Feature_Selection

October 18, 2021

Feature Selection

Finding relationships between features

Feature selection using wrappers

source

What's the Purpose of Feature Selection

Many learning algorithms perform poorly on high-dimensional data. This is known as the curse of dimensionality

There are other reasons we may wish to reduce the number of features including:

1. Reducing computational cost
2. Reducing the cost associated with data collection
3. Improving Interpretability

Dataset: Boston Housing Data

Dependent Variable: MEDV: Median value of owner-occupied homes in 1000's of dollars

Explanatory Variables

CRIM: per capita crime rate by town

ZN: proportion of residential land zoned for lots over 25,000 sq.ft.

INDUS: proportion of non-retail business acres per town

CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)

NOX: nitric oxides concentration (parts per 10 million)

RM: average number of rooms per dwelling

AGE: proportion of owner-occupied units built prior to 1940

DIS: weighted distances to five Boston employment centres

RAD: index of accessibility to radial highways

TAX: full-value property-tax rate per 10,000 dollars

PTRATIO: pupil-teacher ratio by town

B: 1000(Bk - 0.63)^2 where Bk is the proportion of black residents by town

LSTAT: lower status of the population

```
[ ]: from sklearn.datasets import load_boston
     boston_data=load_boston()
     boston_data
```

```
[ ]: {'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, …, 1.5300e+01,
     3.9690e+02,
              4.9800e+00],
             [2.7310e-02, 0.0000e+00, 7.0700e+00, …, 1.7800e+01, 3.9690e+02,
              9.1400e+00],
             [2.7290e-02, 0.0000e+00, 7.0700e+00, …, 1.7800e+01, 3.9283e+02,
              4.0300e+00],
             …,
             [6.0760e-02, 0.0000e+00, 1.1930e+01, …, 2.1000e+01, 3.9690e+02,
              5.6400e+00],
             [1.0959e-01, 0.0000e+00, 1.1930e+01, …, 2.1000e+01, 3.9345e+02,
              6.4800e+00],
             [4.7410e-02, 0.0000e+00, 1.1930e+01, …, 2.1000e+01, 3.9690e+02,
              7.8800e+00]]),
      'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9,
     15. ,
             18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
             15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
             13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
             21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
             35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
             19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
             20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
             23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
             33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
             21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
             20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
             23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
             15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
             17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
             25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
             23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
             32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
             34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
             20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
             26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
             31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
             22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
             42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
             36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
             32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
             20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
             20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
```

```
        22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
        21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
        19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
        32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
        18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
        16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
        13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
         7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
        12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
        27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
         8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
         9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
        10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
        15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
        19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
        29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
        20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
        23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9]),
 'feature_names': array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
'DIS', 'RAD',
        'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'),
 'DESCR': ".. _boston_dataset:\n\nBoston house prices
dataset\n---------------------------\n\n**Data Set Characteristics:**  \n\n
:Number of Instances: 506 \n\n    :Number of Attributes: 13 numeric/categorical
predictive. Median Value (attribute 14) is usually the target.\n\n    :Attribute
Information (in order):\n        - CRIM     per capita crime rate by town\n
- ZN       proportion of residential land zoned for lots over 25,000 sq.ft.\n
- INDUS    proportion of non-retail business acres per town\n        - CHAS
Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)\n        -
NOX      nitric oxides concentration (parts per 10 million)\n        - RM
average number of rooms per dwelling\n        - AGE      proportion of owner-
occupied units built prior to 1940\n        - DIS      weighted distances to
five Boston employment centres\n        - RAD      index of accessibility to
radial highways\n        - TAX      full-value property-tax rate per $10,000\n
- PTRATIO  pupil-teacher ratio by town\n        - B        1000(Bk - 0.63)^2
where Bk is the proportion of blacks by town\n        - LSTAT    % lower status
of the population\n        - MEDV     Median value of owner-occupied homes in
$1000's\n\n    :Missing Attribute Values: None\n\n    :Creator: Harrison, D. and
Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing
dataset.\nhttps://archive.ics.uci.edu/ml/machine-learning-
databases/housing/\n\n\nThis dataset was taken from the StatLib library which is
maintained at Carnegie Mellon University.\n\nThe Boston house-price data of
Harrison, D. and Rubinfeld, D.L. 'Hedonic\nprices and the demand for clean air',
J. Environ. Economics & Management,\nvol.5, 81-102, 1978.   Used in Belsley, Kuh
& Welsch, 'Regression diagnostics\n…', Wiley, 1980.   N.B. Various
transformations are used in the table on\npages 244-261 of the latter.\n\nThe
Boston house-price data has been used in many machine learning papers that
```

```
address regression\nproblems.   \n     \n.. topic:: References\n\n   - Belsley,
Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources
of Collinearity', Wiley, 1980. 244-261.\n   - Quinlan,R. (1993). Combining
Instance-Based and Model-Based Learning. In Proceedings on the Tenth
International Conference of Machine Learning, 236-243, University of
Massachusetts, Amherst. Morgan Kaufmann.\n",
 'filename': 'C:\\Users\\aduzo\\Anaconda3\\lib\\site-
packages\\sklearn\\datasets\\data\\boston_house_prices.csv'}
```

[ ]:
```
import pandas as pd

boston = pd.DataFrame(boston_data.data, columns=boston_data.feature_names)
boston.head()
```

[ ]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
2     17.8  392.83   4.03
3     18.7  394.63   2.94
4     18.7  396.90   5.33
```

[ ]:
```
boston.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
```

```
 13  MEDV     506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

[ ]:
```
boston.nunique()
```

[ ]:
```
CRIM       504
ZN          26
INDUS       76
CHAS         2
NOX         81
RM         446
AGE        356
DIS        412
RAD          9
TAX         66
PTRATIO     46
B          357
LSTAT      455
MEDV       229
dtype: int64
```

CHAS and RAD seems to be categorical, as they have few labels. But CHAS has only 2 values, so we leave it. But we need do One hot encoding for RAD

[ ]:
```
boston['MEDV'] = boston_data.target
boston.head()
```

[ ]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT  MEDV
0     15.3  396.90   4.98  24.0
1     17.8  396.90   9.14  21.6
2     17.8  392.83   4.03  34.7
3     18.7  394.63   2.94  33.4
4     18.7  396.90   5.33  36.2
```

[ ]:
```
dummies = pd.get_dummies(boston.RAD)
dummies
```

[ ]:
```
     1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   24.0
0      1     0     0     0     0     0     0     0     0
1      0     1     0     0     0     0     0     0     0
```

```
2        0     1     0     0     0     0     0     0     0
3        0     0     1     0     0     0     0     0     0
4        0     0     1     0     0     0     0     0     0
..     …    …    …    …    …    …    …    …    …
501      1     0     0     0     0     0     0     0     0
502      1     0     0     0     0     0     0     0     0
503      1     0     0     0     0     0     0     0     0
504      1     0     0     0     0     0     0     0     0
505      1     0     0     0     0     0     0     0     0

[506 rows x 9 columns]
```

[ ]:
```
boston = boston.drop(columns='RAD').
 ↪merge(dummies,left_index=True,right_index=True)
X = boston.drop(columns='MEDV')
y = boston.MEDV
```

[ ]:
```
boston.head(10)
```

[ ]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM    AGE     DIS    TAX  PTRATIO  \
0  0.00632  18.0   2.31   0.0  0.538  6.575   65.2  4.0900  296.0     15.3
1  0.02731   0.0   7.07   0.0  0.469  6.421   78.9  4.9671  242.0     17.8
2  0.02729   0.0   7.07   0.0  0.469  7.185   61.1  4.9671  242.0     17.8
3  0.03237   0.0   2.18   0.0  0.458  6.998   45.8  6.0622  222.0     18.7
4  0.06905   0.0   2.18   0.0  0.458  7.147   54.2  6.0622  222.0     18.7
5  0.02985   0.0   2.18   0.0  0.458  6.430   58.7  6.0622  222.0     18.7
6  0.08829  12.5   7.87   0.0  0.524  6.012   66.6  5.5605  311.0     15.2
7  0.14455  12.5   7.87   0.0  0.524  6.172   96.1  5.9505  311.0     15.2
8  0.21124  12.5   7.87   0.0  0.524  5.631  100.0  6.0821  311.0     15.2
9  0.17004  12.5   7.87   0.0  0.524  6.004   85.9  6.5921  311.0     15.2

   …  MEDV  1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0  24.0
0  …  24.0    1    0    0    0    0    0    0    0     0
1  …  21.6    0    1    0    0    0    0    0    0     0
2  …  34.7    0    1    0    0    0    0    0    0     0
3  …  33.4    0    0    1    0    0    0    0    0     0
4  …  36.2    0    0    1    0    0    0    0    0     0
5  …  28.7    0    0    1    0    0    0    0    0     0
6  …  22.9    0    0    0    0    1    0    0    0     0
7  …  27.1    0    0    0    0    1    0    0    0     0
8  …  16.5    0    0    0    0    1    0    0    0     0
9  …  18.9    0    0    0    0    1    0    0    0     0

[10 rows x 22 columns]
```

0.1 From KNN

[ ]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_predict
from sklearn.linear_model import LinearRegression
from math import sqrt

cv = KFold(n_splits=10, random_state=0, shuffle=True)
classifier_pipeline = make_pipeline(StandardScaler(),
 ↪KNeighborsRegressor(n_neighbors=10))
y_pred = cross_val_predict(classifier_pipeline, X, y, cv=cv)
print("RMSE: " + str(round(sqrt(mean_squared_error(y,y_pred)),2)))
print("R_squared: " + str(round(r2_score(y,y_pred),2)))
```

```
RMSE: 5.39
R_squared: 0.66
```

Filter Features by Variance

[ ]:
```
boston.var()
```

[ ]:
```
CRIM           73.986578
ZN            543.936814
INDUS          47.064442
CHAS            0.064513
NOX             0.013428
RM              0.493671
AGE           792.358399
DIS             4.434015
TAX         28404.759488
PTRATIO         4.686989
B            8334.752263
LSTAT          50.994760
MEDV           84.586724
1.0             0.038039
2.0             0.045271
3.0             0.069597
4.0             0.170469
5.0             0.175968
6.0             0.048840
7.0             0.032532
8.0             0.045271
24.0            0.193198
dtype: float64
```

NOX and CHAS has lower variance, we need to drop them as they will have little to no effect.

```
[ ]: X = X.drop(columns=['NOX','CHAS'])
```

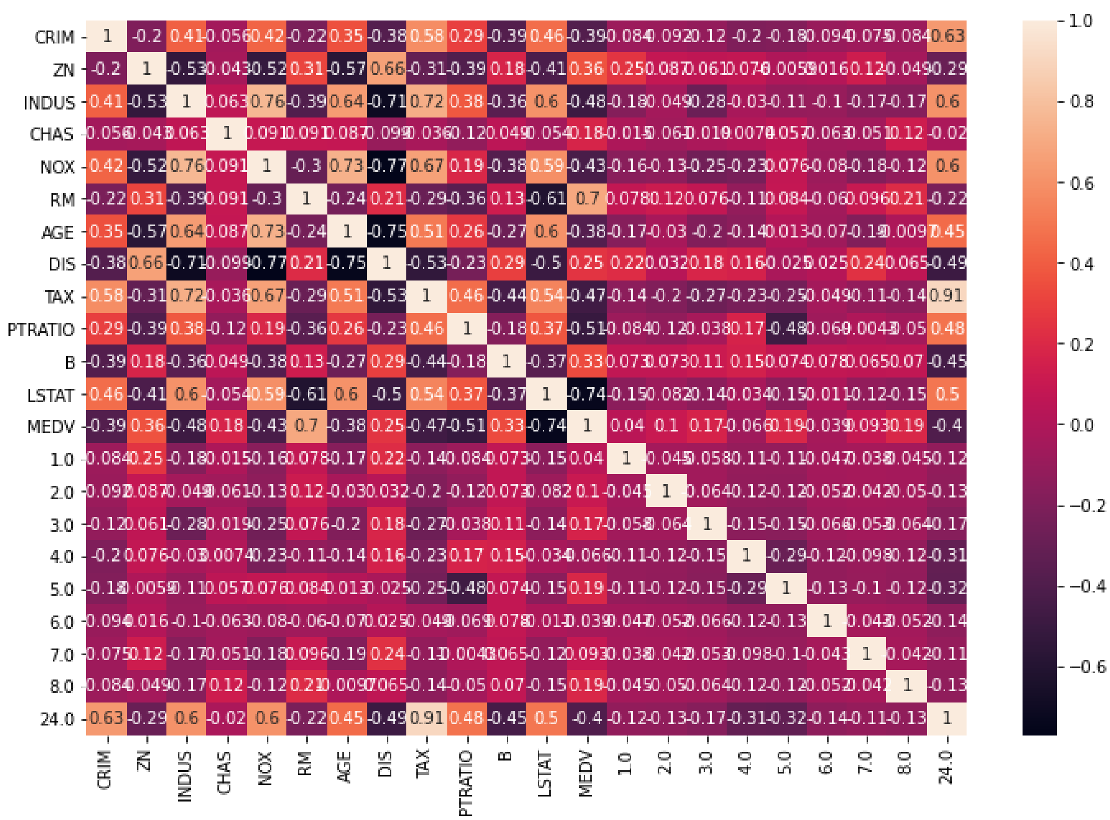
```
[ ]: y_pred = cross_val_predict(classifier_pipeline, X, y, cv=cv)
     print("RMSE: " + str(round(sqrt(mean_squared_error(y,y_pred)),2)))
     print("R_squared: " + str(round(r2_score(y,y_pred),2)))
```

```
RMSE: 5.07
R_squared: 0.7
```

Filter Features by Correlation

```
[ ]: import seaborn as sn
     import matplotlib.pyplot as plt
```

```
[ ]: fig_dims = (12,8)
     fig,ax = plt.subplots(figsize=fig_dims)
     sn.heatmap(boston.corr(),annot=True, ax=ax)
     plt.show()
```




```
[ ]: abs(boston.corr()["MEDV"])
```

```
[ ]: CRIM       0.388305
     ZN         0.360445
     INDUS      0.483725
     CHAS       0.175260
     NOX        0.427321
     RM         0.695360
     AGE        0.376955
     DIS        0.249929
     TAX        0.468536
     PTRATIO    0.507787
     B          0.333461
     LSTAT      0.737663
     MEDV       1.000000
     1.0        0.040453
     2.0        0.104444
     3.0        0.167352
     4.0        0.065711
     5.0        0.187356
     6.0        0.039411
     7.0        0.092802
     8.0        0.190053
     24.0       0.396297
     Name: MEDV, dtype: float64
```

```
[ ]: abs(boston.corr()["MEDV"][abs(boston.corr()["MEDV"])>0.5].drop('MEDV')).index.
      ↪tolist()
```

```
[ ]: ['RM', 'PTRATIO', 'LSTAT']
```

CHecking which correlation values gives best RMSE and R_squared

```
[ ]: vals = [0.1,0.2,0.3,0.4,0.5,0.6,0.7]
     for val in vals:
         features = abs(boston.corr()["MEDV"][abs(boston.corr()["MEDV"])>val].
      ↪drop('MEDV')).index.tolist()

         X = boston.drop(columns='MEDV')
         X=X[features]

         print(features)

         y_pred = cross_val_predict(classifier_pipeline, X, y, cv=cv)
         print("RMSE: " + str(round(sqrt(mean_squared_error(y,y_pred)),2)))
         print("R_squared: " + str(round(r2_score(y,y_pred),2)))
```

```
['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'TAX', 'PTRATIO',
'B', 'LSTAT', 2.0, 3.0, 5.0, 8.0, 24.0]
RMSE: 5.14
```

```
R_squared: 0.69
['CRIM', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE', 'DIS', 'TAX', 'PTRATIO', 'B',
'LSTAT', 24.0]
RMSE: 4.42
R_squared: 0.77
['CRIM', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE', 'TAX', 'PTRATIO', 'B', 'LSTAT',
24.0]
RMSE: 4.33
R_squared: 0.78
['INDUS', 'NOX', 'RM', 'TAX', 'PTRATIO', 'LSTAT']
RMSE: 4.28
R_squared: 0.78
['RM', 'PTRATIO', 'LSTAT']
RMSE: 4.3
R_squared: 0.78
['RM', 'LSTAT']
RMSE: 4.54
R_squared: 0.76
['LSTAT']
RMSE: 5.41
R_squared: 0.65
```

values at 0.4 and 0.5 are quite close: * ['INDUS', 'NOX', 'RM', 'TAX', 'PTRATIO', 'LSTAT'] * RMSE: 4.28 * R_squared: 0.78 * ['RM', 'PTRATIO', 'LSTAT'] * RMSE: 4.3 * R_squared: 0.78

We can choose ['RM', 'PTRATIO', 'LSTAT'] since it has number of smaller labels.

Feature Selection Using a Wrapper

this is sequencial feature selection. we use foword feature selection, adding features one at a time.

[ ]:
```
boston.columns
```

[ ]:
```
Index([   'CRIM',      'ZN',   'INDUS',    'CHAS',     'NOX',      'RM',
           'AGE',     'DIS',     'TAX', 'PTRATIO',       'B',   'LSTAT',
          'MEDV',       1.0,       2.0,       3.0,       4.0,       5.0,
             6.0,       7.0,       8.0,      24.0],
      dtype='object')
```

[ ]:
```
boston = pd.DataFrame(boston_data.data, columns=boston_data.feature_names)
boston['MEDV'] = boston_data.target
boston['RAD'] = boston['RAD'].astype('category')
```

[ ]:
```
boston.head()
```

[ ]:
```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
```

```
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT  MEDV
0     15.3  396.90   4.98  24.0
1     17.8  396.90   9.14  21.6
2     17.8  392.83   4.03  34.7
3     18.7  394.63   2.94  33.4
4     18.7  396.90   5.33  36.2
```

[ ]:
```
dummies = pd.get_dummies(boston.RAD)
boston = boston.drop(columns='RAD').
 ↪merge(dummies,left_index=True,right_index=True)
X = boston.drop(columns='MEDV')
y = boston.MEDV
```

[ ]:
```
!pip install mlxtend
```

```
Collecting mlxtend
  Downloading mlxtend-0.19.0-py2.py3-none-any.whl (1.3 MB)
Requirement already satisfied: matplotlib>=3.0.0 in
c:\users\aduzo\anaconda3\lib\site-packages (from mlxtend) (3.4.2)
Requirement already satisfied: pandas>=0.24.2 in
c:\users\aduzo\anaconda3\lib\site-packages (from mlxtend) (1.2.4)
Requirement already satisfied: setuptools in c:\users\aduzo\anaconda3\lib\site-
packages (from mlxtend) (45.2.0.post20200210)
Requirement already satisfied: scikit-learn>=0.20.3 in
c:\users\aduzo\anaconda3\lib\site-packages (from mlxtend) (0.24.1)
Requirement already satisfied: joblib>=0.13.2 in
c:\users\aduzo\anaconda3\lib\site-packages (from mlxtend) (0.14.1)
Requirement already satisfied: scipy>=1.2.1 in
c:\users\aduzo\anaconda3\lib\site-packages (from mlxtend) (1.4.1)
Requirement already satisfied: numpy>=1.16.2 in
c:\users\aduzo\anaconda3\lib\site-packages (from mlxtend) (1.18.1)
Requirement already satisfied: pyparsing>=2.2.1 in
c:\users\aduzo\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend)
(2.4.6)
Requirement already satisfied: cycler>=0.10 in
c:\users\aduzo\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend)
(0.10.0)
Requirement already satisfied: python-dateutil>=2.7 in
c:\users\aduzo\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend)
(2.8.1)
Requirement already satisfied: pillow>=6.2.0 in
c:\users\aduzo\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend)
(7.0.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
c:\users\aduzo\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend)
(1.1.0)
```

```
Requirement already satisfied: pytz>=2017.3 in
c:\users\aduzo\anaconda3\lib\site-packages (from pandas>=0.24.2->mlxtend)
(2019.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\aduzo\anaconda3\lib\site-packages (from scikit-learn>=0.20.3->mlxtend)
(2.1.0)
Requirement already satisfied: six in c:\users\aduzo\anaconda3\lib\site-packages
(from cycler>=0.10->matplotlib>=3.0.0->mlxtend) (1.14.0)
Installing collected packages: mlxtend
Successfully installed mlxtend-0.19.0
```

[ ]:
```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
```

forward=False means backwards feature selection

[ ]:
```
sfs1 = SFS(classifier_pipeline,k_features=1,forward=False,␣
 ↪scoring='neg_mean_squared_error',cv=cv)
X= boston.drop(columns='MEDV')
sfs1.fit(X,y)
sfs1.subsets_
```

[ ]:
```
{21: {'feature_idx': (0,
   1,
   2,
   3,
   4,
   5,
   6,
   7,
   8,
   9,
   10,
   11,
   12,
   13,
   14,
   15,
   16,
   17,
   18,
   19,
   20),
  'cv_scores': array([-48.59168039, -41.71460196, -26.77341373, -18.77266275,
         -17.19804314, -21.57217647, -20.388668  , -42.540924  ,
         -36.872014  , -15.978064  ]),
  'avg_score': -29.040224843137253,
  'feature_names': ('CRIM',
   'ZN',
```

```
  'INDUS',
  'CHAS',
  'NOX',
  'RM',
  'AGE',
  'DIS',
  'TAX',
  'PTRATIO',
  'B',
  'LSTAT',
  1.0,
  2.0,
  3.0,
  4.0,
  5.0,
  6.0,
  7.0,
  8.0,
  24.0)},
20: {'feature_idx': (0,
  1,
  2,
  4,
  5,
  6,
  7,
  8,
  9,
  10,
  11,
  12,
  13,
  14,
  15,
  16,
  17,
  18,
  19,
  20),
 'cv_scores': array([-50.55976078, -37.94587451, -25.38272745, -17.01997843,
        -12.40205882, -18.35837255, -15.604718  , -36.948534  ,
        -32.602062  , -14.400774  ]),
 'avg_score': -26.12248605490196,
 'feature_names': ('CRIM',
  'ZN',
  'INDUS',
  'NOX',
```

```
  'RM',
  'AGE',
  'DIS',
  'TAX',
  'PTRATIO',
  'B',
  'LSTAT',
  1.0,
  2.0,
  3.0,
  4.0,
  5.0,
  6.0,
  7.0,
  8.0,
  24.0)},
19: {'feature_idx': (0,
  1,
  2,
  4,
  5,
  6,
  7,
  8,
  9,
  10,
  11,
  13,
  14,
  15,
  16,
  17,
  18,
  19,
  20),
 'cv_scores': array([-49.9469902 , -36.10369216, -17.79690784, -17.01997843,
        -11.88650392, -16.03129412, -15.598966  , -36.867982  ,
        -31.510666  , -12.759792  ]),
 'avg_score': -24.552277266666664,
 'feature_names': ('CRIM',
  'ZN',
  'INDUS',
  'NOX',
  'RM',
  'AGE',
  'DIS',
  'TAX',
```

```
  'PTRATIO',
  'B',
  'LSTAT',
  2.0,
  3.0,
  4.0,
  5.0,
  6.0,
  7.0,
  8.0,
  24.0)},
18: {'feature_idx': (0,
  1,
  4,
  5,
  6,
  7,
  8,
  9,
  10,
  11,
  13,
  14,
  15,
  16,
  17,
  18,
  19,
  20),
 'cv_scores': array([-47.78643529, -34.36723529, -17.2377098 , -18.16885882,
        -11.72155098, -13.57922353, -14.488454  , -37.333346  ,
        -25.180216  , -12.412428  ]),
 'avg_score': -23.227545772549018,
 'feature_names': ('CRIM',
  'ZN',
  'NOX',
  'RM',
  'AGE',
  'DIS',
  'TAX',
  'PTRATIO',
  'B',
  'LSTAT',
  2.0,
  3.0,
  4.0,
  5.0,
```

```
  6.0,
  7.0,
  8.0,
  24.0)},
17: {'feature_idx': (0,
  4,
  5,
  6,
  7,
  8,
  9,
  10,
  11,
  13,
  14,
  15,
  16,
  17,
  18,
  19,
  20),
 'cv_scores': array([-47.03072353, -32.70351373, -14.65140196, -16.21732941,
        -11.28102353, -11.20903725, -11.290576  , -33.585092  ,
        -26.068072  , -11.705766  ]),
 'avg_score': -21.574253541176468,
 'feature_names': ('CRIM',
  'NOX',
  'RM',
  'AGE',
  'DIS',
  'TAX',
  'PTRATIO',
  'B',
  'LSTAT',
  2.0,
  3.0,
  4.0,
  5.0,
  6.0,
  7.0,
  8.0,
  24.0)},
16: {'feature_idx': (0,
  5,
  6,
  7,
  8,
```

```
  9,
  10,
  11,
  13,
  14,
  15,
  16,
  17,
  18,
  19,
  20),
 'cv_scores': array([-42.81522353, -31.47450784, -15.20731176, -13.80531373,
        -11.44273922, -11.37029412, -11.618998  , -31.623378  ,
        -24.288234  , -12.40173   ]),
 'avg_score': -20.60477301960784,
 'feature_names': ('CRIM',
  'RM',
  'AGE',
  'DIS',
  'TAX',
  'PTRATIO',
  'B',
  'LSTAT',
  2.0,
  3.0,
  4.0,
  5.0,
  6.0,
  7.0,
  8.0,
  24.0)},
15: {'feature_idx': (0, 5, 6, 7, 8, 9, 10, 11, 14, 15, 16, 17, 18, 19, 20),
 'cv_scores': array([-43.91592353, -31.27087647, -15.49876667, -13.4269549 ,
        -10.21427451, -11.38014706, -11.361924  , -28.393506  ,
        -23.21451   , -11.676652  ]),
 'avg_score': -20.03535351372549,
 'feature_names': ('CRIM',
  'RM',
  'AGE',
  'DIS',
  'TAX',
  'PTRATIO',
  'B',
  'LSTAT',
  3.0,
  4.0,
  5.0,
```

```
  6.0,
  7.0,
  8.0,
  24.0)},
14: {'feature_idx': (0, 5, 6, 7, 8, 9, 10, 11, 14, 16, 17, 18, 19, 20),
 'cv_scores': array([-42.42491765, -30.09559412, -13.93446471, -13.30637647,
         -9.98437059,  -9.57690588, -11.161788  , -28.05578   ,
        -19.261716  , -10.573268  ]),
 'avg_score': -18.837518141176467,
 'feature_names': ('CRIM',
  'RM',
  'AGE',
  'DIS',
  'TAX',
  'PTRATIO',
  'B',
  'LSTAT',
  3.0,
  5.0,
  6.0,
  7.0,
  8.0,
  24.0)},
13: {'feature_idx': (0, 5, 6, 7, 8, 9, 10, 11, 14, 17, 18, 19, 20),
 'cv_scores': array([-40.8025549 , -29.22255098, -13.13314902, -10.07646471,
        -10.1168098 ,  -8.99435098, -10.813808  , -28.734018  ,
        -18.32103   ,  -8.509048  ]),
 'avg_score': -17.872378439215687,
 'feature_names': ('CRIM',
  'RM',
  'AGE',
  'DIS',
  'TAX',
  'PTRATIO',
  'B',
  'LSTAT',
  3.0,
  6.0,
  7.0,
  8.0,
  24.0)},
12: {'feature_idx': (0, 5, 6, 7, 8, 9, 10, 11, 14, 17, 18, 20),
 'cv_scores': array([-41.0094451 , -21.51673333, -12.67205686, -10.2137902 ,
        -11.47662941,  -8.62372353, -10.1718    , -27.261868  ,
        -17.070686  ,  -8.602126  ]),
 'avg_score': -16.86188584313725,
 'feature_names': ('CRIM',
```

```
  'RM',
  'AGE',
  'DIS',
  'TAX',
  'PTRATIO',
  'B',
  'LSTAT',
  3.0,
  6.0,
  7.0,
  24.0)},
11: {'feature_idx': (0, 5, 6, 7, 8, 9, 10, 11, 14, 17, 20),
 'cv_scores': array([-40.90624118, -19.17373333, -10.94333529,  -9.55001176,
        -10.42460784,  -9.05043922, -10.912978  , -25.916776  ,
        -16.161084  ,  -8.243874  ]),
 'avg_score': -16.1283080627451,
 'feature_names': ('CRIM',
  'RM',
  'AGE',
  'DIS',
  'TAX',
  'PTRATIO',
  'B',
  'LSTAT',
  3.0,
  6.0,
  24.0)},
10: {'feature_idx': (0, 5, 6, 7, 8, 9, 10, 11, 17, 20),
 'cv_scores': array([-40.91680784, -15.76651176, -11.42365294,  -8.93385882,
        -11.03835294,  -8.41469804, -11.425298  , -22.612064  ,
        -15.754702  ,  -7.6163    ]),
 'avg_score': -15.390224635294118,
 'feature_names': ('CRIM',
  'RM',
  'AGE',
  'DIS',
  'TAX',
  'PTRATIO',
  'B',
  'LSTAT',
  6.0,
  24.0)},
9: {'feature_idx': (0, 5, 6, 7, 8, 10, 11, 17, 20),
 'cv_scores': array([-38.47787843, -12.4776549 , -11.88287647, -10.2724549 ,
        -14.09338431,  -8.17037451, -11.602286  , -23.061324  ,
        -16.362544  ,  -6.752922  ]),
 'avg_score': -15.315369952941177,
```

```
 'feature_names': ('CRIM',
  'RM',
  'AGE',
  'DIS',
  'TAX',
  'B',
  'LSTAT',
  6.0,
  24.0)},
8: {'feature_idx': (0, 5, 6, 7, 8, 10, 11, 20),
 'cv_scores': array([-38.59446078, -12.4009549 , -11.73373922, -10.4310098 ,
        -14.25609608,  -7.97750392, -11.672444  , -22.930412  ,
        -16.30724   ,  -7.001616  ]),
 'avg_score': -15.330547670588237,
 'feature_names': ('CRIM', 'RM', 'AGE', 'DIS', 'TAX', 'B', 'LSTAT', 24.0)},
7: {'feature_idx': (5, 6, 7, 8, 10, 11, 20),
 'cv_scores': array([-37.01441961, -12.19522353, -12.23485098, -11.11031765,
        -12.00725686,  -7.38938824, -12.413448  , -23.332076  ,
        -19.034374  ,  -7.032862  ]),
 'avg_score': -15.376421686274508,
 'feature_names': ('RM', 'AGE', 'DIS', 'TAX', 'B', 'LSTAT', 24.0)},
6: {'feature_idx': (5, 6, 7, 8, 10, 11),
 'cv_scores': array([-36.25552745, -12.13867451, -12.76186471, -11.0447549 ,
        -20.93062157,  -7.83287647, -12.086392  , -24.469916  ,
        -15.523198  ,  -5.573872  ]),
 'avg_score': -15.861769760784313,
 'feature_names': ('RM', 'AGE', 'DIS', 'TAX', 'B', 'LSTAT')},
5: {'feature_idx': (5, 6, 8, 10, 11),
 'cv_scores': array([-40.36428431, -11.64602745, -11.75288235, -11.05012745,
        -22.02508824,  -9.15882549, -10.276012  , -24.806586  ,
        -17.805982  ,  -6.83536   ]),
 'avg_score': -16.572117529411766,
 'feature_names': ('RM', 'AGE', 'TAX', 'B', 'LSTAT')},
4: {'feature_idx': (5, 8, 10, 11),
 'cv_scores': array([-39.22891373, -14.86316667, -12.3641549 , -10.30318235,
        -21.51774118, -10.57488235, -11.630502  , -27.733194  ,
        -19.617986  ,  -6.808664  ]),
 'avg_score': -17.464238717647056,
 'feature_names': ('RM', 'TAX', 'B', 'LSTAT')},
3: {'feature_idx': (5, 8, 11),
 'cv_scores': array([-40.06265294, -17.68942549, -12.4353902 , -10.99857843,
        -22.06529804, -11.56955098, -11.40437   , -30.8838    ,
        -19.579944  ,  -7.522576  ]),
 'avg_score': -18.42115860784314,
 'feature_names': ('RM', 'TAX', 'LSTAT')},
2: {'feature_idx': (5, 11),
 'cv_scores': array([-40.28273137, -17.3283451 , -16.87722549,  -8.98089608,
```

```
         -23.57910196, -12.06788235, -17.998684  , -41.885448  ,
         -18.271946  ,  -9.099978  ]),
  'avg_score': -20.637223835294115,
  'feature_names': ('RM', 'LSTAT')},
 1: {'feature_idx': (11,),
  'cv_scores': array([-34.76516078, -33.12735686, -20.46438627, -33.75127647,
         -21.83596275, -20.94906667, -27.037996  , -30.307572  ,
         -49.354918  , -21.412932  ]),
  'avg_score': -29.300662780392155,
  'feature_names': ('LSTAT',)}}
```

choose the ave_score with highest value the this

[ ]:
```
X = boston.drop(columns='MEDV')[['CRIM','RM','PTRATIO','LSTAT']]
y = boston['MEDV']
y_pred = cross_val_predict(classifier_pipeline, X, y, cv=cv)
print("RMSE: " + str(round(sqrt(mean_squared_error(y,y_pred)),3)))
print("R_squared: " + str(round(r2_score(y,y_pred),3)))
```

```
RMSE: 4.102
R_squared: 0.801
```

`['CRIM','RM','AGE','DIS','TAX','B','LSTAT',6.0,24.0]` is the has the highest with 'avg_score': -15.315369952941177

[ ]:
```
boston[['CRIM','RM','AGE','DIS','TAX','B','LSTAT',6.0,24.0]]
```

[ ]:
```
        CRIM     RM   AGE     DIS    TAX       B  LSTAT  6.0  24.0
0    0.00632  6.575  65.2  4.0900  296.0  396.90   4.98    0     0
1    0.02731  6.421  78.9  4.9671  242.0  396.90   9.14    0     0
2    0.02729  7.185  61.1  4.9671  242.0  392.83   4.03    0     0
3    0.03237  6.998  45.8  6.0622  222.0  394.63   2.94    0     0
4    0.06905  7.147  54.2  6.0622  222.0  396.90   5.33    0     0
..       …      …    …      …      …       …    …    …     …
501  0.06263  6.593  69.1  2.4786  273.0  391.99   9.67    0     0
502  0.04527  6.120  76.7  2.2875  273.0  396.90   9.08    0     0
503  0.06076  6.976  91.0  2.1675  273.0  396.90   5.64    0     0
504  0.10959  6.794  89.3  2.3889  273.0  393.45   6.48    0     0
505  0.04741  6.030  80.8  2.5050  273.0  396.90   7.88    0     0

[506 rows x 9 columns]
```

[ ]:
```
X = boston.drop(columns='MEDV')[['CRIM','RM','AGE','DIS','TAX','B','LSTAT',6.
 ↪0,24.0]]
y = boston['MEDV']
y_pred = cross_val_predict(classifier_pipeline, X, y, cv=cv)
print("RMSE: " + str(round(sqrt(mean_squared_error(y,y_pred)),3)))
print("R_squared: " + str(round(r2_score(y,y_pred),3)))
```

```
RMSE: 3.914
R_squared: 0.818
```

[ ]:
```
boston[['CRIM','RM','PTRATIO','LSTAT','MEDV']].corr()
```

[ ]:

	CRIM	RM	PTRATIO	LSTAT	MEDV
CRIM	1.000000	-0.219247	0.289946	0.455621	-0.388305
RM	-0.219247	1.000000	-0.355501	-0.613808	0.695360
PTRATIO	0.289946	-0.355501	1.000000	0.374044	-0.507787
LSTAT	0.455621	-0.613808	0.374044	1.000000	-0.737663
MEDV	-0.388305	0.695360	-0.507787	-0.737663	1.000000

LSTAT and RM has high correlation among them apart from MEDV, so we can add an interaction between them.

[ ]:
```
boston['RM*LSTAT']=boston['RM']*boston['LSTAT']
```

[ ]:
```
X = boston.drop(columns='MEDV')[['CRIM','RM','PTRATIO','LSTAT']]
y = boston['MEDV']
y_pred = cross_val_predict(classifier_pipeline, X, y, cv=cv)
print("RMSE: " + str(round(sqrt(mean_squared_error(y,y_pred)),3)))
print("R_squared: " + str(round(r2_score(y,y_pred),3)))
```

```
RMSE: 4.102
R_squared: 0.801
```

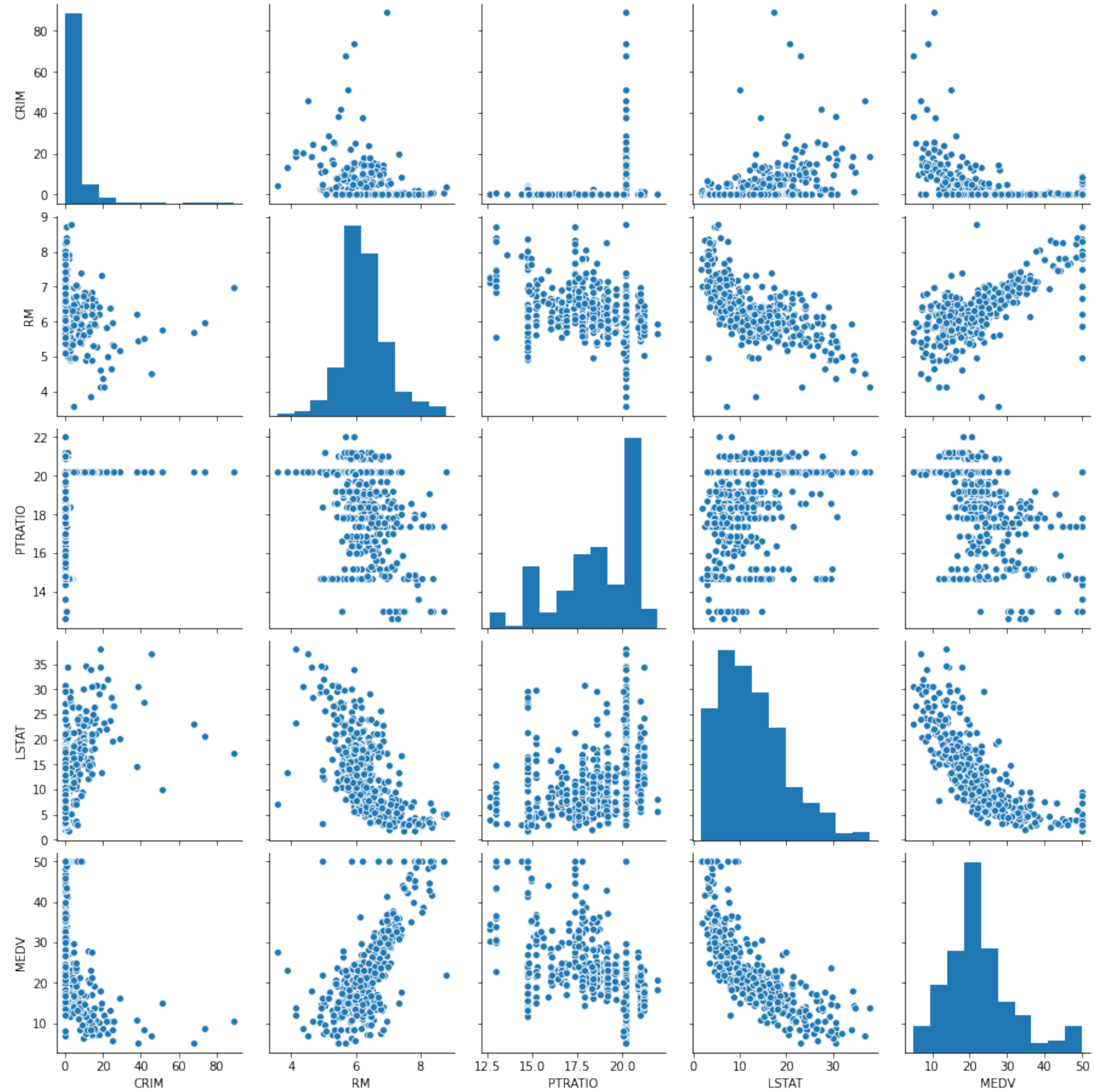
[ ]:
```
X = boston.drop(columns='MEDV')[['CRIM','RM','PTRATIO','LSTAT','RM*LSTAT']]
y = boston['MEDV']
y_pred = cross_val_predict(classifier_pipeline, X, y, cv=cv)
print("RMSE: " + str(round(sqrt(mean_squared_error(y,y_pred)),3)))
print("R_squared: " + str(round(r2_score(y,y_pred),3)))
```

```
RMSE: 4.078
R_squared: 0.803
```

Only a little improvement

[ ]:
```
sn.pairplot(boston[['CRIM','RM','PTRATIO','LSTAT','MEDV']])
```

[ ]:
```
<seaborn.axisgrid.PairGrid at 0x17a31cebf88>
```



[ ]:
```
boston[['CRIM','RM','AGE','DIS','TAX','B','LSTAT',6.0,24.0]].corr()
```

[ ]:

	CRIM	RM	AGE	DIS	TAX	B	LSTAT	\
CRIM	1.000000	-0.219247	0.352734	-0.379670	0.582764	-0.385064	0.455621	
RM	-0.219247	1.000000	-0.240265	0.205246	-0.292048	0.128069	-0.613808	
AGE	0.352734	-0.240265	1.000000	-0.747881	0.506456	-0.273534	0.602339	
DIS	-0.379670	0.205246	-0.747881	1.000000	-0.534432	0.291512	-0.496996	
TAX	0.582764	-0.292048	0.506456	-0.534432	1.000000	-0.441808	0.543993	
B	-0.385064	0.128069	-0.273534	0.291512	-0.441808	1.000000	-0.366087	
LSTAT	0.455621	-0.613808	0.602339	-0.496996	0.543993	-0.366087	1.000000	
6.0	-0.093806	-0.059651	-0.069790	0.025432	-0.048868	0.078322	-0.011330	
24.0	0.632302	-0.222159	0.448516	-0.489642	0.909506	-0.446748	0.495285	

```
              6.0      24.0
CRIM   -0.093806  0.632302
RM     -0.059651 -0.222159
AGE    -0.069790  0.448516
DIS     0.025432 -0.489642
TAX    -0.048868  0.909506
B       0.078322 -0.446748
LSTAT  -0.011330  0.495285
6.0     1.000000 -0.138267
24.0   -0.138267  1.000000
```

Drop outliers

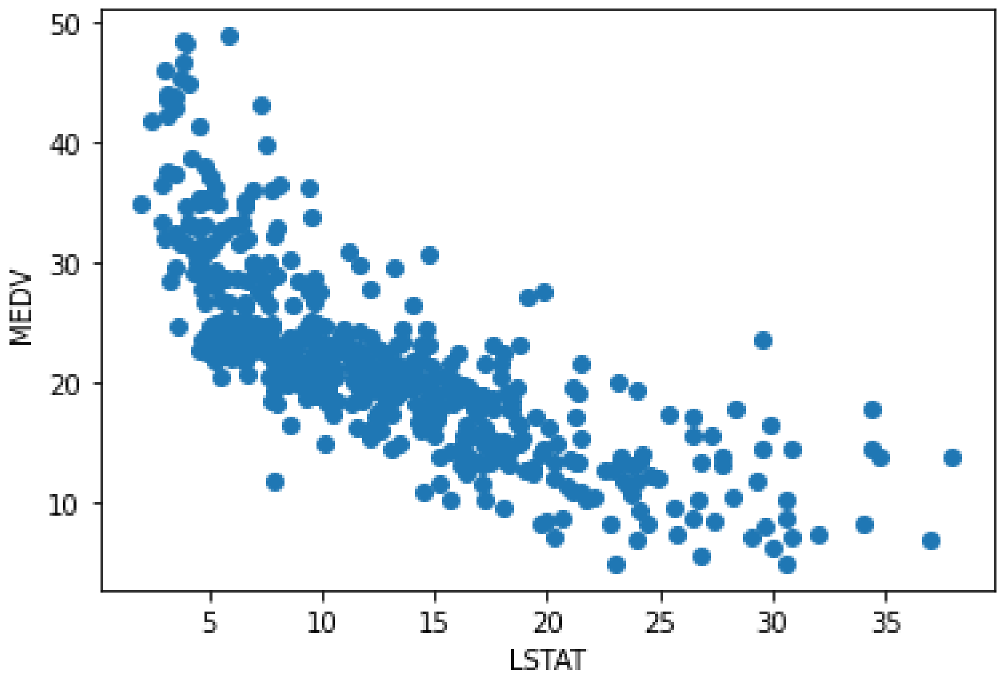
```
[ ]: boston = boston.drop(boston[boston['MEDV']==boston['MEDV'].max()].index.
      ↪tolist())
```

```
[ ]: X = boston.drop(columns='MEDV')[['CRIM','RM','PTRATIO','LSTAT','RM*LSTAT']]
     y = boston['MEDV']
     y_pred = cross_val_predict(classifier_pipeline, X, y, cv=cv)
     print("RMSE: " + str(round(sqrt(mean_squared_error(y,y_pred)),3)))
     print("R_squared: " + str(round(r2_score(y,y_pred),3)))
```

```
RMSE: 3.295
R_squared: 0.824
```

This really improved the model with features `['CRIM','RM','PTRATIO','LSTAT','RM*LSTAT']`

```
[ ]: plt.scatter(boston['LSTAT'],boston['MEDV'])
     plt.xlabel('LSTAT')
     plt.ylabel('MEDV')
     plt.show()
```



There is a non-linear relationship between dependent variable MEDV and independent variable LSTAT

```
[ ]: boston['LSTAT_2']=boston['LSTAT']**2
```

```
[ ]: X = boston.drop(columns='MEDV')[['CRIM','RM','PTRATIO','LSTAT','LSTAT_2']]
     y_pred = cross_val_predict(classifier_pipeline, X, y, cv=cv)
     print("RMSE: " + str(round(sqrt(mean_squared_error(y,y_pred)),3)))
     print("R_squared: " + str(round(r2_score(y,y_pred),3)))
```

```
RMSE: 3.301
R_squared: 0.824
```

That is not good, because our RMSE slighly increased.