Prediction on On-line Advertisement Clicks

ISOM 674 Final Project

Team 10
AJ Sonawala | Danielle Zhao | May Shao | Russel Shen

1. Introduction

"Half the money I spend on advertising is wasted; the trouble is I don't know which half." The saying from John Wanamaker discloses the significance of advertisement launch precision, which is getting even more remarkable in our age of online marketing given the large volume of advertisements being published and huge costs associated with the marketing campaigns. Our project intends to unravel the mystery of targeted marketing by predicting the clicks for on-line advertisements using data for 9 days from October 21 to October 29 in 2014, during which the reactions to more than 30 million instances of online advertisements were recorded.

2. Initial Data Exploration

The entire training dataset has 31,991,090 records (many of which are duplicate rows). It contains 24 variables and no missing values. Given that all of our variables are categorical, we want to first understand the number of levels and distributions of each variable in the training dataset: Click is our binary dependent variable that contains 17.0% of value 1 (ad was clicked on) and 83.0% of value 0 (ad was not clicked on). Its classes are relatively imbalanced, but the imbalance represents the true proportion of the classes, so we didn't rebalance the data. In addition, the number of unique values of all variables are shown below.

Variable Name	Number of unique values	Variable Name	Number of unique values
id	31,991,090	C19	66
device_ip	5,762,925	C21	55
device_id	2,296,165	app_category	36
app_id	8,088	site_category	26
device_model	8,058	C16	9
site_domain	7,341	C15	8
site_id	4,581	C1	7
C14	2,465	banner_pos	7
app_domain	526	device_type	5
C17	407	device_conn_type	4
hour	216	C18	4
C20	171		

Table 2-1

We also examined the summary statistics and distributions of each variable:

variable	#var	n	min	max
id	1	3.20E+07	Inf	NA
click	2	3.20E+07	0	1
hour	3	3.20E+07	14102100	14102923
C1	4	3.20E+07	1001	1012
banner_pos	5	3.20E+07	0	7
site_id	6	3.20E+07	Inf	NA
site_domain	7	3.20E+07	Inf	NA
site_category	8	3.20E+07	Inf	NA
app_id	9	3.20E+07	Inf	NA
app_domain	10	3.20E+07	Inf	NA
app_category	11	3.20E+07	Inf	NA
device_id	12	3.20E+07	Inf	NA
device_ip	13	3.20E+07	Inf	NA
device_model	14	3.20E+07	Inf	NA
device_type	15	3.20E+07	0	5
device_conn_type	16	3.20E+07	0	5
C14	17	3.20E+07	375	23836
C15	18	3.20E+07	120	1024
C16	19	3.20E+07	20	1024
C17	20	3.20E+07	112	2729
C18	21	3.20E+07	0	3
C19	22	3.20E+07	33	1959
C20	23	3.20E+07	-1	100248
C21	24	3.20E+07	1	253

Table 2-2

The distribution of C14 is skewed to the right, while the distribution for C21 is skewed to the left.

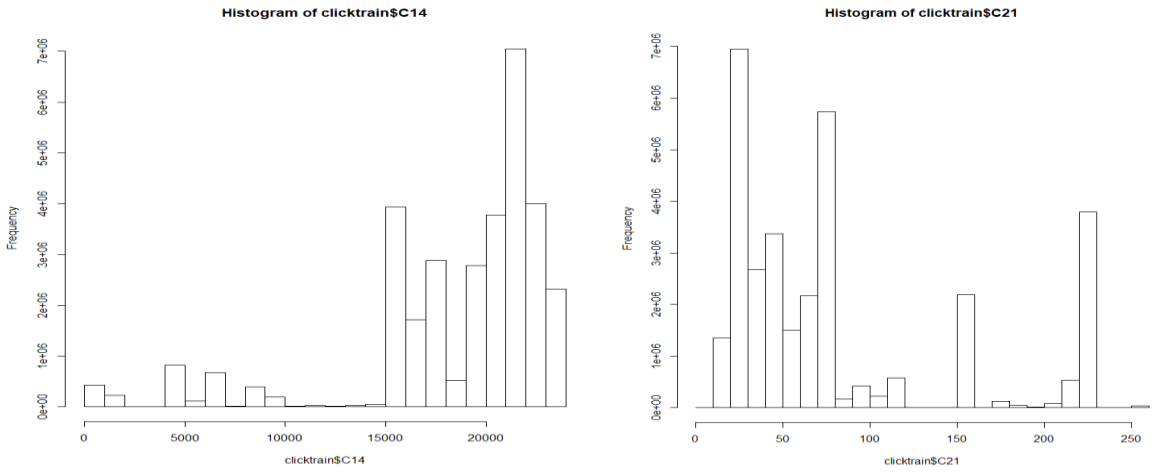

Figure 2-1

We also looked at the correlations among variables from one smaller training subset of one million records. What is worth noticing is that C17 is highly correlated with C14 ($\rho = 0.976$).

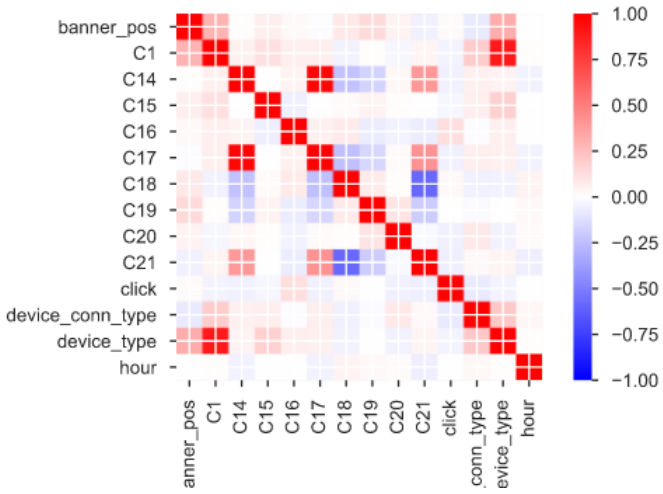

Figure 2-2

From the correlation map we can see several pairs of C variables are highly correlated, including C14 and C17, C18 and C21. C16 has the highest correlation with our target variable click.

3. Data Preparation / Feature Engineering

To prepare our data, we first split the training dataset into around 30 small datasets, each having one million observations. The reason we split the full training set is that the computational cost on the full dataset is too large and each one-million dataset is good enough for the machine learning models. We used Linux Commands *split* and *shuffle* to obtain the smaller datasets needed. We chose three datasets as the training, validation and test sets.

After conducting exploratory data analysis on the full dataset, we realized the *hour* variable is a feature consisting of year, month, day and hour. We decided to transform it to show only the hour by dividing that variable by 100 and getting the remainder, as we believed the time period of a day would influence whether people click on the advertisement or not.

We also noticed that there are too many unique values in each variable so that one hot encoding would result into high dimensionality, which increases the model's training time and memory consumption. In order to reduce the dimensionality, we tried to select meaningful variables from all the variables and grouped the unique values that have low frequency in each variable if that variable has too many levels. However, this data transformation did not generate a good performance on the final model, so that we resorted to another encoding method called hashing. Hashing maps the data of arbitrary size to fixed-size values. To avoid collision between observations but controlling the dimensionality, we set the number of features after hashing as 10,000 through trial and error. In other words, each of the split datasets will consist of 10,000 columns after hashing. After completing feature engineering, the shape of each dataset is 1,000,000 by 10,000.

4. Model Pre-selection

Since the goal of this project is to predict the probability of clicks for on-line advertisements, we pre-selected the following five models which are common and powerful for classification problems:

- Logistic regression
- Ensemble method (based on logistic regression)
- Random forests
- Neural network
- LightGBM

5. Modeling Approach

a) Logistic Regression

We used the LogisticRegression function in *sklearn.linear_model* package to build up the logistic regression model. As for the parameter tuning, we chose to tune the parameter *penalty* and *C*, which represent the penalization method used in the model and the inverse of regularization strength respectively. We created two for loops to iterate through the different *C* parameters using either L1

or L2 penalization method. The reason we did not use grid search is that it is too time-consuming. The models were built up on the training set and tested on the validation set.

After looking at the different values of parameters, we picked the model with the smallest log loss value of 0.4018, where C = 0.1 and *penalty* = 'l1'. To get the generalized result, we tested the performance of the best logistic regression mode on a testing set and got a log-loss of 0.403.

b) Ensemble Method

After using logistic regression based on one training dataset, we wanted to add three more training datasets to fit another three models and come up with an ensemble method. For each single model, we used the best parameters we tried: C = 0.1 and *penalty* = l1. Then we built up four logistic regression models using these four datasets and did the predictions based on the test data set. Finally we combined four models together into one ensemble model by averaging their predictions and generated the log-loss of 0.4065.

We chose to use ensemble method because we want to eliminate the influence of one specific training dataset since each data set only contains 3.33% of the original train data set. Training the model using four different data sets and using the average performance based on four models can avoid the problem of overfitting on a specific train data set.

c) Random Forest

According to what we have learnt in class, the random forest model usually has better performance than classification tree or bagging. This is because the random forest model considers multiple ways of sampling both on features and on datasets. However, since the random forest model's computation process is way too time-consuming, we didn't use grid search or for loop to choose the best parameters. Instead, we decided to base this on our experience to find the relatively better parameters.

We used the RandomForestClassifier function in *sklearn.linear_model* package to build up the random forest model. We made two trails and set the parameter: n_estimators as 10 and 100.

N_estimators is the number of trees in the forest. N_estimators is the number of trees in the forest and in this case, when n_estimator= 100, we will have a better result. In order to compare the performance with other models, we ran the model on the test dataset and the log-loss is 0.4223.

d) Neural Network

For neural network, we first converted X and Y variables into numpy array. As numpy array does not support 10,000 columns as in sparse matrix, we changed the *n_features* parameter in hashing function into 500. Then we rescaled the X variables for both training and validation sets into 0-1 as it can make the model much more efficient and performance better. Then we set the neural net parameters, in which we set epoch as 500 and BatchSize as 250. Regarding the neural network layers, we tried different layers and activation functions by changing them manually as it took too much time to run a for loop to tune the layers. We fitted the neural net models using training set and predicted on the validation set. The best model of neural network consists of four layers of 4 ReLU nodes and one final layer of 1 node using softmax activation function. To get the generalize result, we tested the model on another testing dataset and got a log-loss of 0.4557.

e) LightGBM

Firstly, we converted X and Y into numpy arrays. Same as in neural network, we changed the *n_features* parameter in hashing function into 500. Then we converted Y variables into series as it's the input format for the model. We did the feature scaling on X variables using StandardScaler function. It's worth noting that while fitting the model, we put two datasets in the lightgbm.train function. The second dataset was for inner validation when training the model. If the validation score did not improve within a specific number of rounds, specified by early_stopping_rounds, then the model would stop iterating. Regarding parameter tuning, we mainly tuned the following parameter: *num_boost_round, num_leaves, learning rate* and *early_stopping_rounds.* The parameters of the best LightGBM model is as follows: *application: num_leaves: 0.5, learning_rate: 0.1, num_boost_round: 100, early_stopping_rounds: 100*. Finally we tested the model on a testing set and got the log-loss of 0.5900.

6. Conclusion

Model	Parameter	Log-Loss
Logistic Regression	Penalty='l1', C=0.1	0.4030
Ensemble Method (averaging 4 logistic regression models)	Penalty='l1', C=0.1	0.4065
Random Forest	n_estimators=100	0.4223
Neural Network	4 layers of 4 ReLU, one layer of softmax	0.4557
LightGBM	Num_leaves=31, learning_rate=0.1	0.5900

Table 6-1

Based on the analysis results on the testing dataset, we can see that the logistic regression using L1 penalization method provided the best performance. At the first glance, it seems surprising that other more complicated and robust models do not perform better. The reason is that in this case our dataset has a huge number of features, which causes the curse of dimensionality. Thus we need a simple model to avoid overfitting on the dataset. Logistic regression is way better than other models in terms of this problem. Moreover, the sparsity of the dataset makes tree models hard to find the right splitting point, which makes tree models perform worse than the simple logistic regression model.

7. Appendix

a) Data preparation using one hot encoder and principle component analysis

Before we decided on using hashing in data preparation, we firstly tried using one hot encoder to change every categorical variable into a dummy variable. One hot encoder uses a sparse matrix and includes all the variables, rather than dropping a level for each variable. As mentioned before, the disadvantage of using large amounts of dimensions is that it can cause the curse of dimensionality and a high correlation between variables. In order to solve the problem of too many dimensions, we chose 11 variables which have relevant fewer levels as the first step. Then we tried to make classifications based on each value's frequency firstly and then use one hot encoder to covert categorical variables into dummy variables. For example, we preserved some of these values and classified the other values into a new category: "others". For example, the variable *"app_category"*

has 36 different values, however, five of these values have already cover 99% of the data volume. Thus, for the other 31 values, we can group them all together as a new value called “others”.

Value	Frequency	Cumulative Percentage
07d7df22	21,101,482	65.96%
0f2161f8	7,487,678	89.37%
cef3e649	1,414,688	93.79%
8ded1f7a	1,020,683	96.98%
f95efa07	668,647	99.07%
…	…	…
cba0e20d	1	100.00%
f395a87f	1	100.00%

Table 7-1

However, even after we made preliminary classification to reduce the dimensionality, the datasets still have too many columns. We would like to use principle component analysis to reduce the dimensionality while preserving most important features at the same time. We can use scree plot to find out that as long as we keep the first 30 principle components, we can preserve 80% of the features.

The reason why we didn’t use this method are as follows:

- This data preparation method has too many steps and is time-consuming.
- We only chose 11 variables and other variables are neglected. However, the other variables such as app_id can also be influential in our prediction.
- The criteria we used for reclassification is too subjective. There are several anonymous variables and making classifications only based on frequency instead of business understanding is not reasonable.

b) Code file list

File Name	Content	Description
Project-Code1-Team10.py	Data preparation and Modelling	Hashing, five models and make predictions on test dataset
Project-Code2-Team10.R	Explanatory data analysis (EDA)	EDA of the complete training dataset
Project-Code3-Team10.ipynb	EDA training subset pandas profiling	EDA of the complete of a training subset
Project-Code4-Team10.html	EDA report	the pandas profiling report generated by Project-Code3-Team10

c) Reference:

1. https://booking.ai/dont-be-tricked-by-the-hashing-trick-192a6aae3087
2. https://towardsdatascience.com/all-about-categorical-variable-encoding-305f3361fd02