Week 4 Assignment

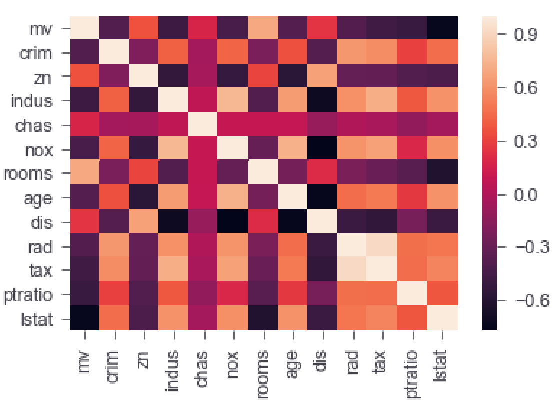
By: Valon Tika

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 a multitude of machine learning models to determine the better approach for trying to predict the housing price. The first set of models were within the regression category of machine learning, which were ridge and lasso regressions. Tree-based algorithms (decision tree and random forest) were used as well. For better performance of data that deals with fewer variables, regression modeling can be performed. When data is more natural by nature, then tree modeling can handle the complexities of the various attributes of the data. 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

After performance testing of the model classes, the results showed that tree-based modeling outperformed the regression modeling approach by overall performance accuracy as well as measurements of error. Out of the two tree-based models that performed well, the decisions of number of rooms, air pollution and crime rate dictated the tree-model’s decision line determination. Furthermore, out of the two tree-based models, it is recommended that random forest modeling would be the better approach. We did find that decision-tree modeling outperform random forest, but the instability of training decision-trees and that random forest modeling proved to have a smaller MSE or model error rate to the training sets.

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. Sebastopal, 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.cs
v')

# 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

import seaborn as sns
corr = corr_matrix
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values)

# 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 = 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 = cols

lasso

comparison = comparison.append(lasso)

# summary

comparison

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


# Tree Modeling addition to code
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

X = boston.iloc[:, 0:11]
```

```
y = boston.iloc[:, 12]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3,
random_state=0) # new set of training data, unscaled



def display_feature_importance(rf_reg):
  importances = rf_reg.feature_importances_
  std = np.std([rf_reg.feature_importances_ for tree in rf_reg.estimators_],
               axis=0)
  indices = np.argsort(importances)[::-1]

  # Print the feature ranking
  print("Feature ranking:")

  feats = []
  for f in range(X.shape[1]):
      print( features[indices[f]], ", F[%d] (%f)" % (f + 1, importances[indices[f]]))
      feats.append(features[indices[f]])

  # Plot the feature importances of the forest
  plt.figure()
  plt.title("Feature importances")
  plt.bar(range(X.shape[1]), importances[indices],
         color="r", yerr=std[indices], align="center")
  plt.xticks(range(X.shape[1]), feats)
  plt.xlim([-1, X.shape[1]])
  plt.xticks(rotation='vertical')
  plt.show()

  display_feature_importance(ridge)


tree = DecisionTreeRegressor(max_depth=10, max_features = 5, random_state=0)

dt_pred = fit_pred(tree, X_train, y_train, X_test)

dt_mse, dt_rmse, dt_score = perf('Decision Tree', tree, X_test, y_test, dt_pred)


dt_perf = pd.DataFrame(['Decision Tree', dt_mse, dt_rmse, dt_score]).T
dt_perf.columns = cols

dt_perf

comparison = comparison.append(dt_perf)

random = RandomForestRegressor(n_estimators=500, max_leaf_nodes=16, n_jobs=1,
                               random_state=0, bootstrap=True)

random_pred = fit_pred(random, X_train, y_train, X_test)

random_mse, random_rmse, random_score = perf('Random Tree', random, X_test, y_test,
random_pred)

random_perf = pd.DataFrame(['Random Forest', random_mse, random_rmse, random_score]).T
random_perf.columns = cols

random_perf

comparison = comparison.append(random_perf)
```

```
comparison.set_index("Regression", drop=True, inplace=True)

comparison.sort_values(by=['RMSE'])


comparison = comparison[comparison.Regression != 'Random Forest']


fig, axes = plt.subplots(nrows=3, ncols=1)
fig.suptitle('Model Performance Comparision', y = 1.025)
fig.subplots_adjust(top=4)

plt.subplot(311)
comparison['Score'].plot(kind='bar', title ="Prediction Score", figsize=(10, 8),
fontsize=12)
plt.xticks(rotation='horizontal')


plt.subplot(312)
comparison['MSE'].plot(kind='bar', title ="Mean Squared Error Rate", figsize=(10, 8),
fontsize=12)
plt.xticks(rotation='horizontal')

plt.subplot(313)
comparison['RMSE'].plot(kind='bar', title ="Root Mean Squared Error Rate",
figsize=(10, 8), fontsize=12)
plt.xticks(rotation='horizontal')

fig.tight_layout()
plt.show()
```