UCL Web Economics Algorithm Challenge 2016

https://github.com/blackwhitehere/webEcon/tree/master/wegroup

Group members:
Ji Zhou
Jiapeng Wang
Stanislaw Guner

Introduction

The goal of this project is to predict the user's click response to each auctioned ad impression in real-time bidding (RTB) display advertising. Specifically, given the information of the incoming bid request, the bid agent should estimate the probability that the user will click on its ad if it is displayed. Click-through rate (CTR) estimation is normally a regression problem where the label is a probability (of click). Several machine learning models are used for CTR estimation in this project.

1. Statistical Analysis

Both training and test data files are in a record-per-line format. Each line of the file contains the labels, click (1) or no click (0) and 23 ad features about users and advertisements. Variable details are listed in Table 1.

There are total 2,847,802 records in the training data and only 2,076 records showing that the web visitor clicked advertisement, accounts for 0.0729%.

Figure 1 shows that real-time bidding occurs more often at 18, 23 and 24 pm compared to other hours. This can be explained by the habit of wed visitors and at that time, people are free to surf websites probably after work and before going to bed. Diagram on the right side says that the weekdays do not really matter, degrees of seven days are close.

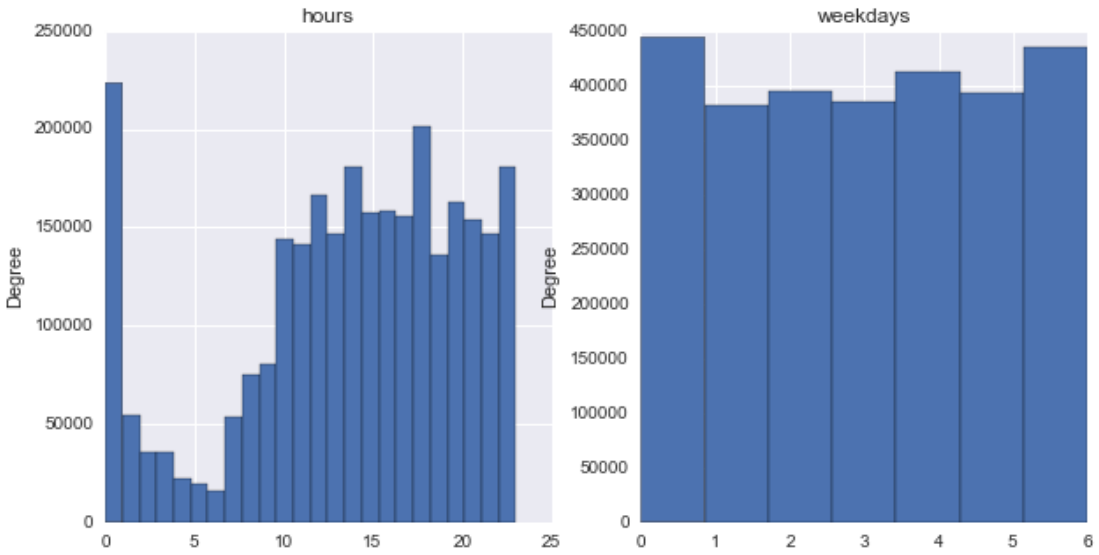

Figure 1: Histogram for 24 hours and 7 weekdays based on all records

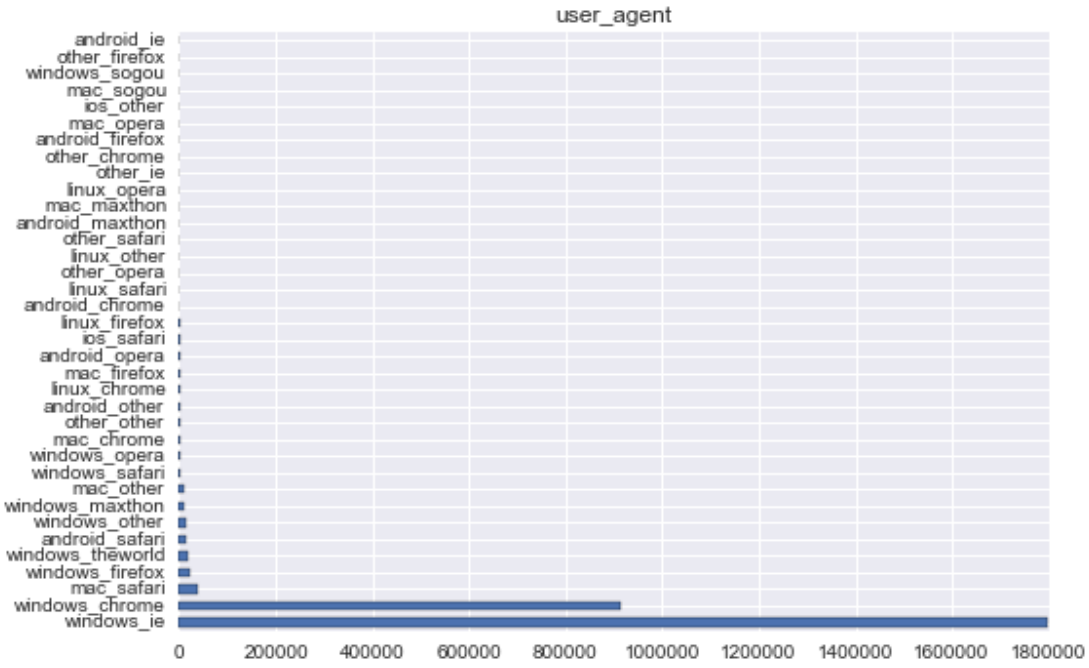

Figure 2: Histogram for *user_agent* based on all records

The analysis on *user_agent* (web browser), tells us that the operating system Windows is the most popular and the primary browsers are IE and Chrome followed by Mac and Safari. The records on *windows_ie* are twice as common as on *windows_chorme* and therefore, the advertiser could focus on targeting those users.

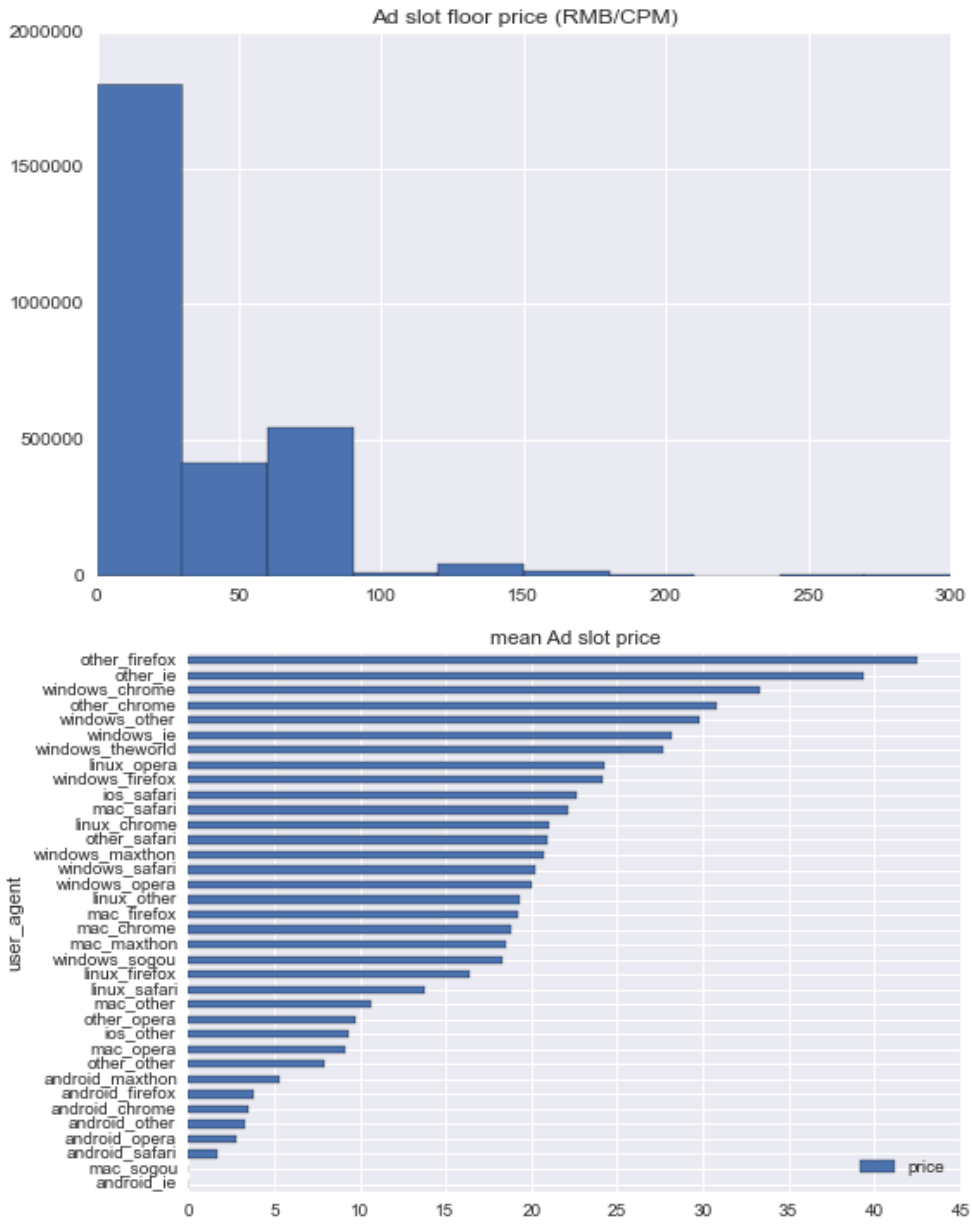

Figure 3: Top: Histogram for *Ad slot floor price* based on all records. Bottom: Bar plot for mean ad slot price by *user_agent*

From above analysis, we know that the users on Windows IE and Chrome counts the most and inevitably, the ad slots on those two browsers are hot and priced high. Shown by the bottom diagram on Figure 3, mean ad slot price for *windows_chrome* ranks the 4th and for *windows_ie* ranks the 6th.

Top histogram in Figure 3 shows the *Ad slot floor price*, in currency RBM per thousand impressions and most of them are less than ¥100. The most expensive ad slot sold for ¥300 while the mean is around ¥29.44.

2. Feature Engineering

A good understanding about the features and feature selection are quite important before applying machine models. Feature selection reduces the number of the features, which leads to better performing models, prevents over fitting and improves the generalization of models.

A few principles we followed for feature engineering:

[1] Delete absolute meaningless features, for examples, User ID: dF_5qwD1UDI, which is helpless to our prediction because character 'dF_5qwD1UDI' contains no information.

[2] Delete high-variation features as which makes clustering more difficult, for example, URL: hz55b000008e5a94ac18823d6f275121, there are total 715316 unique URLs and in fact it is meaningless as well.

[3] Delete zero-variation features under instructions of Principle Component Analysis (PCA), which says features should have variance as large as possible for maximum uniqueness in order to keep distances from each other as far as possible.

Table 1: Detailed feature engineering, column name in red is dropped

Columns	Name	Description
0	Click	[binary] prediction target
1	Weekday	[numerical] 1-7 indicating weekdays
2	Hour	[numerical] 0-23 indicating hours
3	Timestamp	[2] Each record has a unique timestamp, so the variation of this feature would be extremely large, as we already retain weekdays and hours, we'd like to drop Timestamp
4	Log Type	[3] We drop Log Type due to zero variance.
5	User ID	[1] Meaningless to the prediction
6	User-Agent	[character] includes windows_chrome, mac_safari and etc. total 36 unique User agents
7	IP	[1] Meaningless to the prediction
8	Region	[numerical] total 35 unique regions
9	City	[numerical] total 370 unique cities
10	Ad Exchange	[numerical] total 3 Ad exchanges
11	Domain	[character] total 15146 unique domains
12	URL	[2] Drop URL because of high variance
13	Anonymous URL ID	[3] Drop Anonymous URL ID because all of it is null
14	Ad slot ID	[character] total 51529 unique Ad slot IDs
15	Ad slot width	[numerical] examples: 300, 1000, 728 and etc.
16	Ad slot height	[numerical] examples: 90,250,280 and etc.
17	Ad slot visibility	[numerical] examples: 0,1,2,255
18	Ad slot format	[numerical] examples: 0,1,5
19	Ad slot floor price	[numerical] price in RMB currency per CPM (cost per

	(RMB/CPM)	thousand impressions)
20	Creative ID	[character] total 11 unique creative IDs
21	Key Page URL	[character] total 2 unique Key Page URLs
22	Advertiser ID	[3] Drop Advertiser ID because of zero variance
23	User Tags	[2] Series of tags for each record, total 720665 unique tags

Some id features (key_page_url, creative_id) and categorical features (ad_exchange, format) have few levels and so can be converted from strings to integers. This approach is preferred to OneHot encoding due to large storage requirement of OneHot encoder (especially on such a large dataset). The classifier we use utilizes multiple trees and the boosting methodology what mitigates the effect of arbitrary assignment of categorical variables to numbers.

Remaining "id features" are excluded from the dataset. Using id features would only be feasible if one would like to model individual behaviour of websites or individuals. We are however more interested into discovering general rules for behaviour that is characteristic to user-clicks.

It can be observed that *user_tags* column contains variable amount of comma delimited numbers that correspond to a record of user characteristics. As such, the *user_tags* field can be considered as a small document that describes an individual. This suggests an approach to feature engineering inspired by information retrieval theory. The idea is to quantify if the user tags are normal or not. In order to do so, tfidf vectorizer is applied on text in the user_tags column. This process creates a matrix of tfidf scores with rows representing the data records and columns listing all user tags that appeared in user_tags column. In order to reduce dimensionality of this matrix a Singular Value Decomposition is performed and matrix reconstruction with 5 dimensions corresponding to largest singular values is appended to the data matrix. Those 5 new column entries describe different dimensions of specificity of user tags associated with a record. This allows the classifier to learn (possibly) that when some unusual combination of user tags is recorded, then a click is more likely.

User_agent column in fact contains two different pieces of information: the operating system of the user and his/hers browser. It can be argued that the type of operating system and the browser used is correlated with the level of technical sophistication of a user [e.g. internet explorer user on windows (normal user) vs. firefox user on linux (developer)]. This quality in turn can be associated with use of ad blocking software and general willingness to click on internet ads. As such, we create an ordinal scale that e.g. places 'ios'/'ie' as 1 and 'linux'/'firefox' as a 5. We use those ordinal scales and remove the original *user_agent* column.

Timestamp feature contains date information that can be split into different columns. As a result, we obtain a column for day of month, month number in a year and year. Such encoding allows the classifier to learn seasonal patterns in click through rate.

With width and height features available it is natural to derive an “area” feature as the product of the two. This is motivated by the fact that larger ads will gather more attention from the user. We also tested a ratio of width to height with a hypothesis that a skewed ratio corresponds to exotic shape ads that receive different attention. We did not find this feature helpful, but the area is definitely a significant predictor of a click. Another feature is *cost_per_area* that quantifies the value of add space and so can be correlated with usefulness/appeal of the add.

3. Data processing

After dropping helpless features, we then have to change categorical data into numerical representatives using label encoder. For example, the datatype of *creative_id* is a string and therefore its 11 unique levels are transformed to numbers starting from 0.

Training set is split into a validation sample of 30% of observations and the remaining data is used to train the models. Continuous variables like width and height are scaled into range between 0 and 1 to align the scales between different features.

4. Methodology of Machine Learning Model

We used three different models: Random Forest (sklearn), XGBoost tree learner (shown in draft1.ipynb) and XGBoost logistic regression (also draft1.ipynb)

4.1 Random Forest

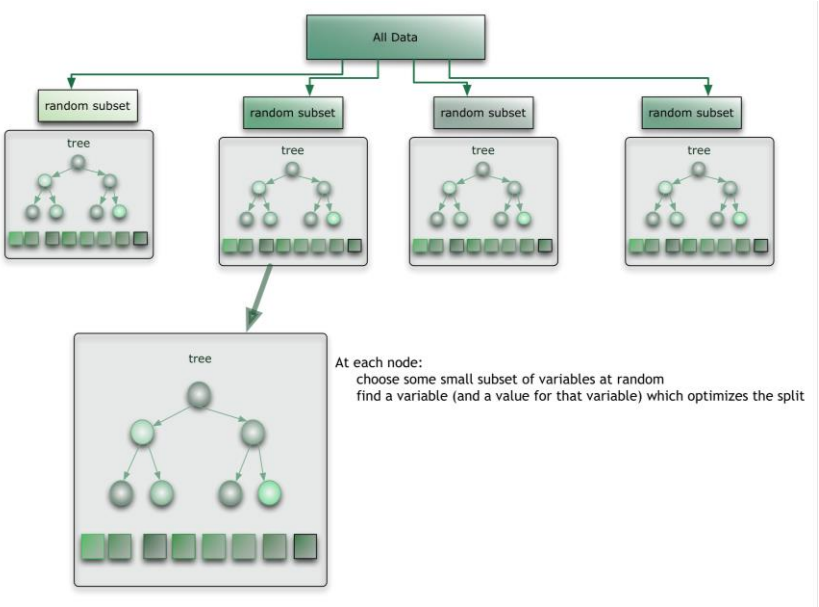

Figure 4: Diagram explanation of Random Forest

4.2 Xgboost

Xgboost, short for eXtreme Gradient Boosting package, is an extension form Gradient Boosting framework by Friedman. Xgboost package is much more robust because it has both linear model solver and tree learning algorithms.

It also has additional features for doing cross validation and finding important variables. There are many parameters, which needs to be controlled to optimize the model. (Srivastava, 2016)

We want to optimize between training loss l and regularization Ω under the Xgboost, shown by the formula:

$$obj\ (\Theta) = \sum_{i}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Where $\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)\,, f_k \in \mathcal{F}, \Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$

K is the number of the trees, f is a function in the functional space $\mathcal{F}$, and $\mathcal{F}$ is the set of all possible CARTs, T is the number of leaves in the tree, w_j is the score of leaf j, λ is the leaf weight penalty parameter, and γ is the tree size penalty parameter.

An extended derivation given by Tianqi Chen:

$$Obj(F_t) = -\frac{1}{2}\sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

Which gives a score to determine how good the tree structure is.

Basically, for a given tree structure, we push the statistics G_i and H_i to the leaves they belong to, sum the statistics together, and use the formula to calculate how good the tree is. This score is like the impurity measure in a decision tree, except that it also takes the model complexity into account. (Xgboost, 2016)

5. Experimental Analysis

5.1 Model accuracy

Given training set with only 0.0729% record showing clicked (click=1), our training models Random Forest predicts only 1 click out of totally 545,420 records in the test set and model accuracy reaches to 99.92%, which is unreliable. Therefore, using prediction accuracy is unreliable and we instead quote AUC values.

5.2 XGBoost parameter calibration

We apply two models from the XGBoost package: tree classifier (gbtree) and logistic regression linear classifier (gblinear).

The first classifier models multi nominal classification with two classes. Since class distribution is highly imbalanced we set the parameter max_delta_step to 10 what helps make the classifier conservative. We also increase default values of lambda, eta and min_child_weight based on observations of validation set performance with different parameters (see draft1.ipynb for details). We allow construction of a deeper tree than default (10) since there is large number of observations that can fall into different groups. We also use subsampling that randomizes instances used to construct trees at each iteration. The choice of those hyper-parameters was achieved by sequentially performing a grid search over few values of a given parameters while keeping the other parameters the same. As a result the order in which optimal parameters are decided can change the values of the optimal parameters.

We first train the default gbtree model using the default XGBoost parameters. This model achieved AUC of 0.57 and 30% validation set. We then perform a sequential hyperparameter search for 'eta' in (0.5,0.7,1*) - "*" marks the optimal choice, 'lambda' (1,1.05*,1.1), max_depth (5,7,10*), min_child_weight(1,15,60*,100), num_round (6*,8,10). The performance with those parameters is 0.7365 on a validation set composed of 30% of observation from the training set. The performance of the model retrained on the full training set with optimal parameters is 0.81 on the training set.

In the second, linear classifier the optimal value of lambda parameter was found to be 1.1. It achieved however an AUC of only 0.62 and so was inferior to gbtree method.

5.3 Feature importance

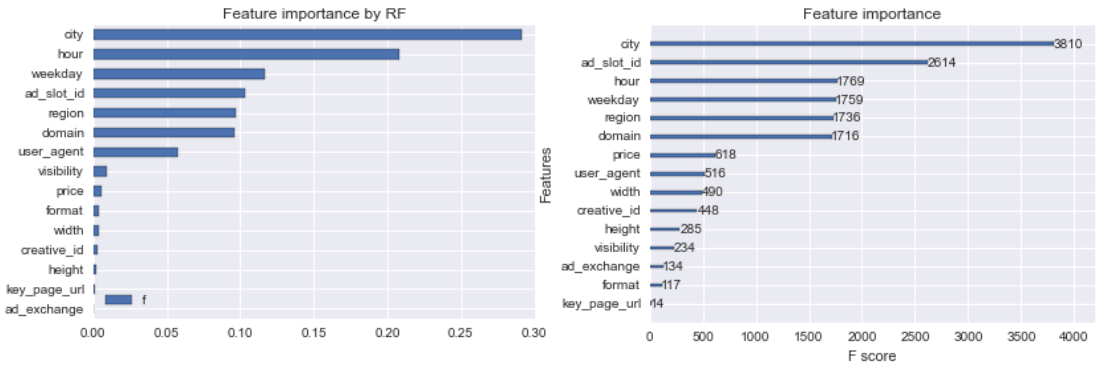

Figure 5: Feature importance returned by Random Forest and Xbgoost respectively

We ask the models to return feature importance (on restricted number of

features) after training process, shown by the Figure 5, Random Forest and Xgboost have similar evaluations. The top six significant features are city, hour, weekday, ad_slot_id, region and domain, and city ranks the first. The variable user_agent is slightly less important than the top six but much important than the rest according to Random Forest. While Xgboost assigns price, user_agent, width and creative_id as the second significant group compared with the rest.

A differently calibrated XGBoost model (on all features) provides additional insight into significance of features. The most significant features are: user_tags_0, city and user_tags_1.

Reference

Srivastava T. (2016) *How to use XGBoost algorithm in R in easy steps* [Online][Access from: http://www.analyticsvidhya.com/blog/2016/01/xgboost-algorithm-easy-steps]

Xgboost (2016) *Introduction to Boosted Trees* [Online] [Access from: http://xgboost.readthedocs.org/en/latest/model.html]