New York University Shanghai

Solutions for Kaggle Competition

SHBI-GB 7311 B1: Machine Learning for Business (Fall 2020)

In this article, I described the process of finishing the classification and regression tasks of Kaggle competition. First, I made preparations, such as importing the data and packages, and defining functions. Second, I cleaned the data by dropping and encoding some series. Also, I deleted outliers and created validation set in this step. Then, I tried Decision Tree, Random Forest, and XGBoost separately. Finally, I used XGBoost to make prediction in both classification task and regression task. I used Python to finish both tasks.

1. Preparations: Data, Packages, and Functions

First, I imported the packages needed. (see Figure 1)

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import f1_score
```

Figure 1: Import the Packages

Then, I defined a function to count how many unique values in each column and to show how many times each unique value appears. (see Figure 2)

```
def describe_data(variablename):
    unique_value = list(variablename.unique())  # unique value in 
each column
    print('There are %i unique values in this column.' 
%len(unique_value))
    print('The frequency of each value is shown below:')
    print(pd.DataFrame(variablename.value_counts()).sort_index()) # 
how many time each unique value appears
    return  pd.DataFrame(variablename.value_counts()).sort_index()
```

Figure 2: Define a Function to Explore the Data

Next, I loaded the data. (see figure 3)

```
train_df = pd.read_csv(r'D:\machine-learning-for-business-
classification\dataframe_train.csv')
test_df = pd.read_csv(r'D:\machine-learning-for-business-
classification\dataframe_test.csv')
classification_df = pd.read_csv(r'D:\machine-learning-for-business-
classification\Classification.csv',index_col=0)
regression_df = pd.read_csv(r'D:\machine-learning-for-business-
regression\Regression.csv',index_col=0)
```

Figure 3: Load the Data

2. Data Cleaning

By calling the function in Figure 2 and using all the variables as argument respectively, I made a data dictionary as is shown in Table 1. (Also see the Excel file)

This table helped me know what is the nature and economic meaning of each variable. Then it helped me decide what variables should be added to my model.

Table 1: The Data Dictionary of the Data Set

Name	Description	Range	unique value	type
courier_id	The ID of the delivery-men	[10007871, 125996858]	979	nominal
date	The date when the observation was recorded.	[2020.2.1, 2020.2.27]	27	interval
wave_index	A wave is a series of action performed by a courier, including a series of delivery and pickup. In a wave, a courier would have a bundle of orders (tracking_id). He would need to decide which item to pick-up first, whether to keep picking up items or to deliver some item first.	[0, 16]	17	
tracking_id		2.100070e+18, 2.100080e+18	2	
courier_wave_start_lng	It is the longitude of the courier when he begins the wave. It should be a matter of consideration given the bundle of actions to perform. Specifically, which item to pickup first, and what action should be taken right afterwards. Supposedly, in one wave, the longitude and latitude should be unique.	[119.88, 122.26]	Continuous	interval
courier_wave_start_lat	The starting latitude of that wave of a certain courier	[36.06, 39.71]	Continuous	interval
action_type	Delivery-man's choice: to pick up a new order or to deliver an old one	delivery, pick up	2	nominal
group		2.020020e+16, 2.020020e+17, 2.020020e+18	3	
level	The level of the courier	[0, 3]	4	ordinal
speed	The speed of the courier	[3.01, 6.94]	Continuous	ratio
max_load	The max load of the courier	[1, 19]	16	ratio
weather_grade	The weather condition of the order	Bad Weather, Normal Weather, Slightly Bad Weather, Very Bad Weather	4	nominal
aoi_id	The id of the Area of Interest (i.e. the delivery destination).	-- i.e. 0001e4c643b3623dea2a0e9bce7d15ad	34912	nominal
shop_id	The id of the shop.	-- i.e. 00009e27a7938a119afe10d36649fa1d	11193	nominal
id		[0, 509603]	509604	
source_type	The information of the courier's previous action. 'source_type' and 'target_type' are categorical values containing three values: 'Assign', 'PickFood', and 'DeliverFood'. When type is 'Assign', the location attributes represent the courier's location; When type is 'PickFood' the location attributes represent the vendor's location; When type is 'DeliverFood' the location attributes represent the customer's location.	ASSIGN, DELIVERY, PICKUP	3	nominal
source_tracking_id		2.100070e+18, 2.100080e+18	2	
source_lng		[119.88, 122.26]	Continuous	interval
source_lat		[36.06, 39.71]	Continuous	interval
target_lng	The geographical information of the target.	[121.06, 122.26]	Continuous	interval
target_lat	The geographical information of the target.	[38.83, 39.70]	Continuous	interval
grid_distance	The shortest traversable distance to the target provided by the GPS.the distance between the source and the target. this is a distance provided by maps	[0, 429173]	Continuous	ratio
expected_use_time	The outcome variable in the regression task	[1, 9246]	Continuous	ratio
urgency	Identifies how urgent the order is	[-340771, 11345]		
hour	The hour in the day.	[6, 23]	Continuous	interval

Furthermore, I explored the data set with other methods to help me know deeply how I should cope with the data set. (see Figure 4)

```
# describe data
# training set
print(train_df.columns.values)  # see column names
train_df.head(1).T  # see sample data
train_df.tail(1).T
train_df.info()  # see core information
train_df.describe()
train_df['date']  # see one column
train_df.query('courier_id==10007871')
describe_data(test_df['source_type'])  # describe each column
# testing set
test_df.info()
print(test_df.columns.values)
test_df.describe()
```

Figure 4: Explore the Data Set with Other Functions

Here, I think I was familiar enough with the data set.

Then I encoded the discrete variables by converting characters to integers. The different values of these discrete variables do not mean that they differ in quantity. Thus, I use one-hot encoding to encode the three variables: action_type, weather_grade, and source_type. (see Figure 5)

The variable of action_type has two different values. Hence, I only took the 0th column of the dummy variable data frame and named it delivery_AT, which is the target variable that I will predict. "AT" means Action_type. I added the suffix to distinguish it from the states in the variable of source_type.

Weather_grade has four unique values. Therefore, ".get_dummies()" function created four new variables, each of which represents a unique weather condition. All of the four new variables were included in my new data frame.

Source_type has three unique values. I named the three new dummy variables "assign_ST", "delivery_ST", and "pickup_ST" separately.

After that, I created new data sets, train_df_concat and test_df_concat, by adding the encoded variables.

```
a_t = pd.get_dummies(train_df['action_type']).iloc[:,0] #  only take 
DELIVERY
a_t = a_t.rename('delivery_AT') # series. rename to make difference 
with source_type
w_g = pd.get_dummies(train_df['weather_grade'])
s_t = pd.get_dummies(train_df['source_type'])
s_t.columns=['assign_ST','delivery_ST','pickup_ST']
train_df_concat = pd.concat([train_df,a_t,w_g,s_t],join='outer', 
axis=1)
train_df_concat.info()
train_df_concat.head(1).T
# testing set
w_g = pd.get_dummies(test_df['weather_grade'])
s_t = pd.get_dummies(test_df['source_type']).iloc[:,1:]
s_t.columns=['assign_ST','delivery_ST','pickup_ST']
test_df_concat = pd.concat([test_df,w_g,s_t],join='outer', axis=1)
test_df_concat.info()
test_df_concat.head(1).T
```

Figure 5: Encoded the Discrete Variables

Before regression, I dropped the unnecessary variables. (see Figure 6)

In the training set, I dropped seven variables from train_df_concat to create Xtrain.

First, I dropped delivery_AT and expected_use_time because they are the two target variables that I should predict in classification and regression separately. Thus, I will include neither of them in any of my models as an independent variable.

Second, I dropped action_type, weather_grade, and source_type because I had added the one-hot encoded variables to replace them.

Finally, I also dropped aoi_id and shop_id. In my opinion, the two variables also contain useful information. However, according to Table X at the top of the document, the two variables has 35 thousand and 11 thousand unique values separately. Also, the different values of the two discrete variables do not mean that they differ in quantity. Thus, if I wanted to include them, I had had to use one-hot encoding. Then, my data frame would become really sparse and my laptop can not cope with such a big data frame when modeling.

For further research, I would do cluster with the variable of aoi_id and shop_id to utilize the information in them. For example, if shops in cluster A prepare food fastest, the delivery-men may be more willing to picked up food from shops in cluster A and they do from shop in cluster B. The two variables contain important individual characteristics.

To create Ytrain, I used delivery_AT, which labels "DELIVERY" as 1 and "PICKUP" as 0.

I did the same for the test set.

Also, I used train_test_split() to create validation set, which help me adjust the parameters of the models.

```
# creat Xtrain, Ytrain
# training set
Xtrain =
train_df_concat.drop(train_df_concat[['action_type','weather_grade','s
ource_type','expected_use_time','delivery_AT','aoi_id','shop_id']],
axis=1)
Xtrain.head(1).T
Xtrain.columns.values
Ytrain = train_df_concat.iloc[:,-8]
# validation set
Xtrain, Xvalid, Ytrain, Yvalid =train_test_split(Xtrain, Ytrain,
test_size=0.33, random_state=42)
# testing set
Xtest =
test_df_concat.drop(test_df_concat[['weather_grade','source_type','aoi
_id','shop_id']], axis=1)
Xtest.head(1).T
```

Figure 6: Dropped the Unnecessary Variables

3. Model Selection and Prediction

Then, I came to the part of models. I am going to deal with the classification problem first.

First, I did a Decision Tree model. Also, I manually fine-tuned the hyper parameters. When fine-tuning, I used f1 score of the prediction of validation set as criterion. (see Figure 7)

After fine-tuning, I set “criterion” = ”gini”, “splitter”=random. For pruning parameters, I set “max_depth = 20”, “min_samples_leaf” = 250, “min_samples_split” = 1000, “max_features” = None.

```
max_features=['auto', 'sqrt', 'log2',None]
clf=tree.DecisionTreeClassifier(criterion="gini",random_state=30,split
ter="random",max_depth=20,min_samples_leaf=250,min_samples_split=1000,
max_features=max_features[3])
clf=clf.fit(Xtrain,Ytrain)

# test the tree on validation set
# the mean accuracy on the given test data and labels
score=clf.score(Xvalid,Yvalid)  # score
print(score)  # 0.7803089767375482
# f1 score
y_pred = clf.predict(Xvalid)  # make prediction
f1 = f1_score(Yvalid, y_pred, average='binary')  # f1 score
print(f1)  # 0.7985341992900028

# information about the tree
clf.get_depth()  # the depth of the tree
[*zip(Xtrain.columns.values,clf.feature_importances_)]  # the
importance of each feature

# cope with testing set
Xtest.to_csv('Xtest.csv')
Xtest=pd.read_csv('Xtest.csv',index_col=0)
Xtest.columns.values

# predict
y_pred = clf.predict(Xtest)
classification_df['action_type_DELIVERY']=y_pred
classification_df.to_csv(r'D:\machine-learning-for-business-
classification\Classification-Mingcong2.csv')
```

Figure 7: The Decision Tree Model

One detail is that I found that 4 rows of data in the testing set are problematic. Hence, I manually modified the four rows of data in the csv file (see the content in the red box).

I used "[*zip(Xtrain.columns.values,clf.feature_importances_)]" to check the importance of each feature in the decision tree. I found out that "grid_distance" is of greatest importance. (see Figure 8)

```
[('courier_id', 0.0005790681215195582),
 ('wave_index', 0.00018138707116238813),
 ('tracking_id', 0.00017320095824008045),
 ('courier_wave_start_lng', 0.0004999269479342609),
 ('courier_wave_start_lat', 0.0004908199824472943),
 ('date', 0.00035663314206646365),
 ('group', 0.00014400009762575406),
 ('level', 0.00041254656314676166),
 ('speed', 0.0005327291746642506),
 ('max_load', 0.0008934824905856235),
 ('id', 0.0010932108211506007),
 ('source_tracking_id', 0.00016238178501846325),
 ('source_lng', 0.0007425277358673984),
 ('source_lat', 0.0009571069298419471),
 ('target_lng', 0.0019111806162168835),
 ('target_lat', 0.0004666490862069979),
 ('grid_distance', 0.5406471690977814),
 ('urgency', 0.04235864539627309),
 ('hour', 0.0020943155786848134),
 ('Bad Weather', 0.0),
 ('Normal Weather', 0.00015914496531234214),
 ('Slightly Bad Weather', 0.0001317129445123393),
 ('Very Bad Weather', 0.0009247572731972549),
 ('assign_ST', 0.4035697783349278),
 ('delivery_ST', 0.0005176248856162639),
 ('pickup_ST', 0.0)]
```

Figure 8: The Importance of Each Feature in the Decision Tree

Thus, I tried to find out whether there were outliers in “grid_distance” (see Figure 9).

```
# cleaning data
# codes to show the character of a variable
describe_data(train_df_concat['grid_distance']).iloc[-90:,:]

# Remove extreme outliers
train_df_concat.drop(train_df_concat[train_df_concat['grid_distance']
> 10000].index, inplace=True)

fig, ax = plt.subplots(1, 1, figsize=(8, 5))
ax.hist(train_df_concat['grid_distance'], bins=300)
plt.show()
```

Figure 9: Find Outliers

I found out that there were only six observations that were larger than 10000. Also, the largest one was about 20 times larger than the second largest observation. At the same time, most of the data in the histogram was compressed in a narrow range. (see Figure 10 and 11)

```
8709.000000      1
9197.000000      1
9198.000000      1
9237.000000      1
9357.000000      1
9785.000000      1
9902.000000      1
10341.000000     1
10462.000000     1
11664.000000     1
13342.000000     1
19005.998460     1
429173.000000    1
```

Figure 10: Six Observations of "grid_distance" were larger than 10000.

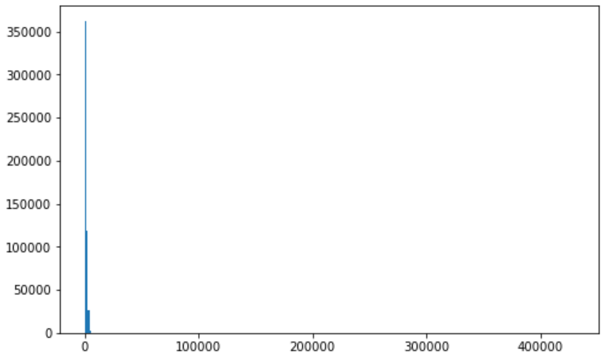

Figure 11: The histogram was compressed in a narrow range.

Thus, I decided to delete the observations whose "grid_distance" is larger than 10000.

Afterwards, the variable and the histogram looked better.

```
7930.000000      1
7955.000000      1
7974.000000      1
8071.000000      1
8356.000000      1
8421.000000      1
8709.000000      1
9197.000000      1
9198.000000      1
9237.000000      1
9357.000000      1
9785.000000      1
9902.000000      1

In [8]:
```

Figure 12: The Treated "grid_distance"

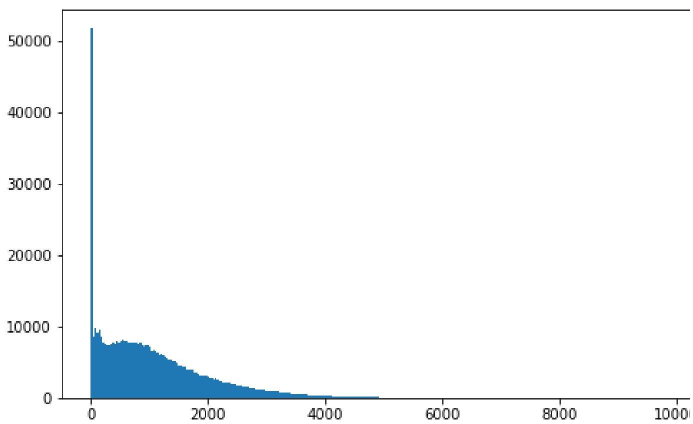

Figure 13: The Treated "grid_distance" looked better in histogram

Next, I ran the Decision Tree model again. At this time, I uploaded my prediction onto Kaggle and received a score of 0.75. Then, my base line model had a score of 0.75.

I also tried to use grid search and the learning curve of hyperparameter (see Figure 14). However, my laptop cannot handle such a large amount of computing. So, I gave up these two methods of adjusting parameters.

```
# learning curve of hyperparameter
superpa = []
for i in range(20,210,10):
    rfc = RandomForestClassifier(n_estimators=50, criterion='gini',
max_depth=20, min_samples_split=25, n_jobs=-1, random_state=0)
    rfc=rfc.fit(Xtrain,Ytrain)
    y_pred = rfc.predict(Xvalid)  # make prediction
    f1 = f1_score(Yvalid, y_pred, average='binary')
    superpa.append(f1)
print(max(superpa),superpa.index(max(superpa)))
plt.figure(figsize=[20,5])
plt.plot(range(20,210,10),superpa)
plt.show()


# This action takes too long time
# grid search
# set parameters
parameters = {'splitter':('best','random'),'criterion':
("gini","entropy"),"max_depth":[*range(3,14,5)],'min_samples_leaf':
[*range(1,10,4)],'min_impurity_decrease':[*np.linspace(0,0.5,3)]}
# search
clf = tree.DecisionTreeClassifier(random_state=25)
GS = GridSearchCV(clf, parameters, scoring='f1', n_jobs=-1,cv=3)
GS.fit(Xtrain,Ytrain)
GS.best_params_
GS.best_score_
```

Figure 14: Do Grid Search and Make Learning Curve of Hyperparameter

Afterwards, I built and ran a Random Forest model. Similarly, I fine-tuned it. Specially, I set n_jobs=-1 to let all the cores of the CPU participate in parallel computing. (see Figure 15)

As a result, I got a score of 0.79 on Kaggle.

```
# Method 2: Random Forest ---0.79446
# fit
rfc = RandomForestClassifier(n_estimators=75, criterion='gini',
max_depth=50, min_samples_split=25, n_jobs=-1, random_state=0)
rfc=rfc.fit(Xtrain,Ytrain)

# judge
score=rfc.score(Xvalid,Yvalid)
print(score)
y_pred = rfc.predict(Xvalid)  # make prediction
f1 = f1_score(Yvalid, y_pred, average='binary')
print(f1)

# predict
y_pred = rfc.predict(Xtest)
classification_df['action_type_DELIVERY']=y_pred
classification_df.to_csv(r'D:\machine-learning-for-business-
classification\Classification-Mingcong-RF.csv')
```

Figure 15: Random Forest Model

Finally, I tried XGBoost. Similarly, I fine-tuned it. (see Figure 16)

```
# method 3: XGBoost --- 0.89806, I chose this model
# load data
dtrain = xgb.DMatrix(Xtrain, label=Ytrain)
dvalid = xgb.DMatrix(Xvalid, label=Yvalid)
dtest = xgb.DMatrix(Xtest)
# Setting Parameters
param = {'max_depth': 15, 'eta': 0.2, 'objective': 'binary:logistic'}
param['nthread'] = 12
param['eval_metric'] = 'auc'
evallist = [(dvalid, 'eval'), (dtrain, 'train')]
# Training
num_round = 100
bst = xgb.train(param, dtrain, num_round, evallist)
# judge
ypred = bst.predict(dvalid)
ypred=ypred>0.5
ypred=ypred+0
f1 = f1_score(Yvalid, ypred, average='binary')
print(f1)
# predict
y_pred = bst.predict(dtest)
y_pred=y_pred>0.5
y_pred=y_pred+0
classification_df['action_type_DELIVERY']=y_pred
classification_df.to_csv(r'D:\machine-learning-for-business-
classification\Classification-Mingcong-XGB.csv')
```

Figure 16: Build and Run XGBoost Model

As a result, I got a score of 0.898 on Kaggle (see Figure 17). Thus, I decided to use the prediction of XGBoost model as my final outcome for classification task.

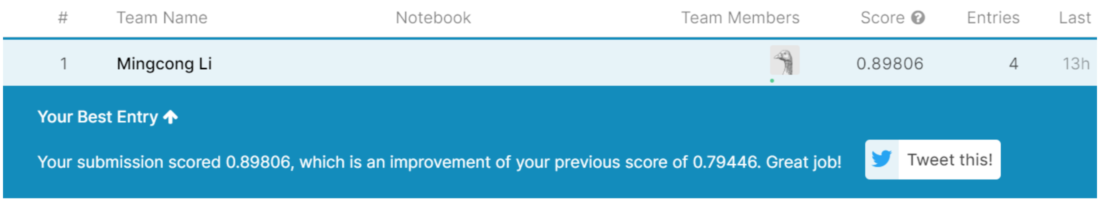

Figure 17: The Prediction of XGBoost Model Got a Score of 0.989 on Kaggle

4. Regression Task

According to the attempts in classification task, I decided to choose XGBoost as my model for regression task directly. Also, I fine-tuned it. (see Figure 18)

```
# Q2: regression
Ytrain = train_df_concat.iloc[:,-11]
Xtrain, Xvalid, Ytrain, Yvalid =train_test_split(Xtrain, Ytrain,
test_size=0.33, random_state=42)

#
# load data
dtrain = xgb.DMatrix(Xtrain, label=Ytrain)
dvalid = xgb.DMatrix(Xvalid, label=Yvalid)
dtest = xgb.DMatrix(Xtest)

# Setting Parameters
param = {'max_depth': 15, 'eta': 0.05, 'objective': 'reg:linear'}
param['nthread'] = 12
param['eval_metric'] = 'mae'
param['booster'] = 'gbtree'
evallist = [(dvalid, 'eval'), (dtrain, 'train')]
# Training
num_round = 41
bst = xgb.train(param, dtrain, num_round, evallist)

# predict
y_pred = bst.predict(dtest)
regression_df['expected_use_time']=y_pred
regression_df.to_csv(r'D:\machine-learning-for-business-
regression\Regression-Mingcong-XGB.csv')
```

Figure 18: Use XGBoost to Do Regression

As a result, I got a score (MAE) of 181.04 on Kaggle. (see Figure 19)

#	Team Name	Notebook	Team Members	Score	Entries	Last
1	Mingcong Li			181.04299	1	12h

Your First Entry

Welcome to the leaderboard!

Figure 19: The Prediction of XGBoost Model Got a Score of 181.04 on Kaggle