kaggle系列（3）：Rental Listing Inquiries（二）：XGBoost

2017-06-13 | Kaggle | 91

上一节我们对数据集进行了初步的探索，并将其可视化，对数据有了初步的了解。这样我们有了之前数据探索的基础之后，就有了对其建模的基础feature，结合目标变量，即可进行模型训练了。我们使用交叉验证的方法来判断线下的实验结果，也就是把训练集分成两部分，一部分是训练集，用来训练分类器，另一部分是验证集，用来计算损失评估模型的好坏。

在Kaggle的希格斯子信号识别竞赛中，XGBoost因为出众的效率与较高的预测准确度在比赛论坛中引起了参赛选手的广泛关注，在1700多支队伍的激烈竞争中占有一席之地。随着它在Kaggle社区知名度的提高，最近也有队伍借助XGBoost在比赛中夺得第一。其次，因为它的效果好，计算复杂度不高，也在工业界中有大量的应用。

今天，我们就先来跑一个XGBoost版的Base Model。先回顾一下XGBoost的原理吧：机器学习算法系列（8）：XgBoost

一、 准备工作

首先我们导入需要的包：

```
import os
import sys
import operator
import numpy as np
import pandas as pd
from scipy import sparse
import xgboost as xgb
from sklearn import model_selection,preprocessing,ensemble
from sklearn.metrics import log_loss
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
```

其中一些包的用途会在之后具体用到的时候进行讲解。

导入我们的数据：

```
data_path = '../data/'
train_file = data_path + "train.json"
test_file = data_path +"test.json"
train_df = pd.read_json(train_file)
test_df = pd.read_json(test_file)
print train_df.shape
print test_df.shape

(49352, 15)
(74659, 14)
```

查看一下前两行：

```
train_df.head(2)
```

	bathrooms	bedrooms	building_id	created	description	display_address	features	interest_level
10	1.5	3	53a5b119ba8f7b61d4e010512e0dfc85	2016-06-24 07:54:24	A Brand New 3 Bedroom 1.5 bath ApartmentEnjoy ...	Metropolitan Avenue	[]	medium
10000	1.0	2	c5c8a357cba207596b04d1afd1e4f130	2016-06-12 12:19:27		Columbus Avenue	[Doorman, Elevator, Fitness Center, Cats Allow...	low

latitude	listing_id	longitude	manager_id	photos	price	street_address
40.7145	7211212	-73.9425	5ba989232d0489da1b5f2c45f6688adc	[https://photos.renthop.com/2/7211212_1ed4542e...	3000	792 Metropolitan Avenue
40.7947	7150865	-73.9667	7533621a882f71e25173b27e3139d83d	[https://photos.renthop.com/2/7150865_be3306c5...	5465	808 Columbus Avenue

二、特征构建

我们不需要对数值型数据进行任何的预处理，所以首先建立一个数值型特征的列表，纳入features_to_use

```
features_to_use = ["bathrooms","bedrooms","latitude","longitude","price"]
```

现在让我们根据已有的一些特征来构建一些新的特征：

```
# 照片数量(num_photos)
train_df['num_photos']=train_df['photos'].apply(len)
test_df['num_photos']=train_df['photos'].apply(len)

# 特征数量
train_df['num_features']=train_df['features'].apply(len)
test_df['num_features']=test_df['features'].apply(len)

# 描述词汇数量
train_df['num_description_words'] = train_df['description'].apply(lambda x: len(x.split
test_df['num_description_words'] = test_df['description'].apply(lambda x: len(x.split("

#把创建的时间分解为多个特征
train_df['created']=pd.to_datetime(train_df['created'])
test_df['created']=pd.to_datetime(test_df['created'])

#让我们从时间中分解出一些特征，比如年，月，日，时
#年
train_df['created_year'] = train_df['created'].dt.year
test_df['created_year'] = test_df['created'].dt.year
#月
train_df['created_month'] = train_df['created'].dt.month
test_df['created_month'] = test_df['created'].dt.month
#日
train_df['created_day'] = train_df['created'].dt.day
test_df['created_day'] = test_df['created'].dt.day
#时
train_df['created_hour'] = train_df['created'].dt.hour
test_df['created_hour'] = test_df['created'].dt.hour

#把这些特征都放到所需特征列表中（上面已经创建，并加入了数值型特征）
features_to_use.extend(["num_photos","num_features","num_description_words","created_ye
```

我们有四个分类型的特征：

- display_address
- manager_id
- building_id
- street_address

可以对它们分别进行特征编码：

```
categorical = ["display_address","manager_id",'building_id',"street_address"]
for f in categorical:
    if train_df[f].dtype == 'object':
        lbl = preprocessing.LabelEncoder()
        lbl.fit(list(train_df[f].values)+list(test_df[f].values))
        train_df[f] = lbl.transform(list(train_df[f].values))
        test_df[f] = lbl.transform(list(test_df[f].values))
        features_to_use.append(f)
```

还有一些字符串类型的特征，可以先把它们合并起来

```
train_df["features"] = train_df["features"].apply(lambda x:" ".join(["_".join(i.split("
print train_df['features'].head(2)
test_df['features'] = test_df["features"].apply(lambda x: " ".join(["_".join(i.split(" "
print test_df['features'].head(2)
```

得到的字符串结果如下：

10000 Doorman Elevator Fitness_Center Cats_Allowed D...

100004 Laundry_In_Building Dishwasher Hardwood_Floors...

然后CountVectorizer类来计算TF-IDF权重

```
tfidf = CountVectorizer(stop_words ="english",max_features=200)
tr_sparse = tfidf.fit_transform(train_df["features"])
te_sparse = tfidf.transform(test_df["features"])
```

这里我们需要提一点，对数据集进行特征变换时，必须同时对训练集和测试集进行操作。现在把这些处理过的特征放到一个集合中（横向合并）

```
train_X = sparse.hstack([train_df[features_to_use],tr_sparse]).tocsr()
test_X = sparse.hstack([test_df[features_to_use],te_sparse]).tocsr()
```

然后把目标变量转换为0、1、2，如下

```
target_num_map = {'high':0 , 'medium':1 , 'low':2}
train_y = np.array(train_df['interest_level'].apply(lambda x: target_num_map[x]))
print train_X.shape,test_X.shape

(49352, 217) (74659, 217)
```

可以看到，经过上面一系列的变量构造之后，其数量已经达到了217个。

接下来就可以进行建模啦。

三、XGB建模

先写一个通用的XGB模型的函数：

```
def runXGB(train_X,train_y,test_X,test_y=None,feature_names=None,seed_val=0,num_rounds=
    #参数设定
    param = {}
    param['objective'] = 'multi:softprob'#多分类、输出概率值
    param['eta'] = 0.1#学习率
    param['max_depth'] = 6#最大深度，越大越容易过拟合
    param['silent'] = 1#打印提示信息
    param['num_class'] = 3#三个类别
    param['eval_metric']= "mlogloss"#对数损失
    param['min_child_weight']=1#停止条件，这个参数非常影响结果，控制叶子节点中二阶导的和的最
    param['subsample'] =0.7#随机采样训练样本
    param['colsample_bytree'] = 0.7# 生成树时进行的列采样
    param['seed'] = seed_val#随机数种子
    num_rounds = num_rounds#迭代次数

    plst = list(param.items())
    xgtrain = xgb.DMatrix(train_X,label=train_y)

    if test_y is not None:
        xgtest = xgb.DMatrix(test_X,label=test_y)
        watchlist = [(xgtrain,'train'),(xgtest,'test')]
        model = xgb.train(plst,xgtrain,num_rounds,watchlist,early_stopping_rounds=20)
      #  early_stopping_rounds 当设置的迭代次数较大时，early_stopping_rounds 可在一定的迭代
    else:
        xgtest = xgb.DMatrix(test_X)
        model = xgb.train(plst,xgtrain,num_rounds)
    pred_test_y = model.predict(xgtest)
    return pred_test_y,model
```

函数返回的是预测值和模型。

5折交叉验证将训练集划分为五份，其中的一份作为验证集。

```
cv_scores = []
kf = model_selection.KFold(n_splits=5,shuffle=True,random_state=2016)

for dev_index,val_index in kf.split(range(train_X.shape[0])):
    dev_X,val_X = train_X[dev_index,:],train_X[val_index,:]
    dev_y,val_y = train_y[dev_index],train_y[val_index]
    pred,model = runXGB(dev_X,dev_y,val_X,val_y)
    cv_scores.append(log_loss(val_y,preds))
    print cv_scores
    break
```

结果如下：

```
[0]     train-mlogloss:1.04135  test-mlogloss:1.04229
Multiple eval metrics have been passed: 'test-mlogloss' will be used for early stopping

Will train until test-mlogloss hasn't improved in 20 rounds.
[1]     train-mlogloss:0.989004 test-mlogloss:0.99087
[2]     train-mlogloss:0.944233 test-mlogloss:0.947047
[3]     train-mlogloss:0.90536  test-mlogloss:0.908933
[4]     train-mlogloss:0.872054 test-mlogloss:0.876526
[5]     train-mlogloss:0.841783 test-mlogloss:0.847383
[6]     train-mlogloss:0.815921 test-mlogloss:0.822307
[7]     train-mlogloss:0.793337 test-mlogloss:0.800476
[8]     train-mlogloss:0.773562 test-mlogloss:0.781413
[9]     train-mlogloss:0.754927 test-mlogloss:0.76381
[10]    train-mlogloss:0.738299 test-mlogloss:0.747959

......
......
[367]   train-mlogloss:0.348196 test-mlogloss:0.548011
[368]   train-mlogloss:0.347768 test-mlogloss:0.547992
[369]   train-mlogloss:0.347303 test-mlogloss:0.548021
[370]   train-mlogloss:0.346807 test-mlogloss:0.548065
[371]   train-mlogloss:0.346514 test-mlogloss:0.548079
[372]   train-mlogloss:0.34615  test-mlogloss:0.548097
[373]   train-mlogloss:0.345859 test-mlogloss:0.548111
[374]   train-mlogloss:0.345377 test-mlogloss:0.548081
[375]   train-mlogloss:0.344961 test-mlogloss:0.548068
[376]   train-mlogloss:0.344493 test-mlogloss:0.548024
[377]   train-mlogloss:0.344086 test-mlogloss:0.547975
Stopping. Best iteration:
[357]   train-mlogloss:0.352182 test-mlogloss:0.547867
```

迭代357次之后，在训练集上的对数损失为0.352182，在验证集上的损失为0.5478。

然后在对测试集进行预测：

```
preds,model=runXGB(train_X,train_y,test_X,num_rounds=400)
```

把结果按照比赛规定的格式写入csv文件：

```
out_df = pd.DataFrame(preds)
out_df.columns = ["high", "medium", "low"]
out_df["listing_id"] = test_df.listing_id.values
out_df.to_csv("xgb_starter2.csv", index=False)
```

看一下最后的结果：

	high	medium	low	listing_id
0	0.015237	0.198407	0.786357	7142618
1	0.014637	0.049824	0.935538	7210040
2	0.002438	0.064852	0.932710	7103890
3	0.021119	0.480018	0.498863	7143442
4	0.008035	0.068569	0.923395	6860601
5	0.000116	0.019608	0.980276	6840081

提交到kaggle上，这样我们整个建模的过程就完成了。

接下来两节中，我们重点讲一讲关于XGBoost的调参经验以及使用SK-learn计算TF-IDF。

kaggle # XGBoost # 特征工程

‹ kaggle系列（2）：Rental Listing Inquiries（一）：EDA

kaggle系列（4）：Rental Listing Inquiries（三）：XGBoost调参指南 ›