

Week 3 Assignment

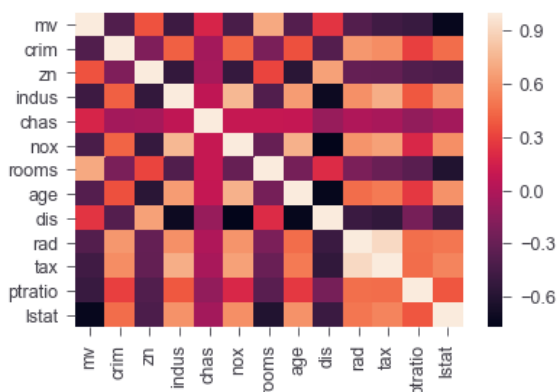
By: Valon Tika

Summary and Problem Definition

In real estate, there are many variables that can be used to determine the price of a house in a market. Typically, many real estate firms look at some of the higher noted metrics such as square footage, age of the property, and you guessed it, location. To try and manually assess each metric into a valuation model for a firm to confidently assess the price of a home can be time consuming and can cost the firm money and overhead to try and create complex models. Prior experience showed that many individuals that have been in the real estate market for quite some time have to manually gauge and set a range of prices for certain homes based on similar experiences in the past. To try and provide a more accurate assessment as to valuation of homes, machine learning can be used to automate and quickly provide more efficient analytics for housing prices by using many variables to dictate the prediction. This can reduce overhead for manual hours to create more drawn out modeling, and better assess and predict housing prices to optimize any given sale.

Research Design, Measurement and Statistical Method

For this analysis, the baseline dataset that was used was within the Boston housing market that



provided the median value of homes across multiple neighborhoods. 12 variables were given along with the median value of the homes to be used to give the machine learning models to see where how the model would perform. A correlation matrix was performed to assess how

closely correlated each variable was to the median value of the homes. Out of the 12 variables, number of rooms seemed mostly correlated to the median price of the homes.

Overview of Programming Work

In this analysis, the Boston housing dataset was cleansed and prepared to perform two machine learning models to determine the better approach for trying to predict the housing price. Both models were within the regression category of machine learning, which were ridge and lasso regressions. Each model provides specific traits and characteristics that make each model work great, but to level the playing field, the data was ran through a standard scalar feature pipeline to ensure that the models were normalized when we were to analyze the results

Review of Results – Recommendations

Out of the two regression models that were used, the ridge regression proved to be a better model slightly by around .7 points. Although this may not seem as much as one would anticipate, this does still hold that a ridge regression outperforms the lasso regression. The recommendation would be to deploy both machine learning, given that the same variables hold true for future predictions, to feed more data into these models for a better assessment, given that only 506 homes were given for this study. The next step would be to benchmark both analyses with how individuals price the same houses to see which prediction holds out to be more accurate to provide a better indication as to which method would be best used moving forward for actual decision making of housing prices.

<i>Regression</i>	<i>Score</i>	<i>MSE</i>	<i>RMSE</i>
<i>Ridge</i>	14.9573	3.86747	0.709716
<i>Lasso</i>	14.2435	3.77405	0.723569

Code Appendix:

```
# Boston Housing Study (Python)
# using data from the Boston Housing Study case
# as described in "Marketing Data Science: Modeling Techniques
# for Predictive Analytics with R and Python" (Miller 2015)

# Here we use data from the Boston Housing Study to evaluate
# regression modeling methods within a cross-validation design.

# program revised by Thomas W. Milller (2017/09/29)

# Scikit Learn documentation for this assignment:
# http://scikit-learn.org/stable/modules/model_evaluation.html
# http://scikit-learn.org/stable/modules/generated/
#   sklearn.model_selection.KFold.html
# http://scikit-learn.org/stable/modules/generated/
#   sklearn.linear_model.LinearRegression.html
# http://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html
# http://scikit-learn.org/stable/modules/generated/
#   sklearn.linear_model.Ridge.html
# http://scikit-learn.org/stable/modules/generated/
#   sklearn.linear_model.Lasso.html
# http://scikit-learn.org/stable/modules/generated/
#   sklearn.linear_model.ElasticNet.html
# http://scikit-learn.org/stable/modules/generated/
#   sklearn.metrics.r2_score.html

# Textbook reference materials:
# Geron, A. 2017. Hands-On Machine Learning with Scikit-Learn
# and TensorFlow. Sebastopol, Calif.: O'Reilly. Chapter 3 Training Models
# has sections covering linear regression, polynomial regression,
# and regularized linear models. Sample code from the book is
# available on GitHub at https://github.com/ageron/handson-ml

# prepare for Python version 3x features and functions
# comment out for Python 3.x execution
# from __future__ import division, print_function
# from future_builtins import ascii, filter, hex, map, oct, zip

# seed value for random number generators to obtain reproducible results
RANDOM_SEED = 1

# although we standardize X and y variables on input,
# we will fit the intercept term in the models
# Expect fitted values to be close to zero
SET_FIT_INTERCEPT = True

# import base packages into the namespace for this program
import numpy as np
import pandas as pd

# modeling routines from Scikit Learn packages
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline, FeatureUnion # Features
import sklearn.linear_model
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet # Models
from sklearn.pipeline import Pipeline, FeatureUnion # Features
```

```

from sklearn.model_selection import KFold, GridSearchCV, train_test_split # cross-
validation / feature tuning
from sklearn.metrics import mean_squared_error, r2_score # Performance Measurement
from sklearn.model_selection import cross_val_predict # Cross-validation
from sklearn.metrics import confusion_matrix # confusion matrix

from sklearn import metrics
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')

# read data for the Boston Housing Study
# creating data frame restdata
boston_input =
pd.read_csv(r'C:\Users\vtika\Desktop\MSDS\MSDS_422\Week3Assignment\jumpstart\boston.csv')

# check the pandas DataFrame object boston_input
print('\nboston DataFrame (first and last five rows):')
print(boston_input.head())
print(boston_input.tail())

print('\nGeneral description of the boston_input DataFrame:')
print(boston_input.info())

# drop neighborhood from the data being considered
boston = boston_input.drop('neighborhood', 1)
print('\nGeneral description of the boston DataFrame:')
print(boston.info())

print('\nDescriptive statistics of the boston DataFrame:')
print(boston.describe())

# set up preliminary data for data for fitting the models
# the first column is the median housing value response
# the remaining columns are the explanatory variables
prelim_model_data = np.array([boston.mv,\
    boston.crim,\
    boston.zn,\
    boston.indus,\
    boston.chas,\
    boston.nox,\
    boston.rooms,\
    boston.age,\
    boston.dis,\
    boston.rad,\
    boston.tax,\
    boston.ptratio,\
    boston.lstat]).T

# dimensions of the polynomial model X input and y response
# preliminary data before standardization
print('\nData dimensions:', prelim_model_data.shape)

# standard scores for the columns... along axis 0
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
print(scaler.fit(prelim_model_data))
# show standardization constants being employed
print(scaler.mean_)
print(scaler.scale_)

# the model data will be standardized form of preliminary model data

```

```

model_data = scaler.fit_transform(prelim_model_data)

# dimensions of the polynomial model X input and y response
# all in standardized units of measure
print('\nDimensions for model_data:', model_data.shape)

model = pd.DataFrame(model_data)

model.columns = ["mv", "crim", "zn", "indus", "chas", "nox", "rooms", "age", "dis", "rad",
"tax", "ptratio", "lstat"]

# EDA
model.hist( bins = 50, figsize = (30, 20)); plt.show()

corr_matrix = model.corr()
corr_matrix

# first we will start with building a ridge regression

label = 'median_value'
features = boston.columns.values[boston.columns != label]
samples = boston.shape[0]

num_pipeline = Pipeline([
    ('std_scaler', StandardScaler())
])

# Full data processing pipeline
full_pipeline = FeatureUnion( transformer_list = [
    ("num_pipeline", num_pipeline)
])

# transformed Model & Response data
X = boston.iloc[:, 0:11].values
y = boston.iloc[:, 11].values

# only scale our input values, leave the response out.
model_prepared = full_pipeline.fit_transform(X)

def perf(modelnm, model, X_test, y_test, y_pred):
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    score = model.score(X_test, y_test)

    return mse, rmse, score

def fit_pred( model, X_train, y_train, X_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    plt.scatter(y_test, y_pred); plt.show()
    return y_pred

cols = ['Regression', 'Score', 'MSE', 'RMSE']
comparison = pd.DataFrame(columns=cols)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=42)

# Ridge Regresiion

```

```

ridge_reg = Ridge(alpha = 1, solver = 'cholesky',
                  fit_intercept = SET_FIT_INTERCEPT,
                  normalize = False,
                  random_state = RANDOM_SEED)

ridge_pred = fit_pred(ridge_reg, X_train, y_train, X_test)

ridge_mse, ridge_rmse, ridge_score = perf('Ridge', ridge_reg, X_test, y_test,
                                           ridge_pred)

ridge = pd.DataFrame(['Ridge', ridge_mse, ridge_rmse, ridge_score]).T
ridge.columns = perf_cols

ridge

comparison = comparison.append(ridge)

# lasso regression

"""# Lasso"""

lasso_reg = Lasso(alpha = 0.1, max_iter=10000, tol=0.01,
                  fit_intercept = SET_FIT_INTERCEPT,
                  random_state = RANDOM_SEED)

lasso_pred = fit_pred(lasso_reg, X_train, y_train, X_test)

lasso_mse, lasso_rmse, lasso_score = perf('Lasso', lasso_reg, X_test, y_test,
                                           lasso_pred)

lasso = pd.DataFrame(['Lasso', lasso_mse, lasso_rmse, lasso_score]).T
lasso.columns = perf_cols

lasso

comparison = comparison.append(lasso)

# summary

comparison

plt.subplot(311)
comparison['Score'].plot(kind='bar', title = "Score", figsize=(15, 10), fontsize=12)
# Overall model score goes to the ridge type regression analysis

```