Team Notebook XGBoost

Input Files

- Zillow Prize: Zillow's Home Value Prediction (Zestimate)
  - properties_2016.csv
  - properties_2017.csv
  - sample_submission.csv
  - train_2016_v2.csv
  - train_2017.csv
  - zillow_data_dictionary.xlsx

+ Add Data Source

Upload Dataset

Zillow Prize: Zillow's Home Value Prediction (Zestimate)

Can you improve the algorithm that changed the world of real estate?

Last Updated 7 months ago

Remove

About this Dataset

In this competition, Zillow is asking you to predict the log-error between their Zestimate and the actual sale price, given all the features of a home. The log error is defined as

$$logerror = log(Zestimate) - log(SalePrice)$$

and it is recorded in the transactions file **train.csv**. In this competition, you are going to predict the logerror for the months in Fall 2017. Since all the real estate transactions in the U.S. are publicly available, we will close the competition (no longer accepting submissions) before the evaluation period begins.

Train/Test split

- You are provided with a full list of real estate properties in three counties (Los Angeles, Orange and Ventura, California) data in 2016.
- The train data has all the transactions before October 15, 2016, plus some of the transactions after October 15, 2016.
- The test data in the public leaderboard has the rest of the transactions between October 15 and December 31, 2016.
- The rest of the test data, which is used for calculating the private leaderboard, is **all** the properties in October 15, 2017, to December 15, 2017. This period is called the "sales tracking period", during which we will not be taking any submissions.
- You are asked to predict 6 time points for **all** properties: **October 2016 (201610), November 2016 (201611), December 2016 (201612), October 2017 (201710), November 2017 (201711), and December 2017 (201712).**
- Not all the properties are sold in each time period. If a property was not sold in a certain time period, that particular row will be ignored when calculating your score.
- If a property is sold multiple times within 31 days, we take the first reasonable value as the ground truth. By "reasonable", we mean if the data seems wrong, we will take the transaction that has a value that makes more sense.

File descriptions

- **properties_2016.csv -** all the properties with their home features for 2016. Note: Some 2017 new properties don't have any data yet except for their

parcelid's. Those data points should be populated when **properties_2017.csv** is available.

- **properties_2017.csv -** all the properties with their home features for 2017 (released on 10/2/2017)
- **train_2016.csv -** the training set with transactions from 1/1/2016 to 12/31/2016
- **train_2017.csv -** the training set with transactions from 1/1/2017 to 9/15/2017 (released on 10/2/2017)
- **sample_submission.csv -** a sample submission file in the correct format

Data fields

- Please refer to **zillow_data_dictionary.xlsx**

Zillow-kaggle data competition And NYU ML Final Project

https://www.kaggle.com/c/zillow-prize-1 (https://www.kaggle.com/c/zillow-prize-1)

Team : CrusaderEmp

- **Can Xu : cx461@nyu.edu**
- **Guanyu Zhu : gz623@nyu.edu**

Before we dive deep into the data, let us know a little more about the competition.

Zillow (https://www.zillow.com/) is an online real estate database company founded in 2006 - Wikipedia

Zestimate:

“Zestimates” are estimated home values based on 7.5 million statistical and machine learning models that analyze hundreds of data points on each property. And, by continually improving the median margin of error (from 14% at the onset to 5% today)

Objective:

Building a model to improve the Zestimate residual error.

[1]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input dire

from subprocess import check_output
print(check_output(["ls", "../input"]).decode("utf8"))

# Any results you write to the current directory are saved as output.
```

```
properties_2016.csv
properties_2017.csv
sample_submission.csv
train_2016_v2.csv
train_2017.csv
zillow_data_dictionary.xlsx
```

[2]:
```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import gc
import pandas as pd
import xgboost as xgb

import seaborn as sns

print("import all module ...")
```

```
color = sns.color_palette()

%matplotlib inline

pd.options.mode.chained_assignment = None
pd.options.display.max_columns = 999
```

```
import all module ...
```

[3]:

```
print('Loading data ...')
```



```
sample = pd.read_csv('../input/sample_submission.csv')
```

```
Loading data ...

/opt/conda/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2717: DtypeWarning: Columns (22,3
2,34,49,55) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Show what's the input like

[4]:

```
print('Binding to float32')

for c, dtype in zip(prop.columns, prop.dtypes):
    if dtype == np.float64:
        prop[c] = prop[c].astype(np.float32)
print(train.shape)
train.head()
```

```
Binding to float32
(90275, 3)
```

[4]:

	parcelid	logerror	transactiondate
0	11016594	0.0276	2016-01-01
1	14366692	-0.1684	2016-01-01
2	12098116	-0.0040	2016-01-01
3	12643413	0.0218	2016-01-02
4	14432541	-0.0050	2016-01-02

[5]:

```
train.tail(3)
```

[5]:

	parcelid	logerror	transactiondate
90272	12995401	-0.2679	2016-12-30
90273	11402105	0.0602	2016-12-30
90274	12566293	0.4207	2016-12-30

[6]:
```
print(prop.shape)
prop.head()
```

```
(2985217, 58)
```

[6]:

	parcelid	airconditioningtypeid	architecturalstyletypeid	basementsqft	bathroomcnt	bedroomcnt	buildingclasstypeid	buildingqualityt
0	10754147	NaN	NaN	NaN	0.0	0.0	NaN	NaN
1	10759547	NaN	NaN	NaN	0.0	0.0	NaN	NaN
2	10843547	NaN	NaN	NaN	0.0	0.0	NaN	NaN
3	10859147	NaN	NaN	NaN	0.0	0.0	3.0	7.0
4	10879947	NaN	NaN	NaN	0.0	0.0	4.0	NaN

[7]:
```
prop.tail(3)
```

[7]:

	parcelid	airconditioningtypeid	architecturalstyletypeid	basementsqft	bathroomcnt	bedroomcnt	buildingclasstypeid	buildin
2985214	168040630	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2985215	168040830	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2985216	168040430	NaN	NaN	NaN	NaN	NaN	NaN	NaN

[8]:
```
print(sample.shape)
sample.head()
```

```
(2985217, 7)
```

[8]:

	ParcelId	201610	201611	201612	201710	201711	201712
0	10754147	0	0	0	0	0	0
1	10759547	0	0	0	0	0	0
2	10843547	0	0	0	0	0	0
3	10859147	0	0	0	0	0	0
4	10879947	0	0	0	0	0	0

[9]:
```
sample.tail(3)
```

[9]:

	ParcelId	201610	201611	201612	201710	201711	201712
2985214	168040630	0	0	0	0	0	0
2985215	168040830	0	0	0	0	0	0
2985216	168040430	0	0	0	0	0	0

Y is target label (logerror)

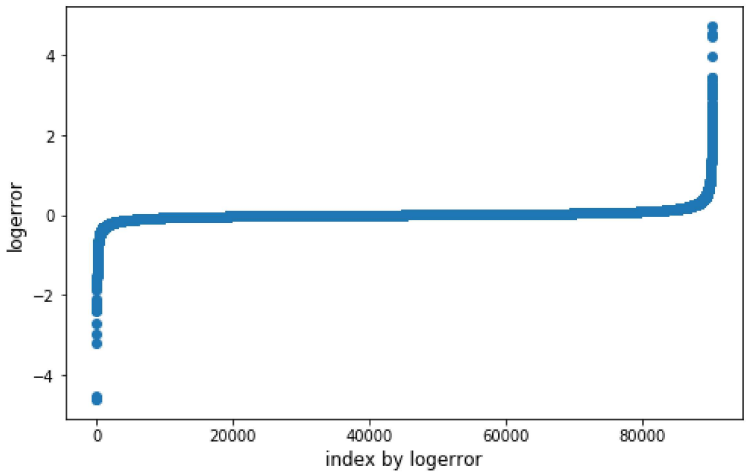
X need to elimate non-numeric type

[10]:
```
print('Creating training set ...')

df_train = train.merge(prop, how='left', on='parcelid')

x_train = df_train.drop(['parcelid', 'logerror', 'transactiondate', 'propertyzoningdesc', 'propertycountyla
y_train = df_train['logerror'].values
```

```
Creating training set ...
```


Target variable for this competition is "logerror" field. So let us do some analysis on this field first.

[11]:
```
plt.figure(figsize=(8,5))
plt.scatter(range(y_train.shape[0]), np.sort(y_train))
plt.xlabel('index by logerror', fontsize=12)
plt.ylabel('logerror', fontsize=12)
plt.show()
```



Split all Training Data into Train and Test data for Cross-Validation

[12]:
```
print(x_train.shape, y_train.shape)

train_columns = x_train.columns

for c in x_train.dtypes[x_train.dtypes == object].index.values:
    x_train[c] = (x_train[c] == True)

del df_train; gc.collect()

split = 80000
x_train, y_train, x_valid, y_valid = x_train[:split], y_train[:split], x_train[split:], y_train[split:]
```

```
(90275, 55) (90275,)
```

Use **XGBOOST** to train, since it is more powerful than sklearn in competition

[13]:
```
print('Building DMatrix...')

d_train = xgb.DMatrix(x_train, label=y_train)
d_valid = xgb.DMatrix(x_valid, label=y_valid)

del x_train, x_valid; gc.collect()
```

```
Building DMatrix...
```


Metrics : Mae not RMSE , since ZillowMae Score

ETA (learning rate) = 0.05

subsample and colsample_bytree to decrease over-fitting

Those all were based on CV gird-search

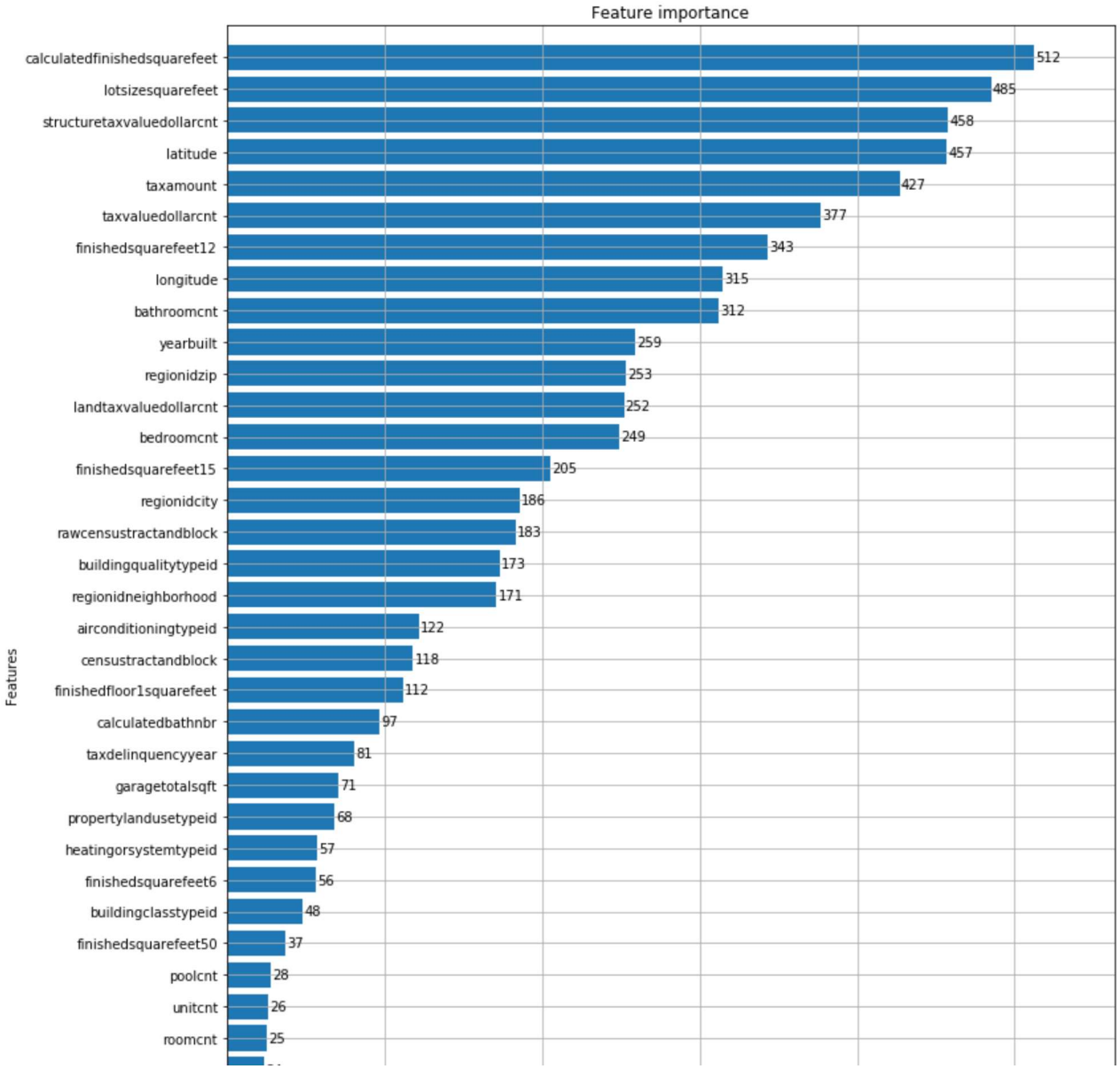
[14]:
```
print('Training ...')


xgb_params = {
    'eval_metric':'mae',   # mae since ZillowMae Score , rmse as default
    'eta':0.05,             # laerning rate by grid search
    'max_depth': 6,
    'subsample': 0.7,           # decrease over-fitting
    'colsample_bytree': 0.7,    #
    'objective': 'reg:linear',
    'silent': 1,  # on
    'seed' : 0
}


watchlist = [(d_train, 'train'), (d_valid, 'valid')]
clf = xgb.train(xgb_params, d_train, 10000, watchlist, early_stopping_rounds=100, verbose_eval=10)
```

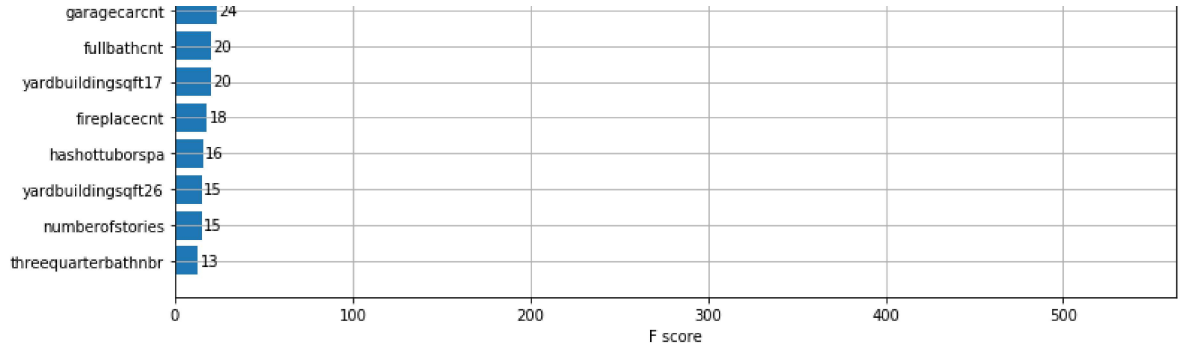
```
Training ...
[0]     train-mae:0.473647      valid-mae:0.466725
Multiple eval metrics have been passed: 'valid-mae' will be used for early stopping.

Will train until valid-mae hasn't improved in 100 rounds.
[10]    train-mae:0.292472      valid-mae:0.285894
[20]    train-mae:0.187569      valid-mae:0.181947
[30]    train-mae:0.128486      valid-mae:0.1238
[40]    train-mae:0.096615      valid-mae:0.092716
[50]    train-mae:0.080704      valid-mae:0.077597
[60]    train-mae:0.073306      valid-mae:0.070987
[70]    train-mae:0.070015      valid-mae:0.068339
[80]    train-mae:0.068535      valid-mae:0.067304
[90]    train-mae:0.06784       valid-mae:0.066991
```

```
[100]   train-mae:0.067459      valid-mae:0.066894
[110]   train-mae:0.067209      valid-mae:0.066888
[120]   train-mae:0.067065      valid-mae:0.066924
[130]   train-mae:0.066951      valid-mae:0.066924
[140]   train-mae:0.066847      valid-mae:0.066968
[150]   train-mae:0.06677       valid-mae:0.066995
[160]   train-mae:0.066702      valid-mae:0.06709
[170]   train-mae:0.066587      valid-mae:0.06716
[180]   train-mae:0.066508      valid-mae:0.067213
[190]   train-mae:0.066432      valid-mae:0.067296
[200]   train-mae:0.066335      valid-mae:0.067354
Stopping. Best iteration:
[106]   train-mae:0.06731       valid-mae:0.066869
```


Show the importance of different feature

[15]:
```
# plot the important features #
fig, ax = plt.subplots(figsize=(12,18))
xgb.plot_importance(clf, max_num_features=40, height=0.8, ax=ax)
plt.show()
```





[16]:
```
del d_train, d_valid
```


[17]:
```
print('Building test set ...')

sample['parcelid'] = sample['ParcelId']
df_test = sample.merge(prop, on='parcelid', how='left')

del prop; gc.collect()

x_test = df_test[train_columns]
for c in x_test.dtypes[x_test.dtypes == object].index.values:
    x_test[c] = (x_test[c] == True)

del df_test, sample; gc.collect()

d_test = xgb.DMatrix(x_test)

del x_test; gc.collect()
```

```
Building test set ...
```

[17]:
```
7
```

[18]:
```
print('Predicting on test ...')

p_test = clf.predict(d_test)

del d_test; gc.collect()
```

```
Predicting on test ...
```

[18]:
```
0
```

[19]:
```
sub = pd.read_csv('../input/sample_submission.csv')
for i,c in enumerate(sub.columns):
    if i > 0 :
        sub[c] = p_test

sub.head(6)
```

[19]:

	ParcelId	201610	201611	201612	201710	201711	201712
0	10754147	0.271763	0.271763	0.271763	0.271763	0.271763	0.271763
1	10759547	0.036884	0.036884	0.036884	0.036884	0.036884	0.036884
2	10843547	0.164364	0.164364	0.164364	0.164364	0.164364	0.164364
3	10859147	0.602909	0.602909	0.602909	0.602909	0.602909	0.602909
4	10879947	0.240830	0.240830	0.240830	0.240830	0.240830	0.240830
5	10898347	0.076973	0.076973	0.076973	0.076973	0.076973	0.076973

[20]:

```
sub.tail(3)
```


	ParcelId	201610	201611	201612	201710	201711	201712
2985214	168040630	0.661705	0.661705	0.661705	0.661705	0.661705	0.661705
2985215	168040830	0.661705	0.661705	0.661705	0.661705	0.661705	0.661705
2985216	168040430	0.661705	0.661705	0.661705	0.661705	0.661705	0.661705

[27]:

```
import datetime

now = datetime.datetime.now()
print( now )

sub.to_csv('sub{0:s}.csv'.format(now.strftime('%Y%m%d_%H%M%S')), index=False, float_format='%.4f')

print("Saved to sub{0:s}.csv".format( now.strftime('%Y%m%d_%H%M%S')) );
```

```
2017-12-13 02:43:59.520347
Saved to sub20171213_024359.csv
```