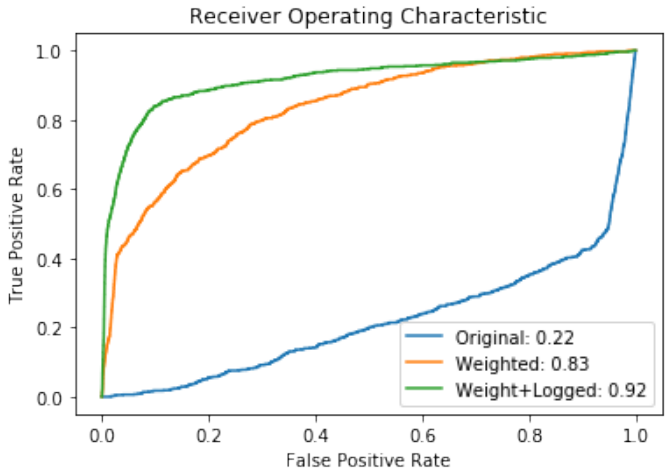

Figure 8: ROC when C=10

4 Random Forest

In the last section, we discover that logistic Regression model can achieve up to 91% auc score at most. One of the possible reason is that Linear Regression model is of relative simplicity compared to others, e.g. Random Forest and boosting. Therefore, we apply the random forest algorithm in this section considering it has more flexibility to increase the complexity of the model by increasing the depth of each tree and the number of trees. Meanwhile, the rising number of trees may reduce the variance of each decision trees and thus increase the stability of our learning algorithm and improve its generalization.

4.1 Model description of Random Forest

The procedures to grow/train a Random Forest is described as the following: First, bootstrap n size data and use them to grow a decision tree. The procedure to grow a tree is specified as: First choose m features out of all features. Split the tree according to Gini index. Here, the tree is continue to grow by splitting on leaf until the maximum depth is achieved or no more data to split. Repeat the above procedures up to T time to get T decision trees.

In order to predict a data point $\mathbf{x}$, performs prediction on all decision trees and use the majority vote.

Table 4: AUC versus max_depth on test set

max_depth	5	9	13	17
AUC on training set	0.94347451	0.97024013	0.98806489	0.99757101
AUC on testing set	0.94239336	0.9639067	0.96747167	0.9583917

4.2 Description of sklearn Random Forest Classifier

In this project, we use the sklearn.ensemble.RandomForestClassifier implemented in sklearn package to train, predict and tune the hyper parameters of Random Forest Classifier. Here we first specify some parameters and evaluation criteria.

Here we use the AUC as our main criteria of which reason is fully discussed in 3.2.1.

Another hyper parameters we need to explicit specify is the max of depth of each tree which is defined as max_depth is sklearn package. As mentioned this serves as one effective method to avoid overfitting when training a single decision tree.

Meanwhile, we also need to defined the number of decision trees in the random forest which is n_estimators in sklearn. Meanwhile, we set the criterion='gini' and set bootstrap='true' to allow the classifier to use bootstrap when grow different trees. This serves as a method to reduce the variance of the Random Forest. As we expect more versatility by training trees on different portion of dataset, the average results of those decision trees become less variant and more stable in the sense of generalization.

4.3 Grid search of Maximum of depth

In this section, we are going to show the fine-tuning results of parameter max_depth by fixing the number of estimators (the number of decision trees) as 13. To achieve the fine-tuning results, the point when the classier begin to over-fit need to be found. Here we use the gird-search function provided by the sklearn. This is done by using cross-validation on three folders and compute their average metrics.

Here, we first search the max_depth in the order of ten and found that the best results happens between 10 to 20.

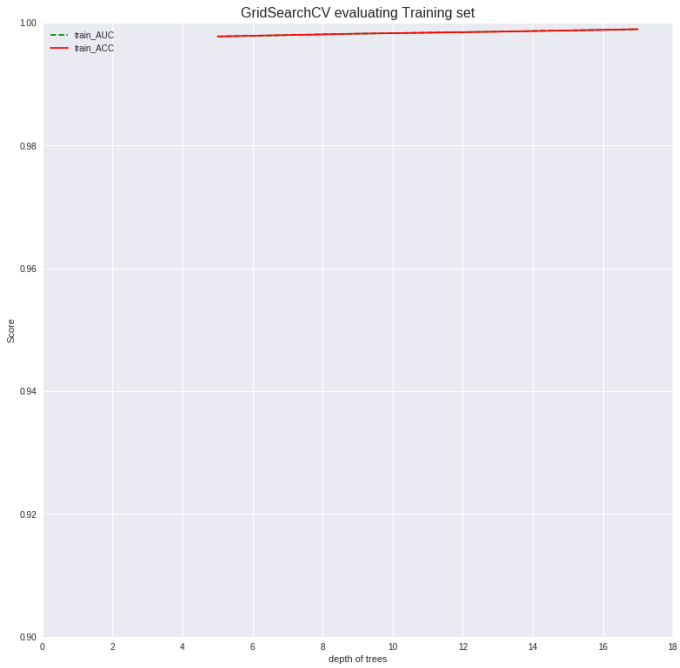

Figure 9: Accuracy and AUC versus max_depth on training set.

From the above figures, we found that by increase the max_depth of each tree, the AUC of training set increases. However, the AUC of testing set began to decrease when max_depth is 13.

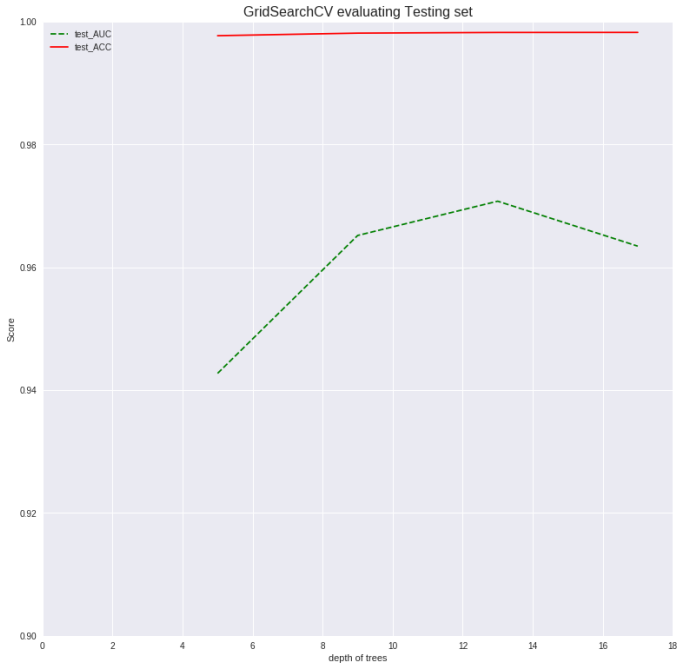

Figure 10: Accuracy and AUC versus max_depth on test set.

This shows that the Random Forest begin to overfit after that. Therefore, we set the max_depth as 13.

4.4 Grid search of number of trees

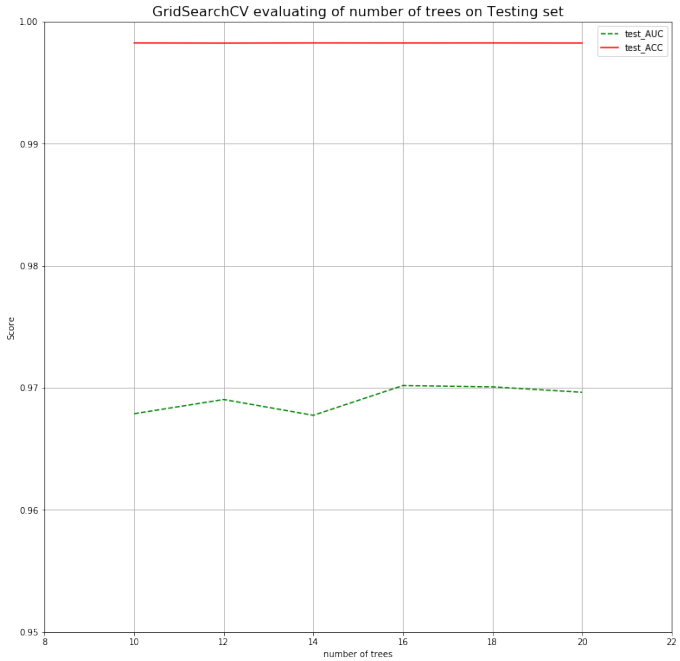

Figure 11: Accuracy and AUC versus n_estimators on training set.

Here, we perform the similar grid search procedure on number of trees. And we discover the similar results that the classifier begin to overfit when n_esimator is 16. And thus we set the max_depth as 16.

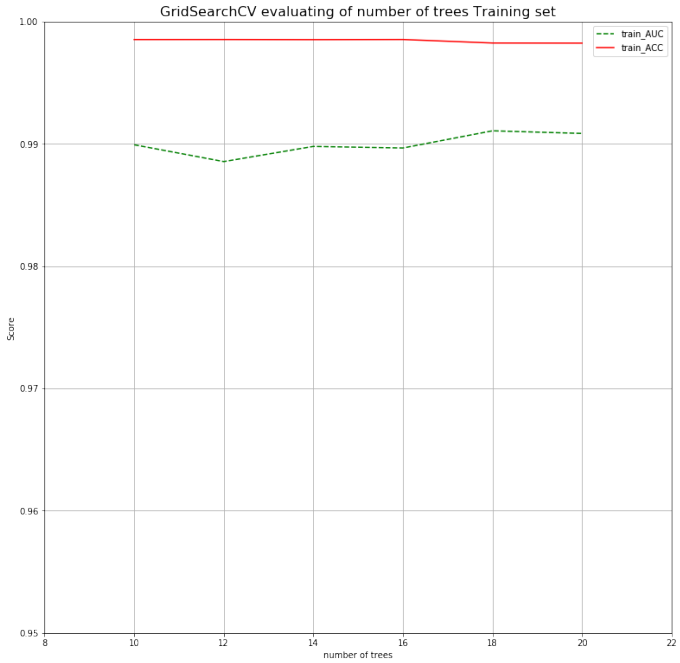

Figure 12: Accuracy and AUC versus n_estimators on test set.

Table 5: AUC versus num_estimator on test set

n_estimator	10	12	14	16	18	20	22
AUC on test set	0.96623241	0.96720696	0.96841928	0.97052279	0.97003114	0.96892305	0.96991107
AUC on train set	0.98778489	0.98971964	0.98890291	0.99131255	0.98923809	0.99076844	0.99186171

4.5 Final results and discussion

In this section, we apply the random forest on the targeted dataset. Specifically, the optimal estimator of this classier is found by grid search in which the maximum depth of each tree is 13 and the number of decision trees is 16. The best AUC we have achieved so far 97%.

The gap between perfect classification and the achieved results by Random forest may lie in the bias of each trees. By which I mean, the random forest can not reduce possible bias of dataset e.g. noise factor in the data, by iterating through the Random forest. This is because that each tree is independent of each other. Therefore, to further improve our classier, one possible way is to generate a new tree based on the last trees which makes them correlated to each other.

5 Gradient descent boosting tree (LightGBM)

In last section, we found that random forest is a good model complex enough to fit the dataset. However, in random forest, every trees are bagged together which means it can only reduce the variance of this problem which means it is robust to outliers, but its performance is still not good enough. In order to reduce bias of this problem, we decided to try Gradient descent boosting tree (GDBT)[7] model. Compared to RF, GDBT use trees to learn residual of last tree instead of using all trees to learn same target. Other reasons why we choose this model are that boosting model can boost weak classifiers to generate a strong classifier and can avoid overfitting efficiently. We also try to use this model to solve imbalanced data problem and construct a useful objective function. In last two parts, we will show the results of our model and comparison between hyperparameters, and we will discuss the result.

5.1 Model description

In this section, we use LightGBM[8] as our GDBT model. In this model, LightGBM used boosting method and decision tree to solve classification problem.

Main idea of this model is using decision tree in each iteration to search optimal solution. Then next iteration will use another decision tree to learn residual of last iteration. Figure.13 and 14 shows the residual learning algorithm. We assume there is a decision tree using age as label and split on income feature. Then one can find in the next iteration GDBT model will learn residual of last tree which is -1,1,-1,1 in this example. Finally, we can use another feature based on information entropy gain to generate a decision tree to solve the residual.

Secondly, LightGBM use objective function for each iteration as below:

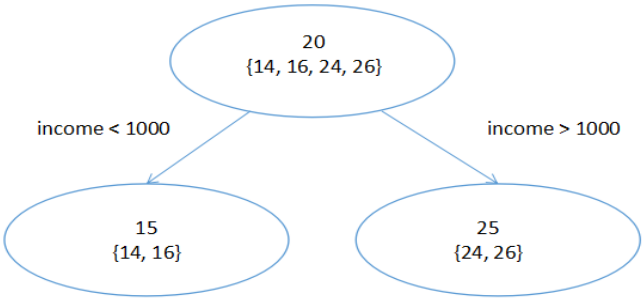

Figure 13: residual

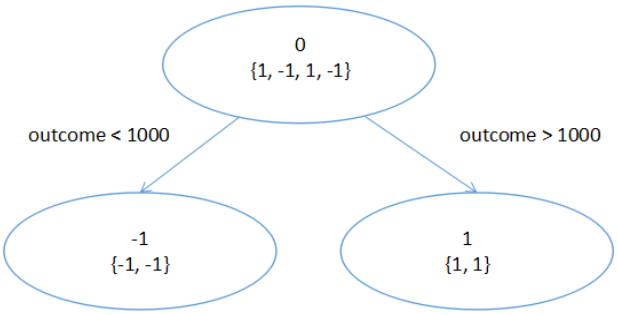

Figure 14: residual

$$Obj^{(t)} = \sum_{i=1}^{n} l(y_i, y_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{8}$$

Where $l()$ denote the loss function, f_t denotes function we learned from tth iteration, $y_i^{(t-1)}$ denotes predicted results after $(t-1)$th iteration, and Ω denotes the regularization operations. Specifically Ω is defined as below:

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2 \tag{9}$$

Where T denotes number of leaves in tth tree, w_j denotes weight of jth leaves. Thus γ denotes L1 regularization term and λ denotes L2 regularization term.

Assume $l()$ function is first and second differentiable, define g as first derivative and h as second derivative. Equation 8 and 9 can be simplified as:

$$Obj^{(t)} = \sum_{j=1}^{T} [(\sum_{i} g_i)\omega_j + \frac{1}{2}(\sum_{i} h_i + \lambda)\omega_j^2] + \gamma T \tag{10}$$

Finally, Equation 10 is the final objective function of LightGBM. We can use this GDBT model to solve our problem. Its main advantage compare to two model we mentioned above is that first it is a tree based model thus it can solve non-linear problem, and secondly it is a boosting model which means it can reduce bias of this problem, thirdly it has l1, l2 regularization term which can avoid overfitting, finally it has good performance on dense feature. These four properties are the reason why we try to use this model to improve the performance.

5.2 Imbalanced data and Objective function

In this problem, as mentioned in data analysis one can find that only 0.26% of data is with positive label which means it is a extremely imbalanced data problem. And we can find in Logistic regression and Random Forest section if we do not deal with imbalanced data the performance will be unacceptable. Thus we try to use logistic loss function and weighted data to solve this problem.

5.2.1 Objective function

This is a binary classification problem, thus we try to use logistic loss as a loss function in Equation 8, because logistic loss is based on the probability of certain label which will have better performance than other loss function. We can define loss function as below:

$$l(y, f(x)) = \sum_i [y_i log(h(f(x_i))) + (1 - y_i) log(1 - h(f(x_i)))] \tag{11}$$

Where $h(x)$ function is the sigmoid function which is defined as:

$$h(x) = \frac{1}{1 + exp(-x)} \tag{12}$$

5.2.2 Imbalanced data

Another solution to solve imbalanced data problem is that give the labels different weights, thus when we try to minimize the loss function we will consider one of label more. In this problem, we have less positive label, thus we give a high positive weight (bigger than 1) to positive labeled points. Thus we can rewrite Equation 11 as below:

$$l(y, f(x)) = w l_{y=1}(y, f(x)) + l_{y=0}(y, f(x)) \tag{13}$$

Where w is the weight we add to positive labeled points.
How to find the value of w is also a problem, we try to use two ways to find its value. The first way is that try to tune it as a hyperparameter, the second way is auto-defined every time when the model extract data from data set and defined w as:

$$w = \frac{\sum_i 1_{[y==1]}}{n} \tag{14}$$

Then we can find the AUC comparison between before we change objective function and after we change the objective function in Table.6. Where 'cross#' denotes the # of cross-validation fold, mean denotes the mean AUC of this 3 folds, origin denotes no weighted data. We can also find

Table 6: Imbalanced data

	cross1	cross2	cross3	mean
origin	0.970670	0.972142	0.970189	0.971000
w=100	0.972343	0.974231	0.975353	0.973976
w=200	0.980712	0.982650	0.980903	0.981422
auto weighted	0.980623	0.981934	0.980734	0.981097

mean cross-validation AUC comparison among different w values in Figure. 15. Thus, we can find that after dealing with imbalanced data, performance of our model increased clearly (0.97 to 0.98), and w equal to 200 have same performance as auto-weighted method. But w=100 has negative performance, we think the possible reason is that this problem is extremely imbalanced, w = 100 can not have let positive label get enough weight for our model.

5.3 Training and Hyperparameter selection

In this part, we will discuss training detail and hyperparameter selection of our model.

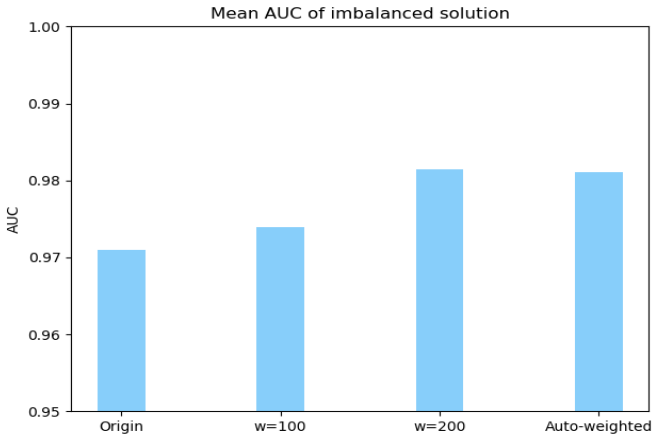

Figure 15: Mean AUC of balanced

5.3.1 Training

When training model, LightGBM used histogram algorithm[8] to find best split position for every decision tree. And compared to original GDBT, lightGBM used leaf-wise to generate a decision tree instead of level-wise which means generate follow one path from root to leaf then generate other path.
On the other hand, same as GDBT[**?**], LightGBM use gradient decent to train Equation.10, which means use gradient decent to find the best f_t minimizing Equation.10 for tth iteration. This method can defined as below:

$$y_1 = y_0 + \epsilon f_1(x) \tag{15}$$
$$y_2 = y_1 + \epsilon f_2(x) \tag{16}$$
$$... \tag{17}$$
$$y_t = y_{t-1} + \epsilon f_t(x) \tag{18}$$
$$\tag{19}$$

Where ϵ is the learning rate parameter.

5.3.2 Hyperparameter selection

Firstly, we try to use grid search with cross validation to find the best combinations of iteration parameters such as learning rate (how far a step when gradient decent) and n_estimators (number of iterations). We search learning rate in [0.1, 0.2, 0.3], and search n_estimators in [300, 400, 500]. The comparison is shown in Table.7 and Figure.16. We can find that when learning rate equal to 0.2 and 0.3, when iteration increased after 300, there is a overfit, as well as learning rate equal to 0.1 after iteration 400. The reason why we think it is a overfitting is that we fix learning rate when we try to search a good iteration parameter, and we can find there is a clear peak performance at 400 iterations. One thing important is that the training AUC is still increasing that is the other reason we think it is a overfitting. And we can find that the best model is learning rate equal to 0.1 and iteration equal to 400.

Secondly, we will tune the tree related parameters (max depth for a single tree and number of

Table 7: n_estimater and learning rate

n_estimator\learning rate	0.1	0.2	0.3
300	0.97955	0.97917	0.97774
400	0.97994	0.97865	0.97699
500	0.97935	0.97848	0.97601

leaves for a single tree). We search max depth in [3, 5, 7] and number of leaves in [3, 5, 7, 10]. The comparison is shown in Table.9 and Figure.17. We can find that when number of leaves bigger

Table 8: training AUC for n_estimater and learning rate

n_estimator\learning rate	0.1	0.2	0.3
300	0.98734	0.98767	0.98790
400	0.98841	0.98860	0.98874
500	0.98875	0.98884	0.98897

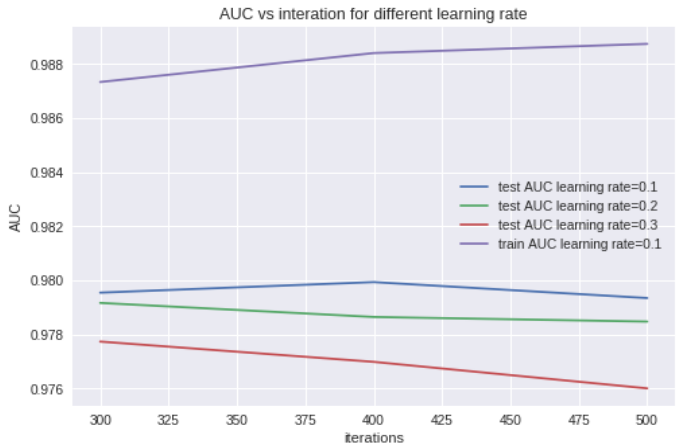

Figure 16: Learning rate and iterations

than 5 there is a overfit, and depth bigger than 5 also will overfit this problem. After tuning we can find max depth equal to 3, number of leaves equal to 5 will have best performance.

Finally, we will tune the regularization parameters for l1 and l2 regularization term. As shown in

Table 9: max depth and number of leaves

number of leaves\max depth	3	5	7
3	0.97979	0.97987	0.97998
5	0.98006	0.98001	0.98001
7	0.97983	0.97995	0.97969
10	0.97969	0.97944	0.97953

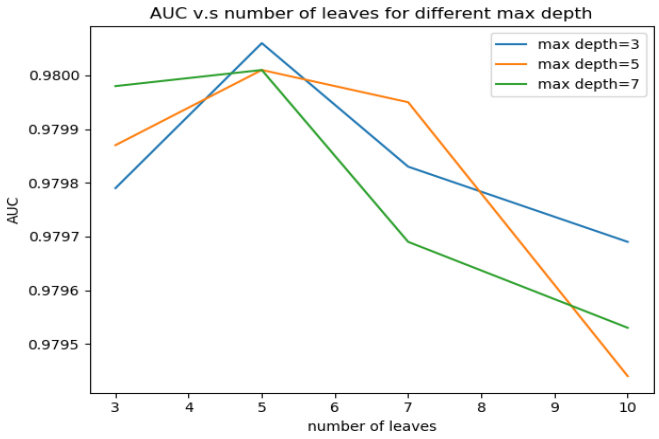

Figure 17: Max depth and number of leaves

Equation.10, γ and λ are the regularization term. We search l1 and l2 in [0.9 0.8 0.7]. The result is shown in Table.10 and Figure.18. We can find that when both γ and λ are low, this model will over fit, and we can find the best parameters are 0.9 and 0.9.

Table 10: Gamma and Lambda

lambda\gamma	0.7	0.8	0.9
0.7	0.97999	0.98000	0.97992
0.8	0.97993	0.97994	0.97994
0.9	0.97994	0.97992	0.98006

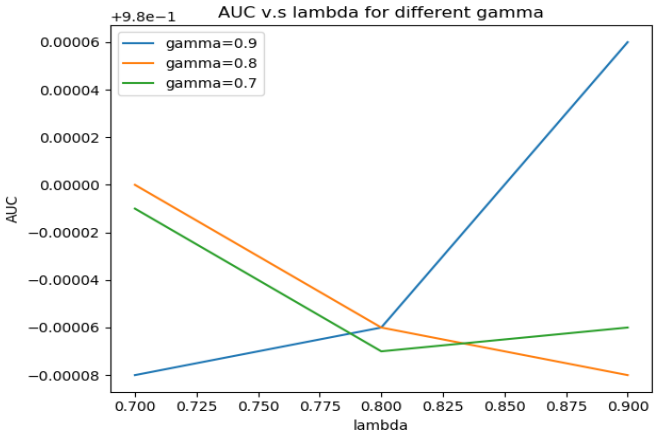

Figure 18: gamma and lambda

5.4 Result and Discussion

After solving the imbalanced problem and tuning the hyperparameter, we did a 5-folds cross-validation to check the AUC of our model. The Table.11 and Figure.19 shows the final results of 5-folds cross-validation test. We can find that the mean AUC is 0.981993, it is higher than

Table 11: Results on CV sets

	cross1	cross2	cross3	cross4	cross5	mean	std
AUC	0.980243	0.983421	0.982312	0.982341	0.981647	0.981993	0.001043
deviation	0.001750	0.001428	0.000319	0.000348	0.000346		

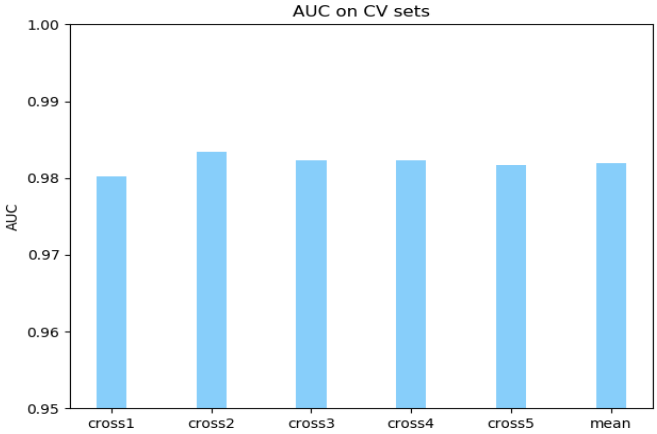

Figure 19: Results on CV sets

Logistic Regression, Random Forest and imbalanced LGB. The reason for the improvement is that this method solved the imbalanced data problem, have non-linear property, and have good model complexity to fit this problem. On the other hand, its deviation is really small, the standard deviation of 5-folds cross validation is only 0.001043, and have similar performance on 5 different folds. Thus, it is a generalized model for this problem.

In conclusion, we find GDBT model have a really good performance with low deviation. It is really suit for solving non-linear, dense feature problem. And because of the boosting structure, it has good performance on imbalanced problem, furthermore, if we use weighted loss function to improve it, it will have perfect performance on classification problem. Also due to the residual learning design, when using this model on different data set, it will have generalized performance with really low deviation. Thus it is a better model than we tried before such as logistic regression and random forest.
However, there is still a problem that if we only use one model as a solution like boosting model we will have high variance if we use this model on other testing sets. Thus, in order to have a generalized model, we should try to ensemble these models together to build a finalized model. Therefore, in next section, we will try to use different method to ensemble different models.

6 Ensemble

In last section, we have a strong improvement compared to simple model like Logistic Regression. But, one single model can still result in high variance which means a single outlier point (in this problem may be some sneaky fraudulent) will highly affect the prediction. Thus in this section we will try to use different method to ensemble models, which means we can use these methods to make our final model be less noisy and robust to outliers. Firstly, we will try to use voting and stacking scheme to improve simple weak model, then we will try to blend strong model like LGB with weak model to improve. Finally, we will show the results and try to discuss and explain the performance.

6.1 Hard Voting

For one data set, there may be some points that different models will have different predicted label. In order to reduce the variance to improve the accuracy, one possible solution is using majority vote. Let $C_i(x)$ denotes ith model and it will return 0 or 1 as predicted label. We can define the model as:

$$\hat{y} = mode\{C_1(x), C_2(x), ..., C_i(x)\} \tag{20}$$

In our experiment, we use logistic regression, random forest and decision tree as base model, in Table.12 we can find the AUC for every single base model on 3-fold cross-validation set.
After using hard voting scheme, we can find the results on same cross-validation sets in Table.13.

Table 12: AUC of base model

	cross1	cross2	cross3	mean
Logistic Regression	0.912322	0.903238	0.902876	0.906145
Random Forest	0.921242	0.922131	0.919421	0.920931
Decision Tree	0.917764	0.915323	0.918732	0.917273

We can find that on every cross set and mean AUC hard voting scheme is much better than Logistic Regression and Decision Tree. It also have 0.1 improvement than Random Forest.
The reason why it is better than single model is that this scheme can contest outliers and can

Table 13: Hard Voting AUC

	cross1	cross2	cross3	mean
Hard voting	0.928585	0.928343	0.930232	0.929053

reduce the variance of single model. But this scheme has problem that we trained our model on imbalanced data set thus predicted label will have some inaccuracies.

6.2 Soft Voting

To solve the problem of hard voting, we try to use some probabilistic model which can predict $P(1|x)$ and $P(0|x)$ such as Logistic Regression with soft voting to build our model. And we use $P_i^+(x)$ denotes ith model predicted probability for positive label of a point and $P_i^-(x)$ denotes ith

model predicted probability for negative label of a point. Then soft voting scheme can defined as below:

$$p(1|x) = \frac{\sum_i P_i^+(x)}{n} \tag{21}$$

$$p(0|x) = \frac{\sum_i P_i^-(x)}{n} \tag{22}$$

$$\hat{y} = argmax_{label}[p(1|x), p(0|x)] \tag{23}$$

Furthermore, we find random forest has best performance among these 3 models, thus we try to ensemble models based on random forest which means we can let random forest have higher weight. We can rewrite equation as:

$$p(1|x) = \frac{\sum_i w_i P_i^+(x)}{n} \tag{24}$$

Using same base model as last part, we have the results shown in Table.14 where w_i means weight for LR, RF and DT.

Compared to Table.12 and Table.13, we can find the soft voting scheme shows better performance

Table 14: Soft voting

Soft voting	cross1	cross2	cross3	mean
w=[1,1,1]	0.929442	0.928754	0.928432	0.928876
w=[1,2,1]	0.930230	0.929452	0.932395	0.930692
w=[1,3,1]	0.930723	0.934482	0.934218	0.933141

when changed to a good weight set. And we can find that, when we give random forest higher weight, it will give good performance. The reason is that soft voting is based on probability for each label, which means it can have high ability to fit different point including outliers. On the other hand, random forest has higher model complexity and higher accuracy than the two other models. Thus when it is given higher weight, the soft voting model will have higher accuracy.

6.3 Stacking

The following two schemes are all based on the idea of using weighted average on predicted label to improve models. However, weight may not be the only solution. When we fit our base model, it generates predictions for each sample. We can then use these predictions as a label to form a new classification problem, and apply another model to fit this new problem. Based on this idea, we designed a stacking model.

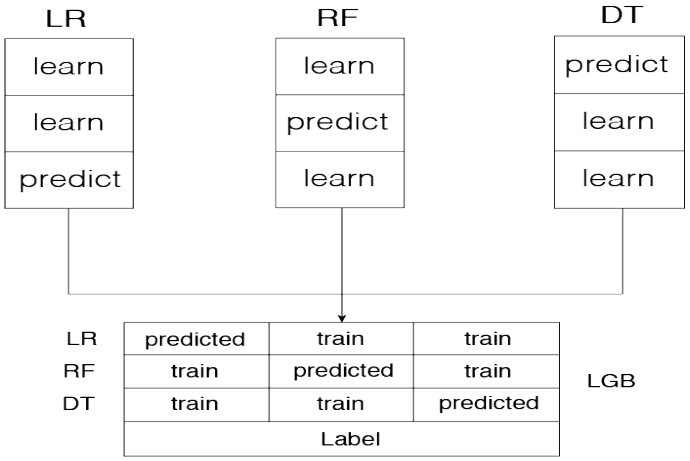

Figure 20: Stack Model

The model stacks the predictions of three models, and then use the outcome as input for a meta-model that produces final predictions. Firstly, split training data into 3 folds with every base

model using 2 folds to train and one fold to predict. Secondly, concatenate predicted labels and two fold original labels as a feature, thus we can get 3 features from the 3 base models. Finally, use another meta-model to solve this new classification problem. The structure is illustrated in Figure.20 to help understand the model.

Table 15: Stacking

	cross1	cross2	cross3	mean
Stacking	0.949573	0.950342	0.951347	0.950420

Using this scheme, we get AUC results shown in Table.15. By comparing the results with Table.13 and Table.14, we can observe considerable improvement in performance with this scheme. The possible reason of improvement is that it uses a non-linear model to fit the weight while voting scheme provides only a rigid choice with majority votes. LGB itself is a strong model compared to the three base models, and we use different bagging data to train base models which gives it strong randomness. However, the AUC score is lower than single LGB model we mentioned in last section, thus we want to figure out the reason why it is worse than a single LGB model and how to improve it.

6.4 Strong model ensemble

In this section, in order to find the reason why the stacking model using LGB as meta-model is worse than single LGB model, we try to use LGB as a base model with RF and DT, and use LR as meta-model. We get the results shown in Table.16.

Table 16: Stacking with LGB

	cross1	cross2	cross3	mean
Stacking with LGB	0.948453	0.946435	0.949872	0.948253

And we can find that the result is slightly worse than stacking three weak classifiers and much worse than single LGB model. Thus we think the possible reason is that LGB, DT, RF are all tree-based algorithms, and LGB has much higher accuracy than two other classifiers, it is possible that the correct predictions of DT and RF are a subset of the correct predictions of LGB, rather than intersecting each other. As a result, therefore this stacking scheme will not have better performance than the best model. In order to improve LGB model with stacking scheme, we should construct other generally different models which have similar performance as LGB, then use a simple meta-model such as Logistic Regression to stack them together to improve it.

7 Conclusion

In this project, we first pre-process the data by performing cumulative encoding on categorical data, constructing new features, and perform log normalization if needed. Then we applied the logistic regression, random forest and boosting algorithm to the dataset separately. We discover that logistic regression model is of low complexity to fit the dataset. Although the random forest can be made complex enough, it can not overcome bias factors like noise by iterating the trees. Given that condition, we turn to the boosting algorithm which is considered as more robust to noise when the dataset is large.

To further, we try to use other ensemble methods such as average voting, stacking to improve simple models like Logistic regression and decision tree. We found that when we applied these strategies and weighted every models we can get higher AUC results. However, we found that when we try to blend LGB which means using LGB as base model in stacking we get lower AUC results. We think the possible reason is that accuracy of other models is much lower than LGB, thus the stacking model with LGB was not improving AUC results.

For the further improvement, the most intuitive methods is to train the model on the whole dataset provided by Talkingdata rather than only small portion of it. In general, larger dataset results in better generalization. Due to the constrain on computation resources of our personal

computer, we prepared to train the dataset on cloud such as Google Cloud Platform, AWS, etc. In addition, we consider construing new feature is the key as more carefully designed features. This is because good feature selection and construction gives more flexibility to the choice of model as good features can be discriminated by relatively less complex models.

8 Appendix: Source code list

`LogisticRegression.py` contains the source code of logistic regression model along with the grid search code for regularization term C. `grid_search_for_depth_of_tree.ipynb` contains the source code performing the gird search for parameter `depth` of random forest along with the results.
`grid_search_for_num_trees.ipynb` contains the source code performing the gird search for parameter `n\_estimator` of random forest along with the results.
`data.py` contains the source code of feature engineering and data preprocessing.
`LGB.py` contains the source code of single LGB model.
`test.py` contains the source code of grid search for LGB model.
`ensemble.py` contains the source code of hard-voting and soft-voting model.
`stacking.py` contains the source code of stacking model.

References

[1] Wikipedia. Click fraud. https://en.wikipedia.org/wiki/Click_fraud.

[2] Kaggle Inc. Talkingdata adtracking fraud detection challenge can you detect fraudulent click traffic for mobile app ads? https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data, 2018.

[3] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[4] pandas 0.22.0 documentation:api reference pandas.dataframe ». https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.corr.html, 2018.

[5] CMU Course Notes. Logistic regression. http://www.stat.cmu.edu/ cshalizi/uADA/12/lectures/ch12.pdf.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[7] Llew Mason, Jonathan Baxter, Peter L Bartlett, and Marcus R Frean. Boosting algorithms as gradient descent. In *Advances in neural information processing systems*, pages 512–518, 2000.

[8] Finley T et al Ke G, Meng Q. Lightgbm: A highly efficient gradient boosting decision tree[c]. *dvances in Neural Information Processing Systems*, pages 3149–3157, 2017.