Project report

——Regression for prediction

Li Ma

11-06-2019

I. Initialization step

In the first step, we decided to choose the dataset BlackFriday for our project, then elaborated the use-case diagram and detailed description of the most important cases and defined the global architecture of our project. After the initial discussion, we assigned our respective tasks and I was responsible for back-end machine learning (Regression for prediction)

The part of backend is machine learning of prediction and classification., my initial vision for the backend framework was Ngnix + uWGSI + Flask(python) + Cloud server and the process is as follows

* Web browser sends an HTTP request to Nginx
* Nginx forwarded to uWSGI
* uWSGI processes it according to the WSGI specification
* uWSGI will send HTTP requests to the Flask application for processing
* Flask application handles HTTP

Get data from an HTTP request and set it as input for the prediction model(trained)

* Flask generates an HTTP response (the prediction result), which is given to uWSGI according to the WSGI specification.
* uWSGI sends HTTP response to Nginx via TCP
* Nginx sends the HTTP response to the web browser via TCP

II. Elaboration step

quick overview of the data (data structure):

Data columns (total 12 columns):

User\_ID 537577 non-null int64

Product\_ID 537577 non-null object

Gender 537577 non-null object

Age 537577 non-null object

Occupation 537577 non-null int64

City\_Category 537577 non-null object

Stay\_In\_Current\_City\_Years 537577 non-null object

Marital\_Status 537577 non-null int64

Product\_Category\_1 537577 non-null int64

Product\_Category\_2 370591 non-null float64

Product\_Category\_3 164278 non-null float64

Purchase 537577 non-null int64

dtypes: float64(2), int64(5), object(5)

None

User\_ID unique element: 5891

Product\_ID unique element: 3623

Gender unique element: 2

Age unique element: 7

Occupation unique element: 21

City\_Category unique element: 3

Stay\_In\_Current\_City\_Years unique element: 5

Marital\_Status unique element: 2

Product\_Category\_1 unique element: 18

Product\_Category\_2 unique element: 18

Product\_Category\_3 unique element: 16

Purchase unique element: 17959

This dataset has 537577 entries with 12 columns (potential features ). The unique User\_ID and Product\_ID are 5891 and 3623, respectively. They are small compared to the total number of entries, therefore, it can be inferred that a great portion of User\_ID and Product\_ID repeat many times and they may contain crucial information. However, these information are only available for the target variable Purchase. We want to predict Purchase based on the information Gender, Age, etc so that they can reflect those in their marketing strategies that target new customers. And according to the info, there are 2 columns where data is missing, Product\_Category2 and Product\_Category\_3 is missing more, so they are not considered in this analysis.

The dataset explored through eda (exploratory data analysis) to provide the necessary conclusions for subsequent processing and modeling. And my group partners Xinzhi Huo and Zhelin Liang did the data analysis section and showed the relationship between data.

III. Construction step

The main goal of the project is to train a model that can predict purchase according to the customers’ information. it is going be a straightforward regression problem where the model performance is evaluated by root mean squared error (RMSE)

Simple Features selection:

According to the step II, I found that User\_ID and Product\_ID are the most relevant to purchases and other attributes are not so highly. So I considered training two cases:

Case1: Train with all features except purchases (In this way, we can predict the purchase amount of old customers in the store based on the established information.)

Case2: Train with all features except purchases,User\_ID and Product\_ID (This way we can predict the purchases of new customers in the store based on given information.)

Simple Features encodeing:

Convert gender strings to binary encoding so that It is better for numerical calculations

Case1: while the ordinal categorical features can be simply encoded with integers, the nominal categorical features need to be one-hot-encoded. In this case, however, the number of unique entires for User\_ID and Product\_ID is too big and one-hot-encoding these features will unnecessarily increase the data dimension and therefore cardinality. In real case, User\_ID and Product\_ID tended to be label-encoded. And other features tended to be one-hot-encoded

Case2: all of the features tended to be one-hot-encoded

then split our train and test data and will standardize the data using StandardScaler

Model selection:

Once the Feature was ready, I started training with some common models. The most commonly used models on Kaggle are basically tree-based models:

* Gradient Boosting
* Random Forest
* Extra Randomized Trees

These models can basically be used by sklearn.

Excellent performance of Gradient Boosting plus Xgboost efficient implementation also make it a good choice for regression.

Through simple learning, considering performance and time, I decided to use two types of models to train: RandomForestRegressor and XgboostRegressor for Regression Prediction.

Before model tuning, I determined the best set of parameters through a process called Grid Search. In fact, this process is arguing that it is a violent search for all combinations based on a given parameter candidate.

param\_grid = {'n\_estimators': [10, 30, 100, 150, 300], 'max\_features': [1,3,5,7,9,11]}

model = grid\_search.GridSearchCV(estimator=rfr, param\_grid=param\_grid, n\_jobs=1, cv=10, verbose=20, scoring=RMSE)

model.fit(X\_train, y\_train)

Model parameters

According to the official document:

The parameters that RF needs to adjust include two parts. The first part is the parameters of the Bagging framework, and the second part is the parameters of the CART decision tree.

RF Bagging framework parameters:

1) n\_estimators: This is the maximum number of iterations of the weak learner, or the maximum number of weak learners. In general, n\_estimators are too small, easy to fit, n\_estimators is too large, the amount of calculation will be too large, and after n\_estimators reaches a certain number, the model increase obtained by increasing n\_estimators will be small, so generally choose a moderate value. The default is 100.

2) oob\_score: Whether to use the sample outside the bag to evaluate the quality of the model. The default is False. I tended to set to True because the out-of-bag score reflects the generalization ability of a model after fitting.

3) criterion: The evaluation criterion of the feature when the CART tree is divided. The CART regression tree corresponding to the regression RF defaults to the mean square error MSE, and another criterion that can be selected is the absolute value difference MAE In general, choosing the default standard is already very good. And the document said that RMSE could be used too.

RF decision tree parameters:

1) The maximum number of features considered in RF partitioning max\_features: There are many types of values that can be used. The default is "auto", which means that sqrtN features are considered in the division; if "log2" means division Consider up to log2N features; if it is "sqrt" or "auto" means that considering up to N features. If it is an integer, it represents the absolute number of features considered. If it is a floating point number, it means taking into account the feature percentage, that is, taking into account (percent xN) the number of features after rounding. Where N is the total number of features of the sample. Generally I used the default "auto". If the number of features is very large, we can flexibly use the other values just described to control the maximum number of features considered in the partitioning to control the generation time of the decision tree. (By the way, Random Forest generally gets the best results near the square root of max\_features set to the number of features)

2) The maximum depth of the decision tree max\_depth: The default is not to enter. If not entered, the decision tree does not limit the depth of the subtree when creating the subtree. In general, this value can be ignored when there is little data or features. If the model sample size is large and there are many features, it is recommended to limit this maximum depth. The specific value depends on the distribution of the data. Commonly used values can be between 10-100

3) The minimum number of samples required for internal node subdivision min\_samples\_split: This value limits the conditions under which the subtree continues to be partitioned. If the number of samples of a node is less than min\_samples\_split, it will not continue to try to select the optimal feature for partitioning. The default is 2. If the sample size is not large, I do not need to control this value. If the sample size is very large, it is recommended to increase this value.

4) The minimum number of samples of the leaf node min\_samples\_leaf: This value limits the minimum number of samples of the leaf node. If the number of leaf nodes is smaller than the number of samples, it will be pruned together with the sibling node. The default is 1, enter the minimum number of samples, or the minimum number of samples as a percentage of the total number of samples. If the sample size is not large, do not need to control this value. If the sample size is very large, it is recommended to increase this value.

5) The smallest sample weight of the leaf node and min\_weight\_fraction\_leaf: This value limits the minimum value of the sum of all sample weights of the leaf nodes. If it is less than this value, it will be pruned together with the sibling node. The default is 0, which means that the weight problem is not considered. In general, if I have more samples with missing values, or if the classification of the classification tree sample is very different, the sample weight will be introduced. At this time, must pay attention to this value.

6) Maximum number of leaf nodes max\_leaf\_nodes: By limiting the maximum number of leaf nodes, over-fitting can be prevented. The default is "None", that is, the maximum number of leaf nodes is not limited. If a limit is imposed, the algorithm will establish an optimal decision tree within the maximum number of leaf nodes. If there are not many features, ignore this value, but if the feature is divided into many, limit it. The specific value can be obtained by cross-validation.

The most important of the above decision tree parameters include the maximum number of features max\_features, the maximum depth max\_depth, the minimum number of samples required for internal node subdivision min\_samples\_split and the minimum number of samples of leaf nodes min\_samples\_leaf

All the XGBoost parameters divided into three categories:

* General parameters: macro function control.
* Booster parameter: Controls the booster (tree/regression) for each step.
* Learning target parameters: Control the performance of training objectives.

General parameters

These parameters are used to control the macro functions of XGBoost:

1) booster [default gbtree] There are two options for choosing a model for each iteration:

Gbtree: tree-based model

Gbliner: linear model

2) Silent[default 0] When this parameter value is 1, the silent mode is on and no information is output.Normally this parameter will remain at the default of 0, because this will help us better understand the model.

3) Nthread [default is the maximum number of possible threads] This parameter is used for multi-thread control and should be entered into the system's core count.

Booster parameter

From xgboost-unity, the bst: prefix is no longer needed for booster parameters. Parameter with or without bst: prefix will be equivalent(both bst:eta and eta will be valid parameter setting)

1) eta [default 0.3] By reducing the weight of each step, the robustness of the model can be improved.Typical values ​​are 0.01-0.2.

2) min\_child\_weight [default 1] Determine the minimum leaf node sample weight sum.This parameter of XGBoost is the sum of the minimum sample weights.This parameter is used to avoid overfitting. When its value is large, the model can be avoided to learn local special samples.But if this value is too high, it will lead to under-fitting. This parameter needs to be adjusted using CV.

3) max\_depth [default 6] this value is the maximum depth of the tree.This value is also used to avoid overfitting. The larger the max\_depth, the more specific and more local samples will be learned.Typical value: 3-10

4) max\_leaf\_nodes.The maximum number of nodes or leaves on the tree.Can replace the role of max\_depth. Because if a binary tree is generated, a tree with a depth of n generates n^2 at most Leaves.

5)gamma [default 0] When the node is split, the node will be split only if the value of the post-split loss function drops. Gamma specifies the minimum loss function drop value required for node splitting.The larger the value of this parameter, the more conservative the algorithm. The value of this parameter is closely related to the loss function, so it needs to be adjusted.

6)max\_delta\_step [default 0] This parameter limits the maximum step size for each tree weight change. If the value of this parameter is 0, it means there is no constraint. If it is given a positive value, it will make the algorithm more conservative.Usually, this parameter does not need to be set. But when the samples in each category are very unbalanced, it is very helpful for logistic regression.

7)subsample [default 1] It is exactly the same as the subsample parameter in GBM. This parameter controls the proportion of random samples for each tree.By reducing the value of this parameter, the algorithm is more conservative and avoids overfitting. However, if this value is set too small, it may cause an under-fitting.Typical value: 0.5-1.

8)colsample\_bytree [default 1] Used to control the proportion of the number of columns per random sample (each column is a feature).Typical value: 0.5-1.

9)colsample\_bylevel [default 1] Used to control each split of each level of the tree, the proportion of samples of the number of columns. I personally don't use this parameter very much, because the subsample parameter and the colsample\_bytree parameter can play the same role. But if you are interested, you can dig more useful for this parameter.

10)lambda [default 1] The L2 regularization term of the weight. (similar to Ridge regression).This parameter is used to control the regularization part of XGBoost. Although most data scientists rarely use this parameter, this parameter can still be used for more purposes in reducing overfitting.

11) alpha [default 1] The L1 regularization term of the weight. (similar to Lasso regression).Can be applied in the case of very high dimensions, making the algorithm faster.

12) scale\_pos\_weight [default 1] When the samples in each category are very unbalanced, setting this parameter to a positive value will make the algorithm converge faster.

Learning target parameter

These parameter are used to control the ideal optimization goal and the measurement method of each step result.

1) Objective [default=reg:linear] Define learning tasks and corresponding learning objectives. The optional objective functions for our regression prediction is :"reg:linear" – linear regression.

2) Eval\_metric [default according to objective] Rmse for regression .

3) Seed [default 0] Seed of random number.Set it to reproduce the results of random data, and can also be used to adjust parameters.

The tuning of Xgboost. The parameters that are generally considered to have a large impact on its performance are:

Eta: The step size when the weight is updated after each iteration. The smaller the training, the slower the training.

Num\_round: The number of iterations in total.

Subsample: The proportion of data used to train each tree when training. Used to prevent Overfitting.

Colsample\_bytree: The proportion of features used to train each tree.

Max\_depth: The maximum depth limit per tree. Unlike Random Forest, Gradient Boosting will eventually overfit without limiting the depth.

Early\_stopping\_rounds: Used to control how many iterations of the Out Of Sample's validation set have not been improved and the training is terminated early. Used to prevent Overfitting.

The general tuning steps are:

1.A portion of the training data is drawn out as a validation set.

2.First set eta higher (such as 0.1) and num\_round to 300 ~ 500.

3.Search for other parameters with Grid Search

4.Gradually reduce eta and find the best value.

5.With the validation set as watchlist, retrain on the training set with the best combination of parameters found. Observe the output of the algorithm and see how the scores on the validation set change after each iteration to get the best early\_stopping\_rounds.

Training process:

RandomForestRegressor:

params = {

'n\_estimators': [10, 30, 100, 150, 300],

'max\_depth': [3, 5, 7, 9 ]

}

gridsearch=GridSearchCV(RandomForestRegressor(),param\_grid=params, cv=3,scoring='neg\_mean\_squared\_error', n\_jobs=-1)

grid\_search.fit(X\_train, y\_train)

preds = grid\_search.predict(X\_test)

print("最佳参数: {}".format(grid\_search.best\_params\_))

print("RMSE 分数: {}".format(mean\_squared\_error(y\_test, preds) \*\* 0.5))

XGBRegressor:

cv\_params = {'n\_estimators': [400, 500, 600, 700, 800]}

other\_params = {'learning\_rate': 0.1, 'n\_estimators': 500, 'max\_depth': 5, 'min\_child\_weight': 1, 'seed': 0,'subsample': 0.8, 'colsample\_bytree': 0.8, 'gamma': 0, 'reg\_alpha': 0, 'reg\_lambda': 1}

cv\_params = {'n\_estimators': [550, 575, 600, 650, 675]}

other\_params = {'learning\_rate': 0.1, 'n\_estimators': 600, 'max\_depth': 5, 'min\_child\_weight': 1, 'seed': 0,'subsample': 0.8, 'colsample\_bytree': 0.8, 'gamma': 0, 'reg\_alpha': 0, 'reg\_lambda': 1}

cv\_params = {'max\_depth': [3, 4, 5, 6, 7, 8, 9, 10], 'min\_child\_weight': [1, 2, 3, 4, 5, 6]}

other\_params = {'learning\_rate': 0.1, 'n\_estimators': 550, 'max\_depth': 5, 'min\_child\_weight': 1, 'seed': 0,'subsample': 0.8, 'colsample\_bytree': 0.8, 'gamma': 0, 'reg\_alpha': 0, 'reg\_lambda': 1}

cv\_params = {'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6]}

other\_params = {'learning\_rate': 0.1, 'n\_estimators': 550, 'max\_depth': 4, 'min\_child\_weight': 5, 'seed': 0,'subsample': 0.8, 'colsample\_bytree': 0.8, 'gamma': 0, 'reg\_alpha': 0, 'reg\_lambda': 1}

cv\_params = {'subsample': [0.6, 0.7, 0.8, 0.9], 'colsample\_bytree': [0.6, 0.7, 0.8, 0.9]}

other\_params = {'learning\_rate': 0.1, 'n\_estimators': 550, 'max\_depth': 4, 'min\_child\_weight': 5, 'seed': 0,'subsample': 0.8, 'colsample\_bytree': 0.8, 'gamma': 0.1, 'reg\_alpha': 0, 'reg\_lambda': 1}

cv\_params = {'reg\_alpha': [0.05, 0.1, 1, 2, 3], 'reg\_lambda': [0.05, 0.1, 1, 2, 3]}

other\_params = {'learning\_rate': 0.1, 'n\_estimators': 550, 'max\_depth': 4, 'min\_child\_weight': 5, 'seed': 0,'subsample': 0.7, 'colsample\_bytree': 0.7, 'gamma': 0.1, 'reg\_alpha': 0, 'reg\_lambda': 1}

cv\_params = {'learning\_rate': [0.01, 0.05, 0.07, 0.1, 0.2]}

other\_params = {'learning\_rate': 0.1, 'n\_estimators': 550, 'max\_depth': 4, 'min\_child\_weight': 5, 'seed': 0,'subsample': 0.7, 'colsample\_bytree': 0.7, 'gamma': 0.1, 'reg\_alpha': 1, 'reg\_lambda': 1}

model = xgb.XGBRegressor(\*\*other\_params)

optimized\_GBM = GridSearchCV(estimator=model, param\_grid=cv\_params, scoring=' neg\_mean\_squared\_error ', cv=5, verbose=1, n\_jobs=4)

optimized\_GBM.fit(X\_train, y\_train)

evalute\_result = optimized\_GBM.grid\_scores\_

optimized\_GBM.fit(X\_train, y\_train)

preds = optimized\_GBM.predict(X\_test)

print("RMSE 分数: {}".format(mean\_squared\_error(y\_test, preds) \*\* 0.5))

print('每轮迭代运行结果:{0}'.format(evalute\_result))

print('最佳参数：{0}'.format(optimized\_GBM.best\_params\_))

print('最佳模型得分:{0}'.format(optimized\_GBM.best\_score\_))

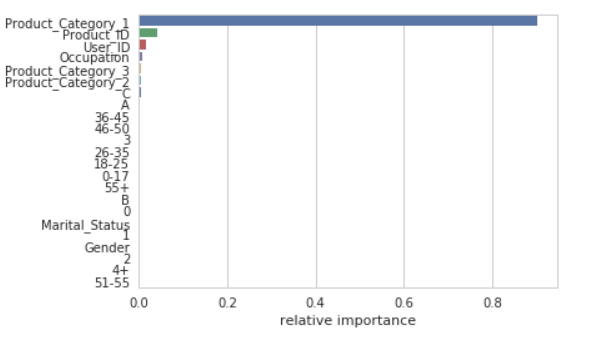
Training result:

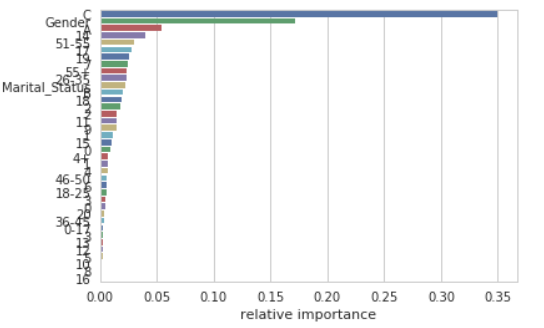
RFregressor: RMSE for case1 : 2930; for case2 : 4422

Xgbregressor: RMSE for case1: 3622; for case2: 6917

From the results of the assessment: Although xgb has better speed performance, it does not perform satisfactorily on this data set. So I choose the RF model

and the feature importance for case 1 and case 2 is





This told that the current model heavily relies on Product\_Category\_1 (followed by Product\_ID and User\_ID) to make prediction on Purchase. This was somewhat expected from EDA as they are directly related to the amount of Purchase that the customers make.and considering only the low-relevant features, now it seemed like the model suffers from high bias issue. Knowing that this dataset is artificial, it could possibly be that the given low-level features do not simply have enough predictive power for Purchase.

As a conclusion, for the purpose of predicting customer's Purchase, the top three features useful included Product\_Category\_1, Product\_ID and User\_ID. However, if the goal is to predict Purchase for the new customer who have never been in the store, those customer-specific and product-specific information cannot be utilized. In this case, only the low-relevant features can be utilized to train the model.

Model improvement

1. different encoding: one-hot encode only for age, city ,and citystay
2. Delete feature quantities with low correlation like marital\_status, gender

The above steps in order to reduce the number of features

1. Change parameters referenced network tutorial and official document

However, these modifications did not improve performance. Rmse have not broken 2930

IV. Deployment

Because of cloud server python environment deployment problems, we are only deploying in the flask production environment.

