Ad Click Through Rate Prediction

Executive Summary:

The objective of our machine learning project is to attempt to predict the probability of clicking on an online advertisement. We are using a dataset with 24 columns of data, with one dependent variable — "click." This categorical variable outputs a 1 if the advertisement is clicked and a 0 if the advertisement is not clicked. The training data that we used to create the models has over 30 million records of data, and the testing data has about 13 million records of data. The project consists of many different steps to optimize. To begin, we did exploratory data analysis to understand the data, data cleaning to manage the variables more effectively, and then moved to model building and model evaluation. In our model building, we explored the parameter tuning and LogLoss validation from different models, including logistic regression, random forest, neural network architectures, and 2 more advanced boosting methods, LightGBM and CatBoost. After the technical analysis, we found that our approach using a Gradient Boosting Decision Tree produced the best on the validation dataset that we generated, with the smallest log loss value. Thus, we decide to choose LightGBM as our final predictive model.

Data Preprocessing

We first import the raw dataset from csv file to the Python pandas data frame. To fulfill data exploration and data cleaning in a more efficient manner in Python, the first step we took was to shuffle through the data and split the data into 10 parts randomly. After that, we decided to get rid of the id column since this column is not necessary to provide information to the predictive models. Next, we process the data cleaning, analyzing the 22

categorical variables provided to build the predictive models. We wanted to make sure that none of the categorical variables was highly skewed, and if they were skewed, we managed to transform them. We decided to create a function to transform the skewed categorical value based on frequency, where we replace a category when its frequency is lower than a threshold. We then applied this to heavily skewed categories including site_id, site_domain, app_id, app_domain, app_category, device_id, device_ip, device_model, c14, c17, c19, c20, and c21. Then, we decided to drop the device_ip attribute since it is too skewed to transform into an informable variable.

The next step we did was encoding the still diverse categories. Encoding methods that we used include one-hot encoding, base 5 encodings, and base 10 encodings. Finally, we separate the numerical variables “hour” into 2 different numeric variables, “time of day” and “day of the week”.

At this point, both the training and test data were ready to be used for predictive modeling. Next, we will explain the technical details of categorical transformation and numerical engineering of the dataset.

Feature Engineering

Categorical Data Encoding

In order to train the most efficient machine learning model and minimize the outcome of the LogLoss function, we decided to check the skewness of each column and transform skewed categorical columns based on their frequency. Once the column has high skewness, we will reduce the complexity of the column. The first way we transform the categorical data calls ‘frequency encoding’ by creating a function to transform skewed

categorical values based on frequency, which means once the ratio of a specific category is less than a certain percentage, default 1%, we will replace it by the category name 'other'. (See code chunk below)

```
# define a function to transform skewed categorical value based on frequency
def categorical_replace(train_data, test_data, column, pct = 0.01):
    """
    train_data: train dataset to input
    test_data: test to input
    column: column name string to input
    pct: transform frequency treshold, default 0.01
    """
    cond = train_data[column].value_counts(normalize = True) > pct
    non_others = cond[cond].index  # define a list to save main category

    train_data['temp'] = 'other'
    train_data.loc[train_data[column].isin(non_others),'temp'] = train_data[column]
    train_data[column] = train_data['temp'].values
    del train_data['temp']
    print("Train Test Replace Finished!")

    test_data['temp'] = 'other'
    test_data.loc[test_data[column].isin(non_others),'temp'] = test_data[column]
    test_data[column] = test_data['temp'].values
    del test_data['temp']
    print("Test data replace finished!")
```

However, after the transformation, we realize the number of unique values in each column is still pretty high. Hence, we use the other 2 methods called 'Based N Encoding' and 'One Hot Encoding' to reduce the complexity of the dataset further. For Based N Encoding, we transform the total number of variables into several columns, where each column is numbered by a range of numbers from 0 to (N-1). It allows us to convert the integers with any value of the base, which is extremely helpful especially when the number of categories is so large to merely label with numbers. For one hot encoding, we create vectors containing 1 and 0 and map each category to the vector; the number of vectors depends on the number of categorical data on each column. We transform low-number

categories with the method, where the loss of information is also the lowest for those categories. See the Table below for a summary of the categorical variable transformations.

Column Name	Original unique values	Original skewness	Transform unique value	Transform skewness	Value of N encoding
site categories:					
site_id	3313	44.16	9	1.22	5
site_domain	4036	48.25	9	1.22	5
site_category	23	2.55	Does not transform		10
app cats:					
app_id	4599	66.84	9	2.56	5
app_domain	289	16.14	4	1.89	One-hot
app_category	29	4.59	Does not transform		10
device cats:					
device_id	418031	646.55	3	1.41	One-hot
device_ip	1348033	250.65	This column is too skew, drop this.		
device_model	6198	26.05	15	3.83	10
Category without names:					
C1	7	2.62	Keep the same		
C14	2299	8.012	23	4.78	5
C15	8	2.80	Keep the same		
C16	9	2.98	Keep the same		
C17	404	10.18	24	4.48	5

C18	4	-0.13	Keep the same		
C19	66	5.29	16	2.17	5
C20	166	11.80	16	3.56	5
C21	55	3.61	18	1.95	5

Table-1 Variables Transformations

Numerical Variable

For numerical data, we transform the 'hour' column from timestamp into readable time and divide it into 2 columns, which represent day (contains the year, month and day) and time respectively. To further reduce the complexity of the data, we convert the time column into 4 different variables: set 00 AM - 06 AM as '1', 07 AM-12 PM as '2' … to the number 4. We also transfer the day of the week column into numbers, eg. set Monday as '1', and Tuesday as '2', etc., which makes it easier to read and train the model.

Model Building

As mentioned above, the problem we're trying to solve is a classification problem, where we need to predict the True or False as well as the probability of the click rate of ads. We started by picking different methods and comparing their general performance before we dug into parameter tuning.

We then choose to implement the following 5 models: Logistic Regression, Random Forest, Neural Net, LightGBM (short for the light gradient-boosting machine), and CatBoost. The logic behind choosing these 5 models is we want to apply different techniques apart from tree-based algorithms. We start off by using Logistic Regression as a baseline for all other

models. Then we implement the Random Forest model. We jump over the Decision Tree model and go straight to Random Forest because Random Forest would generalize the data in a better way than a simple decision tree model since the output of Random Forest is the random ensemble results from multiple trees. Also, with randomization introduced to the model, the trees can generate more information with the many categorical variables in our dataset and reduce the effect of potential outliers in our categories. Although we understand that Random Forest will be much more computationally intensive than a decision tree model, especially given the size of the dataset, we believe those advantages of random forest are worth the wait.

Besides Random Forest, we also run experiments on 2 other tree-based algorithms: LightGBM and CatBoost. LightGBM is a gradient boosting framework that uses tree-based learning algorithms, it is designed to be distributed and efficient with advantages such as lower memory usage, faster training speed, and the capability of handling large-scale data. Using gradient-based one-side sampling, Lightgbm carries a leaf-wise growth while training and is much faster and compatible with our large and complex datasets.

As for CatBoost, it's also a gradient-boosting technique based on decision trees. Its most worth-mentioning feature is that it can understand and has a great advantage of dealing with categorical features. Since our dataset has a large number of categorical features, we figure this might be a good fit. Besides, the model boosts itself automatically, making it more flexible to different data sets and less loaded to parameter tuning.

After finalizing what model we are going to use, we fit models with one of the splits of our training data (1/10 as mentioned above) and start the parameter tuning. We mainly focus on tuning LightGBM and Random Forest since they perform the best after our first tune.

Before tuning comprehensively into LightGBM and Random Forest, we also manually tuned parameters for Neural Net and logistic regression, such as type of penalty, number of units, weights of kernel regularizer, number of nodes dropped off, adding kernel initializer and activation function for the hidden layer. Unfortunately, none of these changes improved model performances greatly, and we decided to stick to LightGBM and Random Forest.

During the tuning of RandomForest, we use TuneGridSearchCV from the tune_sklearn package to select the best parameters. This method uses Bayesian Optimization, and it significantly reduces process time from over 500 minutes when using normal GridSearchCV to only 40 minutes. In RandomForest, we are looking for the best combination of max depth, the minimum number of samples required to split an internal node, a criterion to split a node, the number of trees we want to build, and the number of features to take into account the best split. Random Forest with a "max_depth" of 20, "min_samples_split" of 100, and "n_estimators" equals 10 works the best.

For LightGBM, we use GridSearchCV to search over the learning rate and the number of leaves. In the end, LightGBM with the learning rate of 0.1 and the number of leaves of 300 works the best.

Final results

After hours of tuning, here are our final LogLoss results of each tuned model (**Table-2**) .

Model	Logistic Regression	Random Forest	Neural Net	LightGBM	CatBoost
Log-loss score	0.436	0.404	0.455	0.401	0.402

Table-2 LogLoss of Tuned Classification Models

Our best performing LightGBM model has an average AUC of 0.73 (Fig-1), and log loss score of

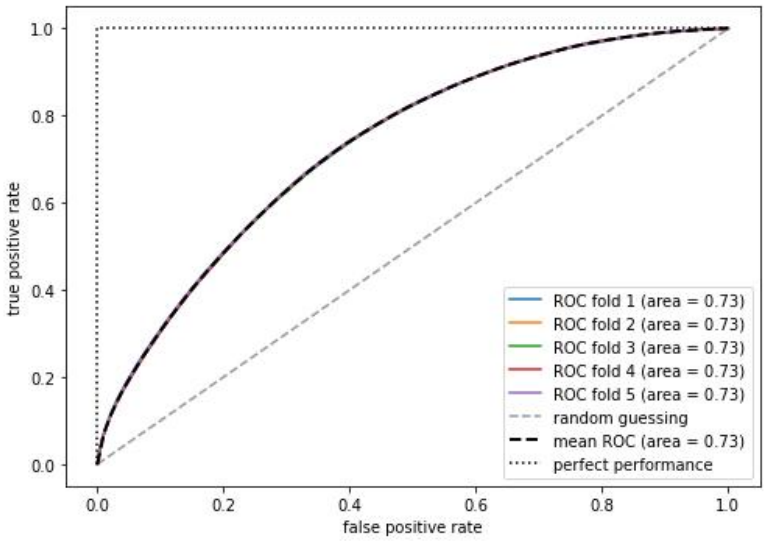

Figure-1 ROC Curve of LightGBM

0.401, which outperforms our best Random Forest model, which has a log loss score of 0.404. Therefore we use LightGBM as our final model. The plot of feature importance for the LightGBM model shows that timing is really important when it comes to whether to click ads. Days_of_week is the most important feature, and time_of_day is the 3rd important feature. In addition, device model and site_category are also important factors that affect click rate. This indicates that the marketing department needs to consider different ads push strategies for iOS devices, Windows, or Android devices. This also applies to ads on different kinds of sites such as information sites or entertainment sites.

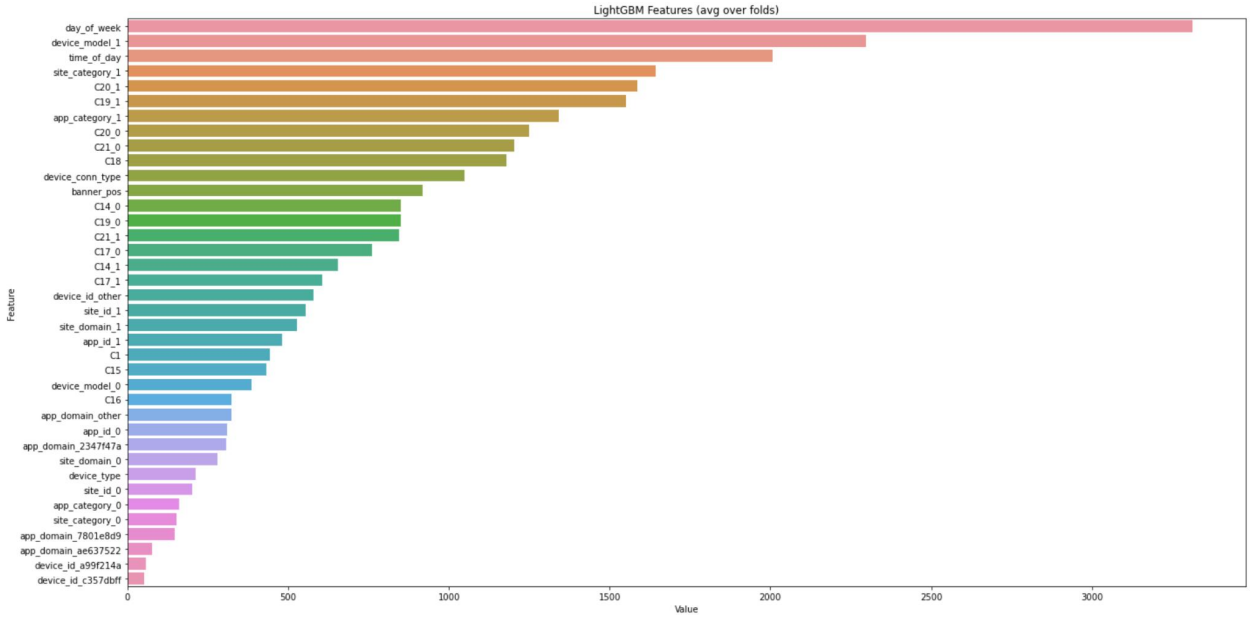
LightGBM Features (avg over folds)
day_of_week
device_model_1
time_of_day
site_category_1
C20_1
C19_1
app_category_1
C20_0
C21_0
C18
device_conn_type
banner_pos
C14_0
C19_0
C21_1
C17_0
C14_1
C17_1
device_id_other
site_id_1
site_domain_1
app_id_1
C1
C15
device_model_0
C16
app_domain_other
app_id_0
app_domain_2347f47a
site_domain_0
device_type
site_id_0
app_category_0
site_category_0
app_domain_7801e8d9
app_domain_ae637522
device_id_a99f214a
device_id_c357dbff
Feature
0
500
1000
1500
2000
2500
3000
Value

Figure-2