04_RidgeAndLasso_v1.0-PauloBraga

June 29, 2020

[1]:
```
''' Paulo Simplício Braga
    29.06.2020
'''

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib

import matplotlib.pyplot as plt
from scipy.stats import skew
from scipy.stats.stats import pearsonr
from sklearn import datasets,linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

from sklearn.linear_model import Ridge, RidgeCV, ElasticNet, LassoCV,
 ↪LassoLarsCV
from sklearn.model_selection import cross_val_score

%config InlineBackend.figure_format = 'retina' #set 'png' here when working on
 ↪notebook
%matplotlib inline
```

[2]:
```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

I - Preparação dos Dados

[3]:
```
all_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'],
                      test.loc[:,'MSSubClass':'SaleCondition']))
```

[4]:
```
#log transform the target:
train["SalePrice"] = np.log1p(train["SalePrice"])

#log transform skewed numeric features:
numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index
```

```python
skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna()))
skewed_feats = skewed_feats[skewed_feats > 0.75]
skewed_feats = skewed_feats.index

all_data[skewed_feats] = np.log1p(all_data[skewed_feats])
```

[5]:
```python
all_data = pd.get_dummies(all_data)
```

[6]:
```python
#filling NA's with the mean of the column:
all_data = all_data.fillna(all_data.mean())
```

II - Modelos

[7]:
```python
# Função para calcular Root Mean Square Error com auxílio da
# Cross-Validation
def rmse_cv(model):
    rmse = -(cross_val_score(model, X_train, y,\
                        scoring="neg_root_mean_squared_error", cv = 5))
    return rmse
```

1. Regressão Logística

[8]:
```python
#Cria as matrizes para o sklearn:
X_train = all_data[:train.shape[0]]
X_test = all_data[train.shape[0]:]
y = train.SalePrice
```

[9]:
```python
reg = LinearRegression().fit(X_train, y)
reg.score(X_train,y)
```

[9]: 0.9473349439971036

[10]:
```python
rmse=rmse_cv(reg)
print(rmse)
```

```
[0.1242066  0.14781412 0.28084261 0.11374295 0.15959811]
```

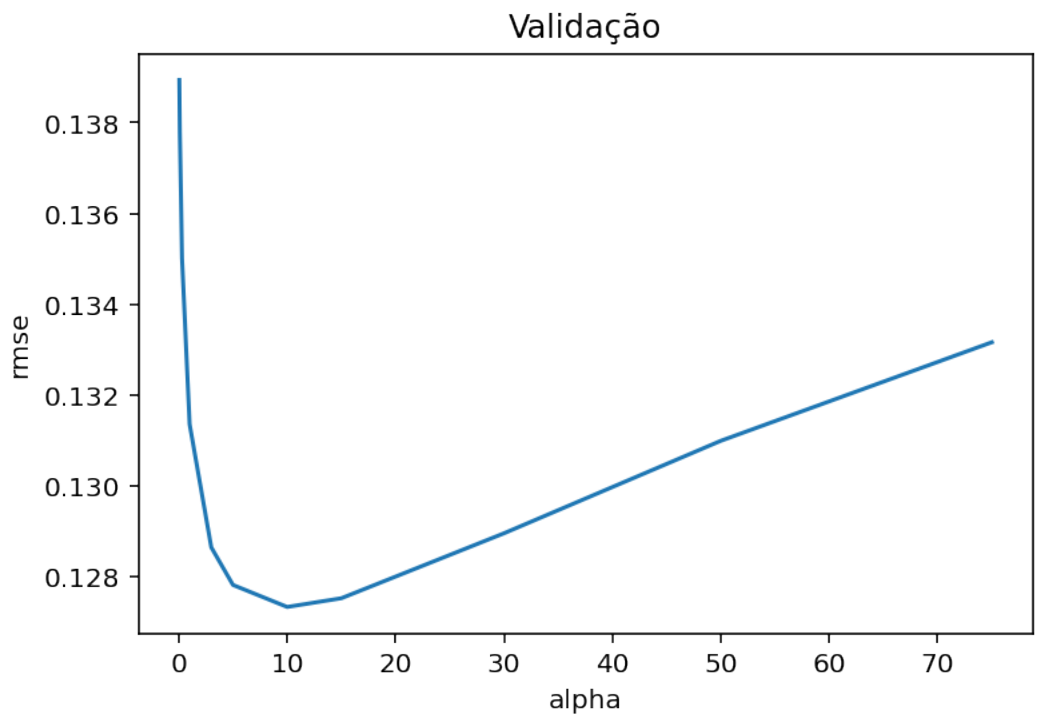
2 - Ridge

[11]:
```python
# Definição do intervalo de alphas à ser utilizado
alphas = [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75]

# Chama a função 'rmse_cv' e guarda a média dos valores por ela
# calculados em cv_ridge
cv_ridge = [rmse_cv(Ridge(alpha = alpha)).mean()
            for alpha in alphas]
```

[12]:
```
# Cria uma matriz de uma dimensão (cv_ridge) com o devido
# indexador (alphas), para montar o gráfico
cv_ridge = pd.Series(cv_ridge, index = alphas)
cv_ridge.plot(title = "Validação")
plt.xlabel("alpha")
plt.ylabel("rmse")
```

[12]: Text(0, 0.5, 'rmse')



3. Lasso

[13]:
```
# Definição do intervalo de alphas
alphas_lasso = [1, 0.8, 0.6, 0.4, 0.2, 0.1, 0.001, 0.00075, 0.0005]

# Matriz para guardar a média do rmse retornado da função rmse_cv
lasso_cv = len(alphas_lasso)*[0]

# Loop para cálculo do modelo 'Lasso' para cada um dos valores de
# alpha na lista 'alphas_lasso'. O modelo é passado para a função
# 'rmse_cv' e o valor médio é guardado em 'lasso_cv'
for i in range(len(alphas_lasso)):
    model_lasso=LassoCV(alphas=[alphas_lasso[i]]).fit(X_train, y)
    lasso_cv[i] = rmse_cv(model_lasso).mean()
print(rmse)
```

```
[0.1242066  0.14781412 0.28084261 0.11374295 0.15959811]
```

[14]:
```
# Cria uma matriz de uma dimensão (lasso_cv) com o devido
# indexador (alphas_lasso), para montar o gráfico
cv_lasso = pd.Series(lasso_cv, index = alphas_lasso)
cv_lasso.plot(title = "Validação")
plt.xlabel("alpha")
plt.ylabel("rmse")
```

[14]:
```
Text(0, 0.5, 'rmse')
```