mymain

October 18, 2018

```
In [25]: import pandas as pd
         import numpy as np
         from glmnet import LogitNet
         from sklearn import datasets, linear_model
         from sklearn.model_selection import train_test_split
         from matplotlib import pyplot as plt
         from sklearn.datasets import load_svmlight_files
         from sklearn.metrics import accuracy_score
         import xgboost as xgb
         import csv
         from xgboost import XGBRegressor
         from sklearn import preprocessing
         import seaborn as sns
         import math
         from sklearn.ensemble import RandomForestRegressor
         from sklearn.preprocessing import Imputer
         from sklearn.ensemble import GradientBoostingRegressor
         from sklearn.preprocessing import StandardScaler
```

0.0.1 Computer system spec

MacBook Air (13-inch, Early 2014)
Processor 1.4 GHz Intel Core i5
Memory 4 GB 1600 MHz DDR3

0.0.2 Load the dataset Ames_data.csv

```
In [41]: # read csv file
         df = pd.read_csv('/Users/glen/Desktop/Statistical learning/Project/Ames_data.csv')
```

```
In [40]: # Descriptive statistics for each column
         # df.describe()
```

```
In [28]: # df shape
         testnum = int(round(df.shape[0]*0.3 , 0))
         print(df.shape)
         print("Test number:", testnum)
```

```
#make sure PID is unique
print("unique PID number",pd.unique(df['PID']).size)
```

```
(2930, 83)
Test number: 879
unique PID number 2930
```

0.0.3 Split the data into 70% training and 30% testing dataset

```
In [29]: # Split data in input and output
         original_y = df['Sale_Price']
         original_X = df.drop(columns = ['Sale_Price'])

         #Split data in train and test set
         X_train, X_test, y_train, y_test = train_test_split(
             original_X, original_y, test_size=0.3,random_state=12)

         print("Train dataset contains {0} rows and {1} columns".
               format(X_train.shape[0], X_train.shape[1]))
         print("Test dataset contains {0} rows and {1} columns".
               format(X_test.shape[0], X_test.shape[1]))

         #Save the train.csv and test.csv file

         X_train.to_csv('train.csv', index = False)
         X_test.join(y_test).to_csv('test.csv', index = False)

         # pd.merge(X_test,y_test,left_on='index',right_on='index')
```

```
Train dataset contains 2051 rows and 82 columns
Test dataset contains 879 rows and 82 columns
```

0.0.4 Data preprocessing

Because I try to use one-hot encoding later. In order to decrease the number of categories. First I delete the column if its categories exceed 15. Training dataset and test dataset would get different number of columns while doing one-hot encoding, so I need to align the X_train and X_test together to get the same number of column. Then I use imputer to replace the Nan value by the mean value.

```
In [30]: #data preprocessing

         #delete column 'Street', 'Longtitude', 'Latitude'
         X_train = X_train.drop(columns = ['Street','Longitude','Latitude','Garage_Yr_Blt'])
         X_test = X_test.drop(columns = ['Street','Longitude','Latitude','Garage_Yr_Blt'])

         #Choose column
```

```
choose_column = [col for col in X_train.columns if
                 (X_train[col].nunique() < 15 and X_train[col].dtype == "object")
                 or X_train[col].dtype in ['int64','float64']]

X_train = X_train[choose_column]
X_test = X_test[choose_column]

print("Train dataset contains {0} rows and {1} columns".
      format(X_train.shape[0], X_train.shape[1]))
print("Test dataset contains {0} rows and {1} columns".
      format(X_test.shape[0], X_test.shape[1]))
```

```
Train dataset contains 2051 rows and 74 columns
Test dataset contains 879 rows and 74 columns
```

```
In [31]: #implement one hot encoding to get the result
         X_train = pd.get_dummies(X_train)
         X_test = pd.get_dummies(X_test)
         print("Train dataset after one hot contains {0} rows and {1} columns".
               format(X_train.shape[0], X_train.shape[1]))
         print("Test dataset after one hot contains {0} rows and {1} columns".
               format(X_test.shape[0], X_test.shape[1]))
```

```
Train dataset after one hot contains 2051 rows and 264 columns
Test dataset after one hot contains 879 rows and 256 columns
```

```
In [32]: # align X_train and X_test so column number of X_train and X_test could be the same

         X_train, X_test = X_train.align(X_test, join = 'left', axis=1)
         print("Train dataset contains {0} rows and {1} columns".
               format(X_train.shape[0], X_train.shape[1]))
         print("Test dataset contains {0} rows and {1} columns".
               format(X_test.shape[0], X_test.shape[1]))
```

```
Train dataset contains 2051 rows and 264 columns
Test dataset contains 879 rows and 264 columns
```

0.0.5 Label Encoder

I've tried to use label encoder first, but it turns out a bad predicted result compared to one hot encoding.

```
In [33]: ###### Label Encoder

         # def label_encoder(dataframe ,col_name):
         #     #Use label encoder
```

```
#     le = preprocessing.LabelEncoder()
#     le.fit(dataframe[col_name])

#     #create a new dataframe d then replace the original value
#     test = le.transform(dataframe[col_name])
#     d = {'col1': list(test)}
#     df = pd.DataFrame(data=d, dtype=np.int8)

#     #Replace original value by label_encoded value
#     dataframe[col_name]  = df.values
#     return None

# # X_train label_encoder transform
# for col in X_train.columns:
#     if X_train[col].dtype == "object":
#         label_encoder(X_train,col)

# # X_test label_encoder transform
# for col in X_test.columns:
#     if X_test[col].dtype == "object":
#         label_encoder(X_test,col)

# print("Train dataset contains {0} rows and {1} columns".
# format(X_train.shape[0], X_train.shape[1]))
# print("Test dataset contains {0} rows and {1} columns".
# format(X_test.shape[0], X_test.shape[1]))
```

In [34]:
```
#Use Imputer to deal with missing value or nan
#train_X and test_X are numpy array
my_imputer = Imputer(missing_values='NaN', strategy='mean')
train_X = my_imputer.fit_transform(X_train)
test_X = my_imputer.transform(X_test)
print("Train dataset contains {0} rows and {1} columns".
      format(train_X.shape[0], train_X.shape[1]))
print("Test dataset contains {0} rows and {1} columns".
      format(test_X.shape[0], test_X.shape[1]))
```

```
Train dataset contains 2051 rows and 264 columns
Test dataset contains 879 rows and 264 columns
```

0.0.6 Save X_train and y_train in csv files. Then I use glmnet in R to predict the best lambda.

In [35]:
```
# save X_train & X_test as csv file which are used to compute best lambda
# best lambda = 763.7335
X_train.fillna(0)
X_train.to_csv('X_train.csv', index = False)
y_train.to_csv('y_train.csv', index = False)
```

0.0.7 XGboost

XGBoost is an optimized distributed gradient boosting system designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework.

```
In [36]: #XGboost
         XGB = XGBRegressor()
         XGB.fit(train_X, y_train, verbose=False)
         pre_test_y = XGB.predict(test_X)
         ans = round(math.sqrt(np.mean(np.square(np.log(pre_test_y) -
                                          np.log(np.array(y_test))))), 4)
         print("Root-Mean-Squared-Error (RMSE) for XGBoost model is {0}".format(ans))
```

```
Root-Mean-Squared-Error (RMSE) for XGBoost model is 0.1288
```

```
In [37]: # export the result as mysubmission1.txt
         PID = np.array(X_test['PID'].values).reshape(len(X_test),1)
         Sale_Price = pre_test_y.reshape(len(pre_test_y),1)
         ans = np.concatenate((PID,Sale_Price),axis=1)

         # save the data with mixted datatype, %d saves PID as integer and
         # %10.5f can save the sale_price as double
         np.savetxt("mysubmission1", ans, delimiter=',', header="PID,Sale_Price",
                    fmt='%d %10.5f', comments='')
```

0.0.8 Gradient Boosted Regression

I've tried random forest algorithm in the beginning, but the RMSE was still high. So I use GBR which is similiar to XGboost.

```
In [38]: # Use Gradient Boosted Regression Trees
         gbrt=GradientBoostingRegressor(n_estimators=100)
         gbrt.fit(train_X, y_train)
         pre_test_y=gbrt.predict(test_X)
         ans = round(math.sqrt(np.mean(np.square(np.log(pre_test_y) -
                                          np.log(np.array(y_test))))), 4)
         print("Root-Mean-Squared-Error (RMSE) for GBRT model is {0}".format(ans))
```

```
Root-Mean-Squared-Error (RMSE) for GBRT model is 0.1263
```

```
In [39]: # export the result as mysubmission2.txt
         PID = np.array(X_test['PID'].values).reshape(len(X_test),1)
         Sale_Price = pre_test_y.reshape(len(pre_test_y),1)
         ans = np.concatenate((PID,Sale_Price),axis=1)

         # save the data with mixted datatype, %d saves PID as integer and
```

```python
# %10.5f can save the sale_price as double
np.savetxt("mysubmission2", ans, delimiter=',', header="PID,Sale_Price",
           fmt='%d %10.5f', comments='')
```