MSDS 630

Advance Machine Learning

In-class Kaggle competition

Predict in-app purchase

Keep Learning

Tian Qi[1]
Meng-Ting[2]
Miguel Romero[3]

Instructor:
Dr.Yannet Interian

March 12, 2019

Abstract

Mobile application and Data Science have experimented a significant growth in recent years. However, data in these areas is usually messy, contains a large amount of noise and signal relays on area expertise. In this project, we seek to investigate the data without serious area expertise and obtain meaningful results in the task of predicting user behaviour.

Give user events in the apps, session information, logs attributes, and messages information about an unknown app we aim to extract useful information and predict which users will make a purchase in the following 7-days and 14-days period. The provided data is granulated in users-session ([-third field] depending on the table) and a heavy section of this project is based on performing EDA of different features and aggregations types. The resulting model achieves over a 99% mean Area Under the Receiver Operating Characteristic curve (AUC).

$^1tqi2$, $^2joycemtchang$, $^3r0mer0m$

Acknowledgments

We thank Leanplum to facilitate the data and our professor Yannet Interian to propose this project.

1 Description of your dataset and EDA

1.1 EDA

There are four datasets for this problem, which are Sessions, Events, Attributes and Messages, recording users' activities on the app from 2018-10-01 through 2018-12-31. Sessions describes the time users engaged the app and some geographic data of each session. Events records what users did while they were on the app. Attributes shows the information about users. Messages contains the metadata of themselves. We started by looking at the Sessions dataset. There 22 columns in the Sessions dataset, we drop some columns that we thought trivial. For example, the columns is_mau and is_wau had only False value, thus it is reasonable to drop these two columns. We leave time-related and location-related columns. Besides, we also remove the users who were actually a developer. Secondly, we work on the Events dataset. There are about 100 million rows in the dataset, coming from about 600 thousand users. The timestamp in the Events dataset is not ready to be used for analysis since it is epoch time. We'll explain how we pre-process the time data later in feature engineering session. We look into the data with event equals to 8, which is purchase event, and find there are only 34200 purchase events. That means, only 5% of the record in Events dataset are purchase event. Last but not least, we have a look on Attribute dataset. There are two columns we are interested in, which are attribute and *attribute_value*. We first count the unique values of each attribute, and drop column with only one unique value. Moreover, we find one column has less than 100 unique values with integer data type, which we guess it would to be churn scores of users. We also look into single user's data, and discover some attributes having same *attribute_value*. While we are checking if they the same things, we find another interesting thing- there is one *attribute_value* has many 0, wide range of value, and in the format of xxx.xx, which might be m money-related column. Also, we find another column with float data type and values range from 0 to 100, that is highly possible to be LVT.

1.2 Feature Engineering

In this session, we'll explain the ways we do feature engineering, create labels and separate training and validation data.

1. Features: After the EDA, we decide to choose time-related, money-related, purchase-related(event equals to '8'), and users' score-related columns as our features. The columns we choose are: *user_id_hash* and timestamp of every dataset, *previous_sessions_duration* from Sessions dataset, and *attribute/attribute_value* from Attributes dataset. As we mention in EDA session, the timestamp data is not ready to be used since it is in epoch time with milliseconds unit, so we create a function to transfer the timestamp data to $yyyy - mm - dd$ format. During feature engineering, we adopt the concept of "window" to "aggregate" the data by "window" of each user. We create window with length of one week, then aggregate time-related/ money-related/ purchase-related by each window to create columns like 'time spent on app in week 1', 'how many times purchase in week 1', 'time spent on app in week 2', 'how many times purchase in week 2'... and so on. Other than data grouped by each window, we also have data aggregated all together, like sum of the time each user spent, some of money each user spent, and a column multiplying these two features. User's attribute data is also included. We have data like potential LVT and churn score we find in Attributes dataset.

2. Labels Creation: There are two steps for as to create the labels for each user. First, we choose the target date, let's say 2018-11-30 for example, then if a user made a purchase during 2018-12-01 to 2018-12-07, then we label 1 for 7-days and 14-days. If a user made a purchase during 2018-12-08 to 2018-12-14, then we label 0 for 7-days and label 1 for 14-days. On the other hand, if user didn't make any purchase during 2018-12-01 to 2018-12-14, then we label 0 for both 7-days. and 14-days.

3. Training/ Validation Data Split The way we separate the data to training data and validation data is shown in the picture below.

   We make use of data from 2018-10-06 to 2018-11-30 for training purpose and data from 2018-10-20 t0 2018-12-14 for validation purpose.

4. Kaggle Submission Preparation

Figure 1: Train-Validation split

Since some of the user ids from *sample_submission* file don't exist in the datasets, so we need to create the data for those new users. The way we do is to 1) add users to the training data, and 2) impute all the columns with zero.

2 Machine Learning Methods

Since this is a time-related problem, we can't use cross-validation to tune parameters, we should use a fixed validation instead. In order the get the hyper-parameter that is best for predicting future, we first train the model with different hyper-parameters by features on 10.6-11.16 (6 weeks) and label on 11.17-11.30 (both 7 days and 14days), then we move one step ahead–use the feature on 10.20-11.30 and label on 12.1-12.14 to validate, return the hyper-parameter that give the best AUC. Then with those hyper-parameters, we fit model on the feature on 10.20-11.30 and label on 12.1-12.14, then use this model and the feature extracted from 11.4-12.14, to predict the future: label for next 7 and 14 days.

Since we have two research questions, we built a separate model for each. With the sliding window framework and 15 features, we tried the random forest at first and set it as our baseline model, then we tried XGboost and LightGBM, which are two popular boosting algorithms and they usually give the better result than random forest. In order to improve our model, later we got three new features about the user's attribute, we only built the xgboost and LightGBM because the previous result showed they out-performed random forest, we also tried 2 model ensemble techniques: stacking and blending, so totally we had 7 models in the end.

3 Experimental results

For the first three models that were built without the 3 attribute features, we got local validation score (average AUC on 7 days and 14 days prediction) around 95% and the LB (Kaggle public leader board) score around 94.7%, you can see the detail from the following table:

	Validation Score	LB Score
Random Forest	0.947	0.9433
xgboost	0.953	0.9502
LightGBM	0.952	0.9489

Table 1: Performance with firstly-considered features and models.

You can see the xgboost performed best among the three, and the random forest was the last.

For the second two models added with attribute features, they all had local validation score 99.46%, and the LB score around 99%, you can see the detail from the following table:

	Validation Score	LB Score
xgboost	0.9946	0.9906
LightGBM	0.9946	0.9902

Table 2: Performance with secndly-considered features and models.

We also plot the feature importance plot, you can see all three new attributes features have a high importance score, and I think that is the reason why we got good rank in the LB. Also, in general, the closer the window feature is, the more important they are.

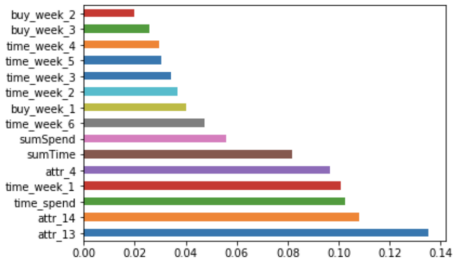

Figure 2: Feature importance for the 7 days XGboost model.

For the two ensemble models, the stacking model, which is usually a good technique to improve your score 0.00x at Kaggle, somehow performed worse than the single model–0.977 on LB. The reason is the single model is already good enough and stacking may cause extra variation (Hugo Pino). The blending way, which is just taking the average result on the two models (0.9908 on LB), slightly out-performed the best single model–xgboost, which

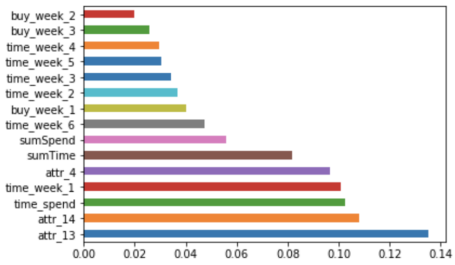

Figure 3: Feature importance for the 14 days XGboost model.

also suggest our single model is good enough.

As a result, we found that the public LB score always decreases around 0.03-0.05 than local validation score, so there might be a slight difference in distribution between them, and in order to "keep learning", we definitely need more features to improve our score at 0.00x level.

4 List the responsibilities

Tian (Luke) Qi: Feature Engineer and Modeling
Miguel Romero: EDA and Feature Engineer
Meng-Ting(Joyce) Chang: EDA and Data Preparation

5 Pointers to our code

Github repository

- EDA
- Feature engineering
- Model