網路搜索與探勘 Twitter RecSys Challenge 2020 期末書面報告

林宣佑 資管三 **106306027**
陳瑀芋 資科碩一 **108753144**
劉一凡 資科碩一 **108753213**
葉冠宏 資科碩一 **108753208**

Goal

This challenge aims to evaluate novel algorithms for predicting 4 different engagement rates at a large scale, and push the state-of-the-art in recommender systems.

4 regression(classification task)
The rate of user retweet
The rate of user reply
The rate of user retweet with comment
The rate of user like

Data Introduction

The dataset comprises of roughly 200 million public engagements, along with user and engagement features, that span a period of 2 weeks and contain public interactions (Like, Reply, Retweet and Retweet with comment), as well as 100 million pseudo negatives which are randomly sampled from the public follow graph. While sampling the latter pool of Tweets, we take special care about preserving user privacy.

Evaluation

The submitted methods will be evaluated on a held-out test set generated from more recent Tweets on the platform, and the evaluation metrics will include precision-recall area under curve (PR-AUC) and cross-entropy loss.

Experiments

Approach1-Logistic regression

model introduction:
Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable.

Description:
Because our goal is to predict whether the user will engage the tweet. We change the timestamp feature into a Boolean type. If the timestamp has some value, we label it as 1. Otherwise we label it as 0. Hence our training data could be seen as a binary-classification task format(Although the evaluation is AUC).

First of all, we treat the text tokens as a feature. We make the assumption that if the tweet's content is useful, the result will be better than the null model. We set the max length as 200 and max iteration as 1,000 times for eleven files. However when we ran a logistic regression, the result of the validation set is very pool. The result is as follows:

RetweetPRAUC: 0.1478
RetweetRCE: -410.1860
ReplyPRAUC: 0.0481
ReplyRCE: -820.6110
LikePRAUC: 0.5331
LikeRCE: -901.1610
RTWithCommentPRAUC: 0.0037
RTWithCommentRCE: -173.4100

In the experiment we use all the training data, which takes a lot of time but has such bad result. Due to the computer hardware resources limitation, we decide to use small data first to see if our work is effective. Also, because the text token seems to be not quite useful, we transfer to the dense feature.

Approach2-XGBoostRegressor

model introduction:

XGBoost is an implementation of the gradient tree boosting algorithm that is widely recognized for its efficiency and predictive accuracy.

Gradient tree boosting trains an ensemble of decision trees by training each tree to predict the *prediction error of all previous trees* in the ensemble:

$$\min_{\nabla f_{t,i}} \sum_i L(f_{t-1,i} + \nabla f_{t,i}; y_i),$$

where $\nabla f_{t,i}$ is the prediction generated by the newest decision tree for sample i and $f_{t-1,i}$ is the prediction generated by all previous trees, $L(...)$ is the loss function used and y_i is the target we are trying to predict.

Description:
The second experiment we do is using the XGBoostRegressor. We choose the first 12,000,000 data as training set. The params we set is n_estimator=100 and binary logistic objective function. We didn't use the text token, hashtag, language, all of ids as the feature. Other than this we use all we could use. Here is the result:

RetweetPRAUC: 0.1735
RetweetRCE: -46.3127
ReplyPRAUC: 0.0749
ReplyRCE: -25.6983
LikePRAUC: 0.5852

LikeRCE: -158.4220
RTWithCommentPRAUC: 0.0140
RTWithCommentRCE: -7.6065

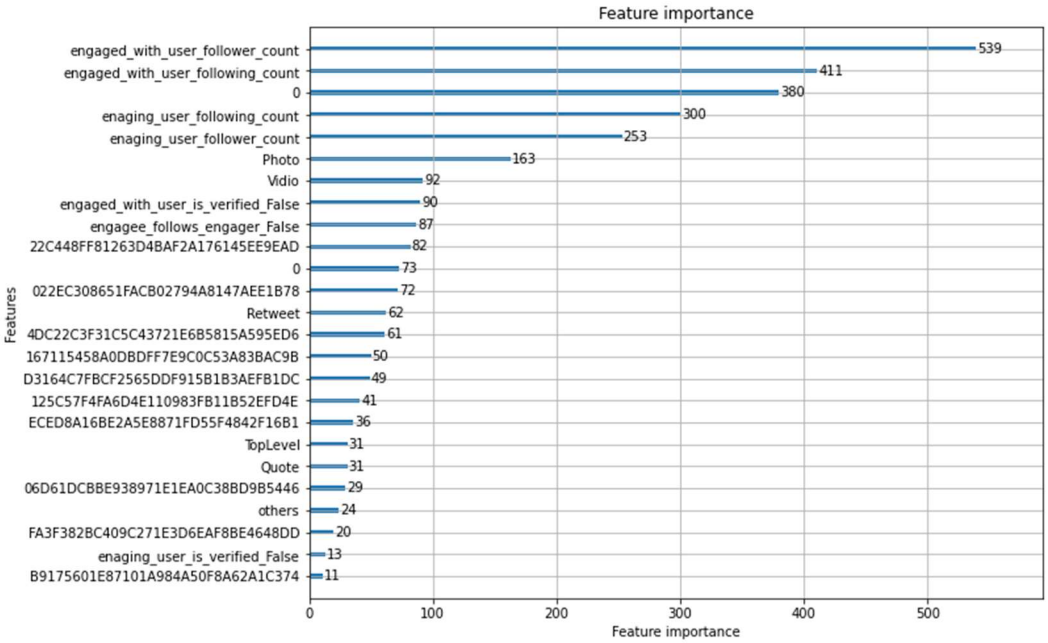
We can observe that the outcome is just slightly better than using text token. However the cross entropy loss is much lower than the previous loss. So we guess that maybe the model is somehow overfitting.

Approach3-lightgbm

model introduction:
LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

1.Faster training speed and higher efficiency.
2.Lower memory usage.
3.Better accuracy.
4.Support of parallel and GPU learning.
5.Capable of handling large-scale data.

Feature analysis:



Based on the above feature importance analysis, we choose some of the features to train our lightgbm model.

Categorical data:

present_media	Count frequency of each category of ['Photo', 'Vidio', 'Gif']
tweet_type	One-hot to [‘Retweet’, ‘Quote’, ‘Reply’, ‘TopLevel’]
language	Top 10 languages and others
present_links	Is present links or not. (1,0)
engaged_with_user_is_verified enaging_user_is_verified	True or false to One-hot

engagee_follows_engager	

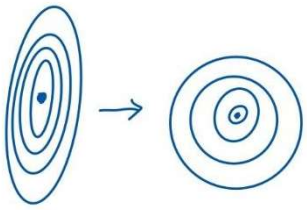
Numeric data:
engaged_with_user_follower_count,
engaged_with_user_following_count,
enaging_user_follower_count,
enaging_user_following_count

Standardization:

To equalize the importance between features, we usually scale the data by min-max scaler or z-score scaler. Here we adopt z-score normalization. There are two advantage of scaling the data:

1. Make the model more precise: Suppose we want to compare variation between two features, computing distance without normalization is incomparable.

2. Convergence is faster: when we do gradient descent with the data that have not been scaled, it might take a long time due to imbalance of feature scale(direction).



Note:
z-score normalization: set standard deviation as 1 and mean as 0, which make some statistical theory useful, e.g., PCA, Kmeans…

min-Max standardization: set min as 0 and Max as 1.

Result:

RetweetPRAUC: 0.2291
RetweetRCE: 6.6564
ReplyPRAUC: 0.0634
ReplyRCE: 4.7089
LikePRAUC: 0.5775
LikeRCE: -2.4552
RTWithCommentPRAUC: 0.0131
RTWithCommentRCE: 1.4129

Approach4-word2vec

model introduction:
Word2vec is a group of related models that are used to produce word embeddings.

Description:
Dealing with text tokens, I tried to use word2vec model to implement word embedding. With 250 dimensions, the result is extremely terrible.

RetweetPRAUC:0.2235
RetweetRCE:6.4062
ReplyPRAUC:0.0726
ReplyRCE:5.9729
LikePRAUC:0.5611
LikeRCE:-2.4552
RTWithCommentPRAUC:1.8655
RTWithCommentRCE:148

EDA:

Retweet	0.11
Reply	0.026
Retweet with comment	0.0074
Like	0.46

Due to imbalance of data, it implies that 'Retweet with comment' seldom shows up.

Approach5-XGBoostClassifier

model introduction:

XGBoost is an implementation of the gradient tree boosting algorithm that is widely recognized for its efficiency and predictive accuracy.

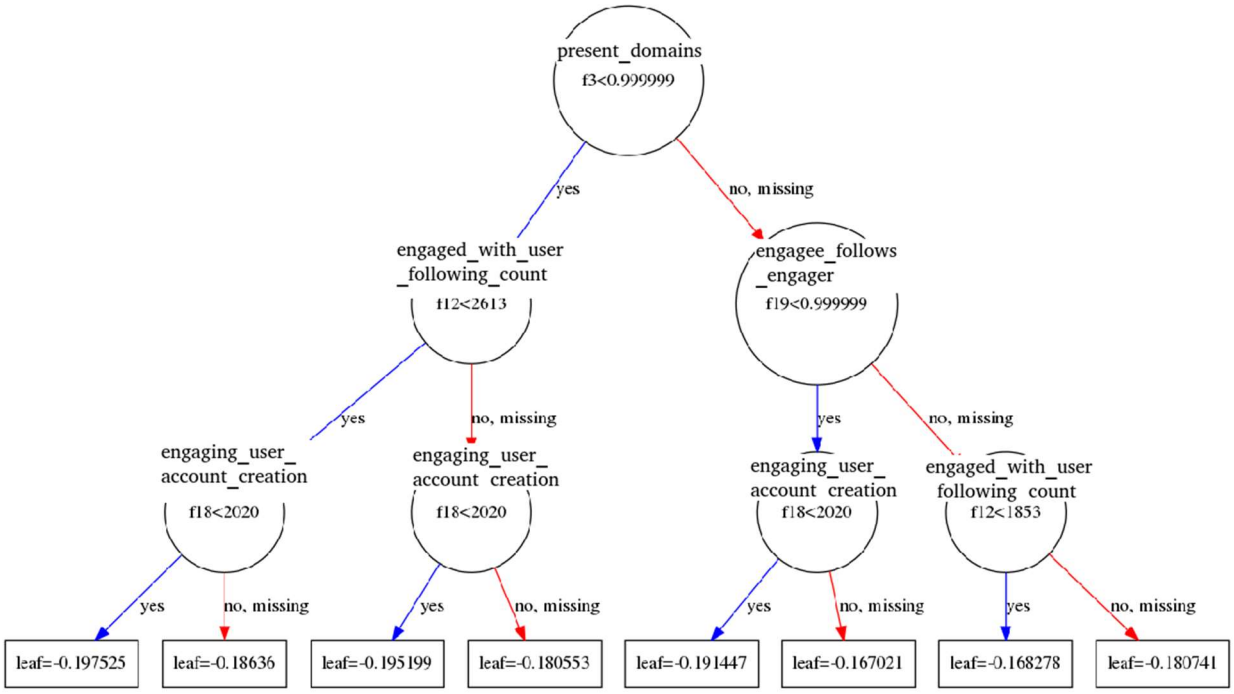
Gradient tree boosting trains an ensemble of decision trees by training each tree to predict the *prediction error of all previous trees* in the ensemble:

$$\min_{\nabla f_{t,i}} \sum_{i} L(f_{t-1,i} + \nabla f_{t,i}; y_i),$$

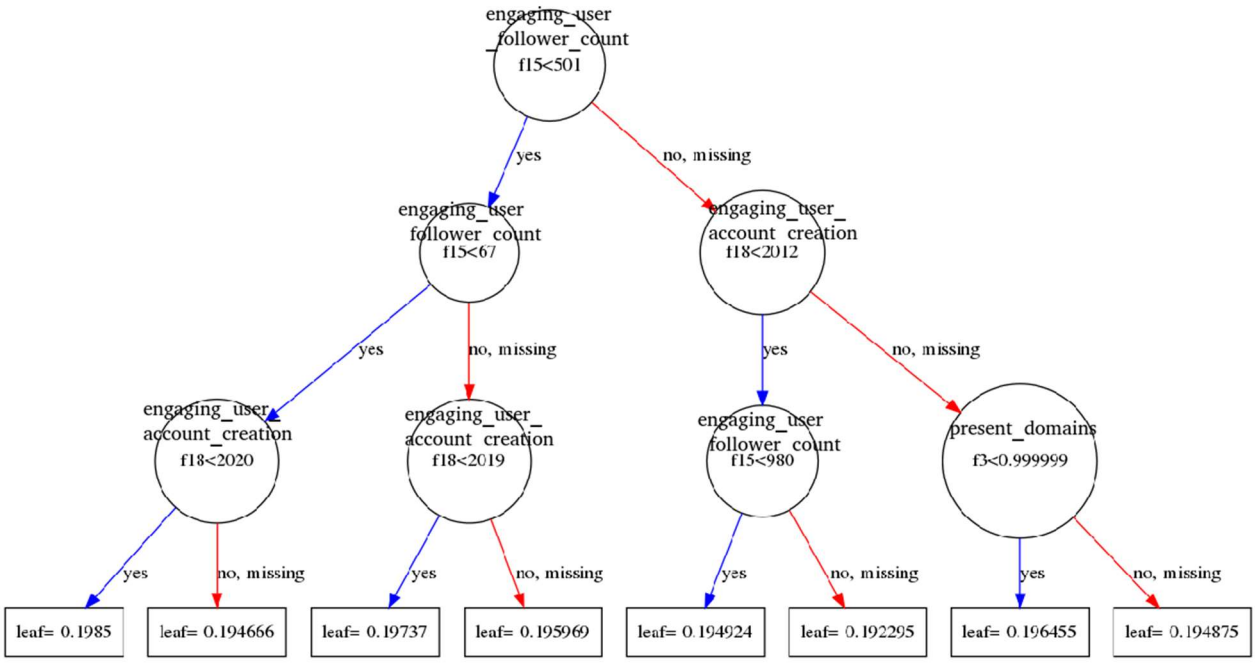
where $\nabla f_{t,i}$ is the prediction generated by the newest decision tree for sample i and $f_{t-1,i}$ is the prediction generated by all previous trees, $L(\ldots)$ is the loss function used and y_i is the target we are trying to predict.

Feature analysis:

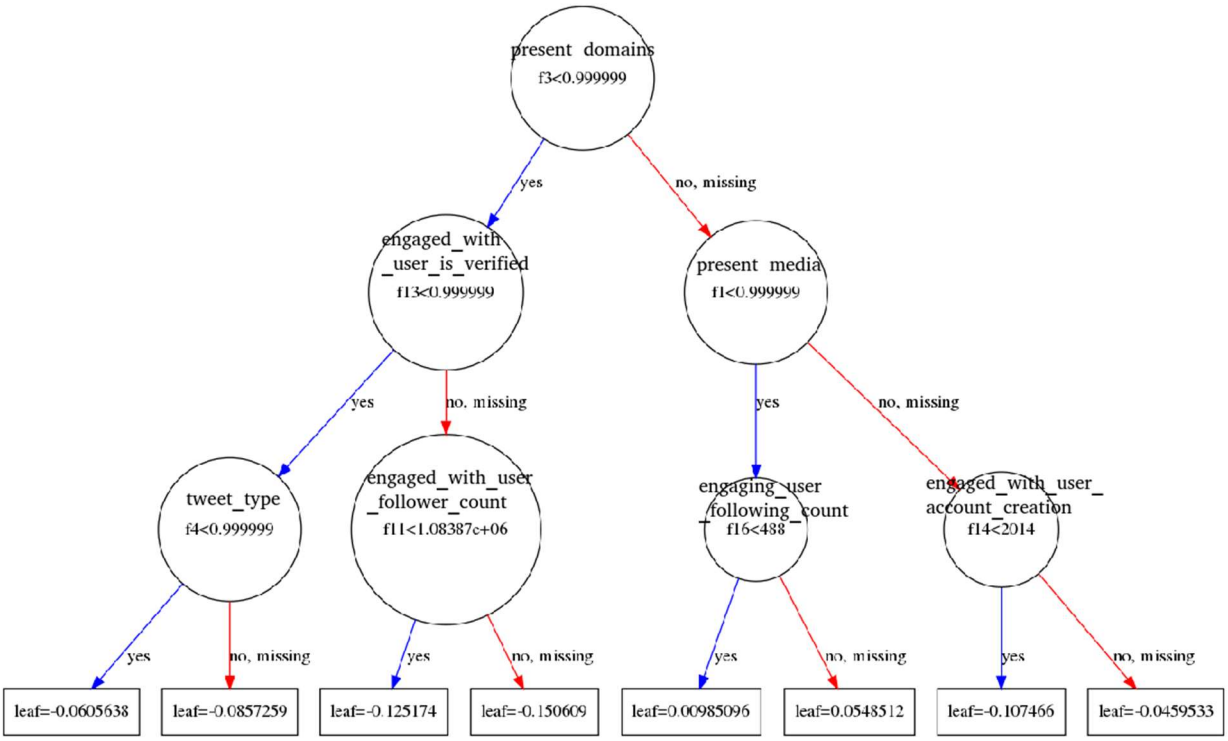
tree plot: reply



tree plot: retweet with comment



tree plot: like



1,'present_media' **2,'present_links'** **3,'present_domains'** **4,'tweet_type'** **5,'languages'** **6,'tweet_year'** **7,'tweet_month'** **8,'tweet_day'** **9,'tweet_hour'** **10,'tweet_weekday'**	**11,'engaged_with_user_follower_count'** **12,'engaged_with_user_following_count'** **13,'engaged_with_user_is_verified'** **14,'engaged_with_user_account_creation'** **15,'engaging_user_follower_count'** **16,'engaging_user_following_count'** **17,'engaging_user_is_verified'** **18,'engaging_user_account_creation'** **19,'engagee_follows_engager'**
reply [0. 0.05714286 0. 0.14285715 0. 0. 0. 0. 0. 0. 0. 0.03809524 0.23809524 0. 0. 0.0047619 0.09047619 0. 0.2952381 0.13333334]	**retweet with comment** [0. 0.05714286 0. 0.14285715 0. 0. 0. 0. 0. 0. 0. 0.03809524 0.23809524 0. 0. 0.0047619 0.09047619 0. 0.2952381 0.13333334]
like [0. 0.05714286 0. 0.14285715 0. 0. 0. 0. 0. 0. 0. 0.03809524 0.23809524 0. 0. 0.0047619 0.09047619 0. 0.2952381 0.13333334]	

Attempt1:

We try to use the XGBoostClassifier rather than Regressor and set the n_estimators=10. This time we treat our features more carefully. We choose the feature with human intuition. Here we use present_links, present_domains as boolean; present_media, tweet_type, language as category(for present_media we do the count vectorizer because there might be several media presentation); engaged_with_user_is_verified, enaging_user_is_verified, engagee_follows_engager as boolean; engaged_with_user_follower_count, engaged_with_user_following_count, enaging_user_follower_count, enaging_user_following_count as int.
We didn't use the timestamp features.

Here is the result:

RetweetPRAUC:0.3291
RetweetRCE:-400.4080
ReplyPRAUC:0.5135
ReplyRCE:-250.4300
LikePRAUC:0.6728
LikeRCE:-750.6050
RTWithCommentPRAUC:0.5037
RTWithCommentRCE:-172.7440

The performance significantly improves!

Attempt 2:

To make sure the timestamp is useless, we add the timestamp features. We also transform the tweet timestamp into year, month, day, hour and whether it is in weekdays. For engaged_with_user_account_creation and enaging_user_account_creation, we use the original timestamp.

Here is the result:

RetweetPRAUC:0.3293
RetweetRCE:-400.0680
ReplyPRAUC:0.5135
ReplyRCE:-250.4300
LikePRAUC:0.6223
LikeRCE:-770.9690
RTWithCommentPRAUC:0.5037
RTWithCommentRCE:-172.7440

The result shows that timestamp influences 'LikePRAUC' in a bad way, and 'RetweetPRAUC' only performances slightly better. We speculate the time feature disturbs the model from the right way. Maybe it's because the users of Twitter are from all around the world and everyone has different timezone. The storage of timestamp is all the

same, but it doesn't have the information of which timezone it is. Otherwise, intuitively we think that there are more tweets during the break than working hours.

Attempt 3:

This time, we want to try the text feature again. But we extract the length of text as a dense feature. We also add a feature called bothAreFamous. It comes from the enaging_user_follower_count and engaged_with_user_follower_count. If both of the followers are higher than 10,000, we give the bothAreFamous value as 1, otherwise 0. Also we change the params a little. We change the n_estimators=20 and max_depth=10(the default is six).

Here is the result:

RetweetPRAUC:0.3272
RetweetRCE:-399.4690
ReplyPRAUC:0.1182
ReplyRCE:-250.5870
LikePRAUC:0.6322
LikeRCE:-760.9180
RTWithCommentPRAUC:0.0037
RTWithCommentRCE:-172.8970

The result is still not as good as only using original features.

Attempt 4:

Due to the imbalance of data, it implies that 'Retweet with comment' seldom shows up. Thus, we use null model for prediction of 'Retweet with comment' that it predicts for all 0s as a baseline. Here is the result:

RetweetPRAUC:0.5516
RetweetRCE:-401.0440
ReplyPRAUC:0.5135
ReplyRCE:-250.4300
LikePRAUC:0.6728
LikeRCE:-750.6050
RTWithCommentPRAUC:0.5037
RTWithCommentRCE:-172.7440

We can see that only guessing zeros for 'Retweet with comment' is far better than training it in attempt 3. The best AUC we had previously is 0.3293, but that of all guessing zeros is 0.5516. Finally, we decide to take attempt 4 as our final answer.

Reflection

1. Although we found ‘Retweet with comment’ is imbalanced, we hadn’t done any of sampling techniques like ADASYN or SMOTE.

2. We should first submit this one as a baseline, i.e., EDA should have a higher priority than things above.