# mylasso

October 18, 2018

```
In [17]: 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 [18]: # read csv file
         df = pd.read_csv('/Users/glen/Desktop/Statistical learning/Project/Ames_data.csv')
         # df.head(5)

In [19]: # Descriptive statistics for each column
         # df.describe()

In [20]: # 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 [21]: # 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 [22]: #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 [23]:
```
#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 [24]:
```
# 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
```

In [25]:
```
#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
```

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

```
In [26]: # 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)
```

### 0.0.6 Coordinate Descent lasso algorithm

Use coordinate descent to calculate the result. Unfortunately, I can't get a decent result from this model. My smallest RMSE value is 0.48

```
In [27]: # implement Lasso using Coordinate Descent
         def one_step_lasso(r, x, lam):
             # x: nx1 matrix
             # r: nx1 matrix
             # Use the soft-thresholding result lasso_j = sgn(LS_j)(|LSj|)+
             # return beta_j
             xx = np.sum(np.square(x))
             xr = np.sum(x*r)
             b = (abs(xr) -lam/2)/xx
             b = np.sign(xr)*(b if b>0 else 0)
             return b

         def mylasso(X, y, lam, n_iter, standardize = True):
             # X: n-by-p design matrix without the intercept
             # y: n-by-1 response vector
             # p: p-by-1 vector
             # lam: lambda value
             # n.iter: number of iterations
             # standardize: if True, center and scale X and y.

             #p is the number of features (columns)
             p = X.shape[1]

             # n is the number of rows
             n = X.shape[0]

             # YOUR CODE
             # If standardize  = TRUE, center and scale X and Y
             # record the corresponding means and sd
```

```
    # scaler = StandardScaler()
    # X = scaler.fit_transform(X)
    # y = scaler.fit_transform(y)
    if standardize == True:
        mean_y = np.mean(y)
        mean_X = np.mean(X,axis = 0)
        sd_y = np.std(y)
        sd_X = np.std(X,axis = 0)
        X = (X - np.mean(X,axis = 0))/np.std(X,axis = 0)
        y = (y - np.mean(y))/np.std(y)


    # Initial values for residual and coefficient vector b
    # b: p-by-1 vector, without intercept

    b = np.zeros(p)
    r = y


    for step in range(n_iter):
        for j in range(p):
        # YOUR CODE
        # 1) Update the residual vector to be the one
        # in blue on p37 of [lec_W3_VariableSelection.pdf].
        # r <-- current residual + X[, j] * b[j]
        # r is n-by-1 vector

            r = r + X[:,j]*b[j]
        # 2) Apply one_step_lasso to update beta_j
        # b[j] = one_step_lasso(r, X[, j], lam)
            b[j] = one_step_lasso(r, X[:,j], lam)

#           # 3) Update the current residual vector
#           # r <-- r - X[, j] * b[j]
            r = r - X[:,j]*b[j]
    # YOUR CODE: scale back b and add intercept b0
    # For b0, check p13 of [lec_W3_VariableSelection.pdf].

    b = (sd_y/sd_X)*b
    b_intercept = mean_y - np.dot(mean_X.T,b)
    return (b_intercept,b)
```

In [28]:
```
lam = 1000
X = train_X
y = y_train
n_iter = 50
standardize = True
b_intercept = mylasso(X, y, lam, n_iter)[0]
```

```
b = mylasso(X, y, lam, n_iter)[1].reshape(264,1)
# np.unique(b)
```

In [29]:
```
pre_test_y = np.dot(test_X,b) + b_intercept
ans = round(math.sqrt(np.mean(np.square(
    np.log(pre_test_y) - np.array(np.log(y_test))))), 4)
print("Root-Mean-Squared-Error (RMSE) for Coordinate decent lasso model is {0}".
      format(ans))
```

```
Root-Mean-Squared-Error (RMSE) for Coordinate decent lasso model is 0.4684
```

In [30]:
```
# export the result as mysubmission3.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("mysubmission3", ans, delimiter=',',
           header="PID,Sale_Price", fmt='%d %10.5f', comments='')
```