Prediction of Boston House Price

Presenters: Jinyi Shang
Yu Cao
Chaohui Li

Contents

PART 01

Introduction

Introduction

The house value is related to many factors. So our project will focus on solving the problem of predicting house price for house buyers and sellers.

The data is collected from Boston. There are total 506 entries and 13 features.

PART 02

Dataset Description

Dataset Description

#	Column	Description	Non-Null Count	Dtype
0	CRIM	Crime rate per capita in towns	486 non-null	float64
1	ZN	Proportion of residential land	486 non-null	float64
2	INDUS	Proportion of non-commercial land in urban areas	486 non-null	float64
3	CHAS	Charles River **Dummy Variable**	486 non-null	float64
4	NOX	Environmental protection index	506 non-null	float64
5	RM	Number of rooms per house	506 non-null	float64
6	AGE	Proportion of self-occupied units built before 1940	486 non-null	float64
7	DIS	Weighted distance from Boston's five employment centers	506 non-null	float64
8	RAD	Convenience Index to Highway	506 non-null	int64
9	TAX	Real estate tax rate per US$10,100	506 non-null	int64
10	PTRATIO	Teacher-student ratio in towns	506 non-null	float64
11	B	Proportion of blacks in towns	506 non-null	float64
12	LSTAT	Proportion of landlords belonging to the lower income class	486 non-null	float64
13	MEDV	Median of house price of self-occupied house	506 non-null	float64

Dataset Description

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
count	486	486	486	486	506	506	486
mean	3.612	11.212	11.084	0.070	0.555	6.285	68.519
std	8.720	23.389	6.836	0.255	0.116	0.703	28.000
min	0.006	0.000	0.460	0.000	0.385	3.561	2.900
median	0.254	0.000	9.690	0.000	0.538	6.209	76.800
max	88.976	100.000	27.740	1.000	0.871	8.780	100.000

	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
count	506	506	506	506	506	486	506
mean	3.795	9.549	408.237	18.456	356.674	12.715	22.533
std	2.106	8.707	168.537	2.165	91.295	7.1561	9.197
min	1.130	1.000	187.000	12.600	0.320	1.730	5.000
median	3.207	5.000	330.000	19.050	391.440	11.430	21.200
max	12.127	24.000	711.000	22.000	396.900	37.970	50.000

PART 03

Data Preprocessing

Data Preprocessing

Missing Values

The numbers of missing values for every feature:

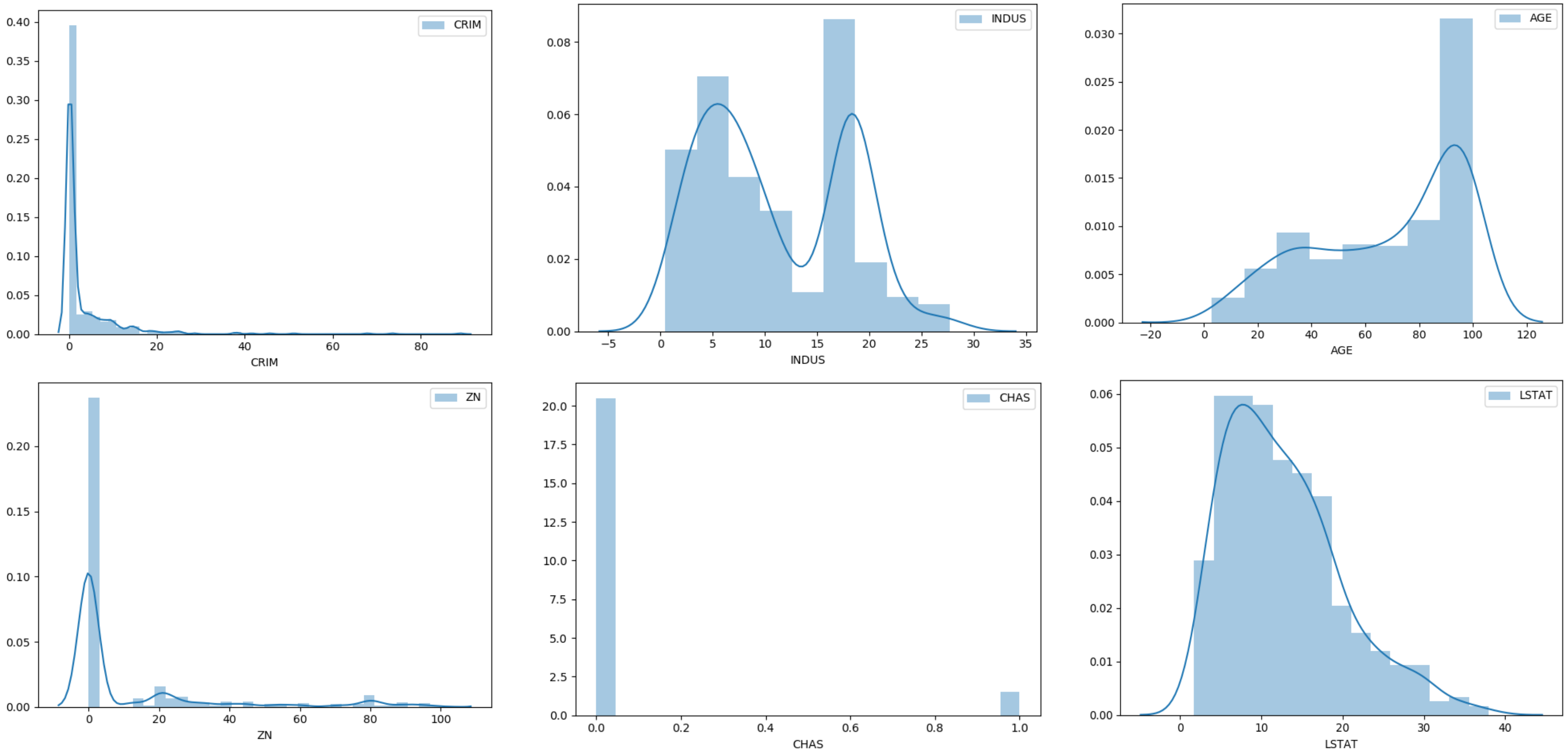
CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	TRUE
20	20	20	20	0	0	20

DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
0	0	0	0	0	20	0

Data Preprocessing

Missing Values

The distribution for features that exist missing values:



Data Preprocessing

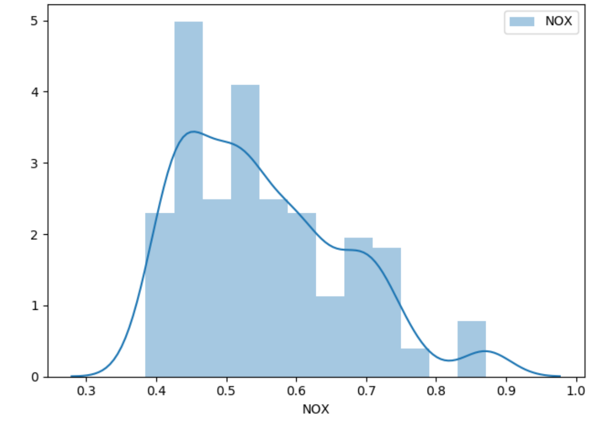
Missing Values

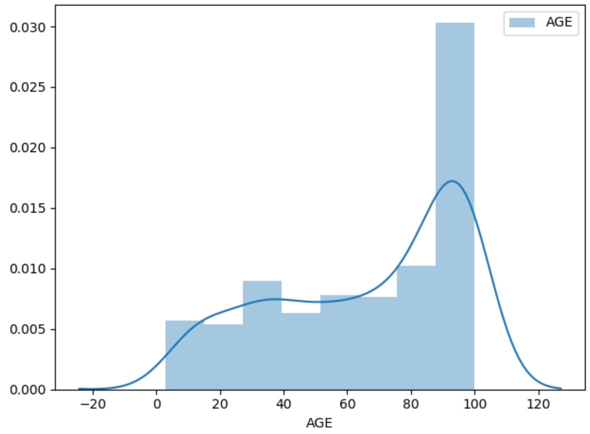
Three kinds of filling strategies:

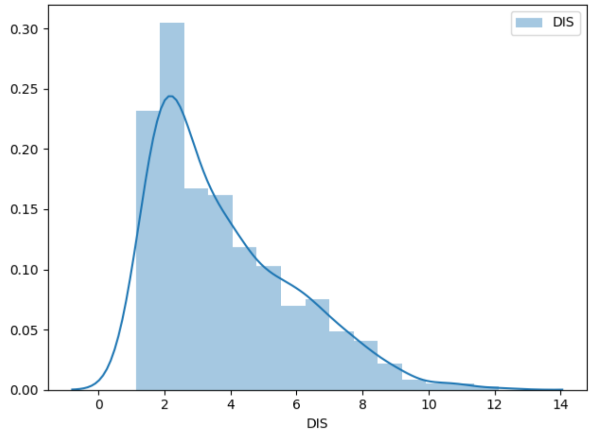
```
imp1 = SimpleImputer(missing_values=np.nan, strategy='median')
imp2 = SimpleImputer(missing_values=np.nan, strategy='mean')
imp3 = SimpleImputer(missing_values=np.nan, strategy='most_frequent')

df = boston_housing.copy()
imp1.fit(np.array(df['CRIM']).reshape(-1, 1))
df['CRIM'] = imp1.transform(np.array(boston_housing['CRIM']).reshape(-1, 1))
imp1.fit(np.array(df['ZN']).reshape(-1, 1))
df['ZN'] = imp1.transform(np.array(boston_housing['ZN']).reshape(-1, 1))
imp2.fit(np.array(df['INDUS']).reshape(-1, 1))
df['INDUS'] = imp2.transform(np.array(boston_housing['INDUS']).reshape(-1, 1))
imp3.fit(np.array(df['CHAS']).reshape(-1, 1))
df['CHAS'] = imp3.transform(np.array(boston_housing['CHAS']).reshape(-1, 1))
imp1.fit(np.array(df['AGE']).reshape(-1, 1))
df['AGE'] = imp2.transform(np.array(boston_housing['AGE']).reshape(-1, 1))
imp1.fit(np.array(df['LSTAT']).reshape(-1, 1))
df['LSTAT'] = imp1.transform(np.array(boston_housing['LSTAT']).reshape(-1, 1))
```

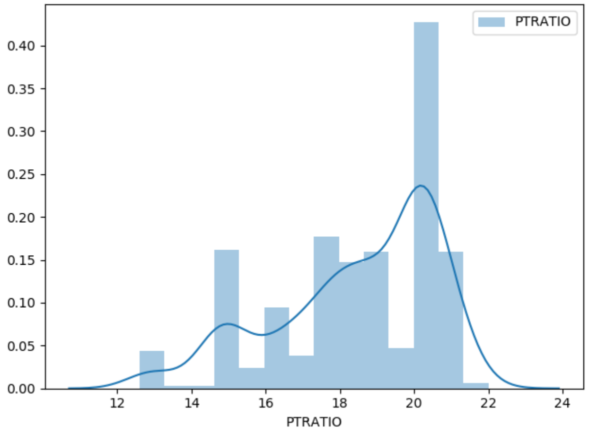
Data Preprocessing

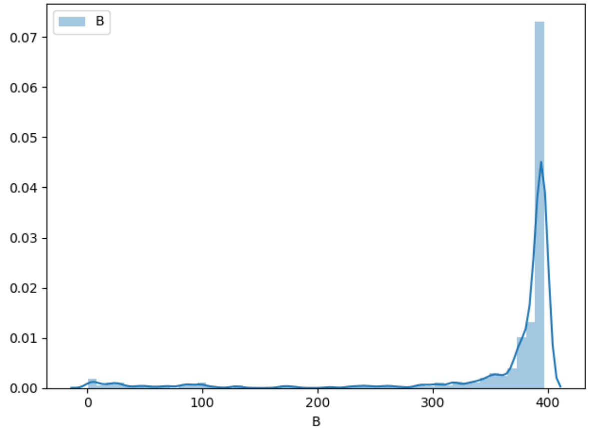
Transformation and Standardization











Non-linear transformation

```
pt = preprocessing.PowerTransformer()
features_pt = pt.fit_transform(features)
```

method='box-cox'
method='yeo-johnson'
Standardize=True

Box-Cox requires input data to be strictly positive.
Yeo-Johnson supports both positive and negative data.

Data Preprocessing

Feature Significance Test

sklearn.feature_selection.SelectKBest(score_func, k)

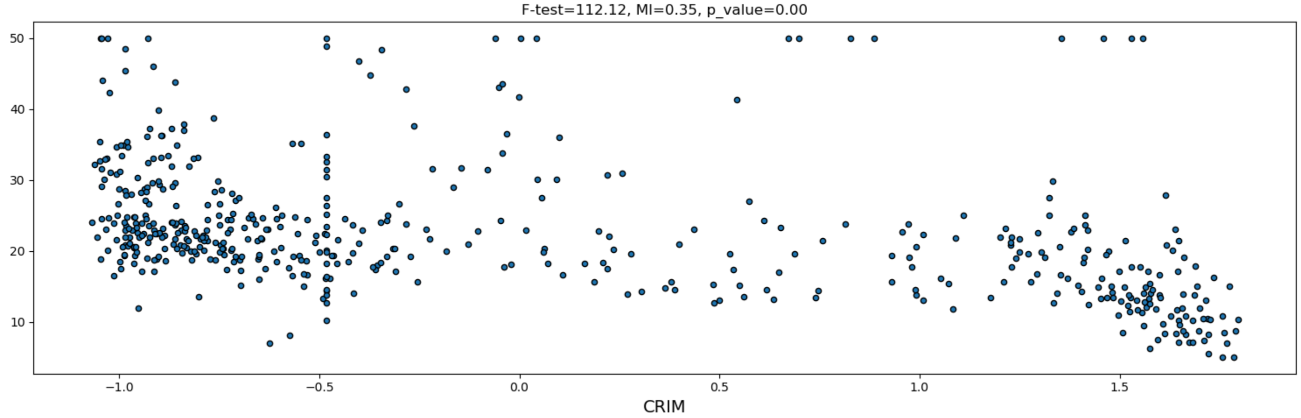
score_func:
For regression: f_regression, mutual_info_regression

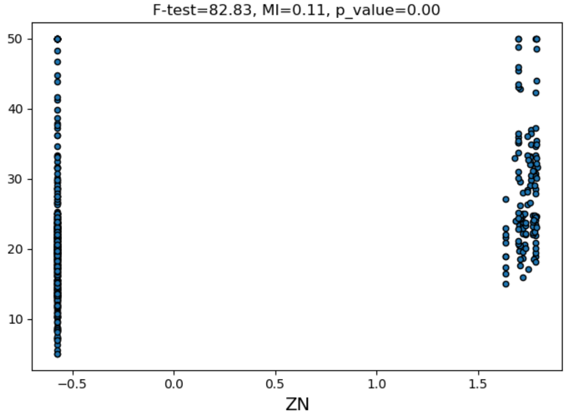
k:
k=int or “all”

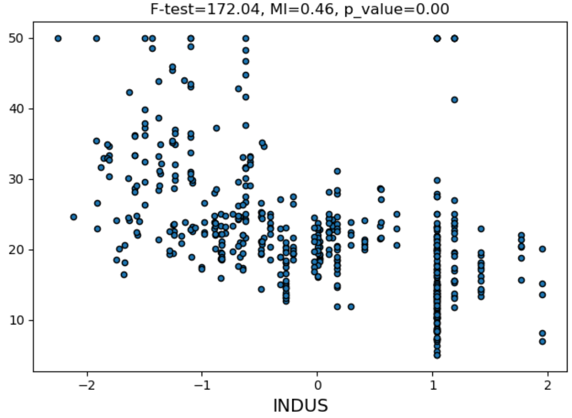
```
f_test, _ = f_regression(features_pt, price)
mi = mutual_info_regression(features_pt, price)
new = SelectKBest(f_regression, k='all')
new.fit_transform(features_pt, price)
print('p_values of features are:', new.pvalues_)
plt.figure(figsize=(15, 5))
for i in range(len(cols)):
    plt.scatter(features_pt[:, i], price, edgecolors='black', s=20)
    plt.xlabel("{}".format(cols[i]), fontsize=14)
    plt.title("F-test={:.2f}, MI={:.2f}, p_value={:.2f}".format(f_test[i], mi[i], new.pvalues_[i]))
    plt.show()
```

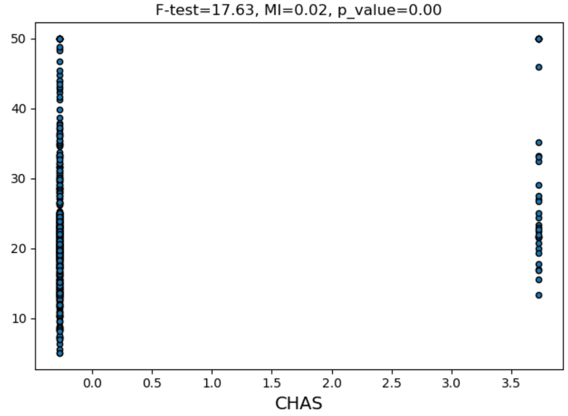
Data Preprocessing

Feature Significance Test



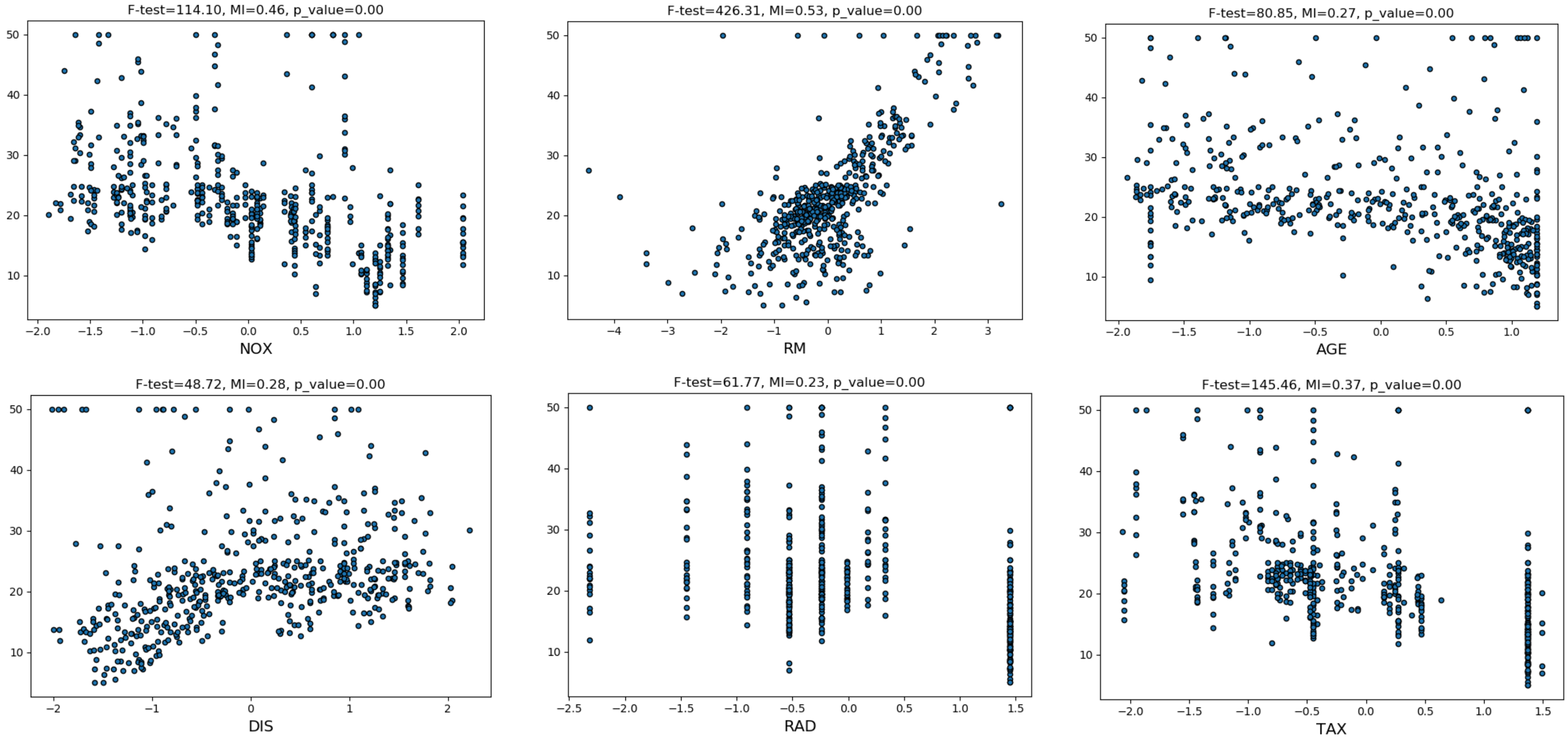






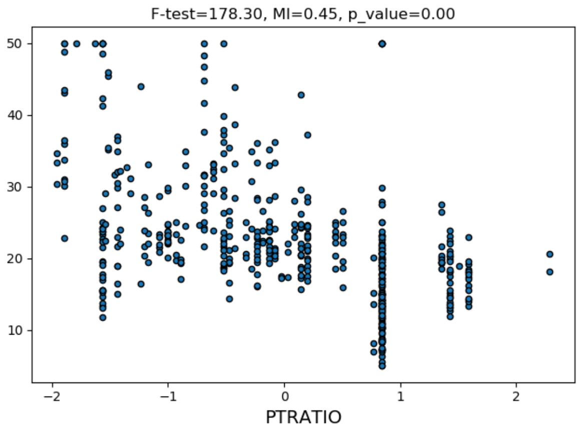
Data Preprocessing

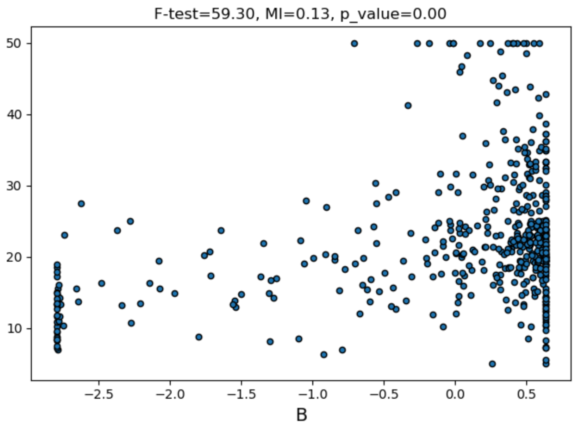
Feature Significance Test

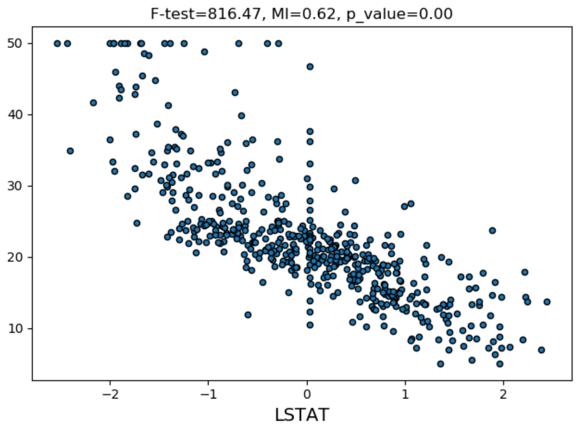


Data Preprocessing

Feature Significance Test







summary of p_value

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
8.59E-24	2.08E-18	5.06E-34	3.17E-05	3.79E-24	4.33E-69	4.92E-18

DIS	RAD	TAX	PTRATIO	B	LSTAT
9.35E-12	2.35E-14	1.32E-29	4.91E-35	7.22E-14	1.75E-107

PART 04

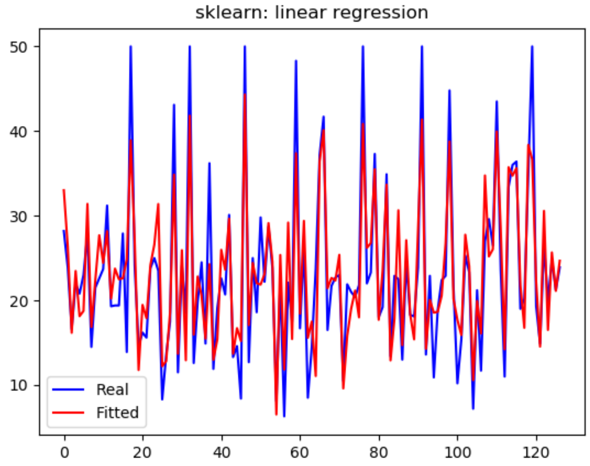
Linear Regression

Results

MSE & R^2:

MSE: 20.239 R^2: 0.796

Real vs. Fitted

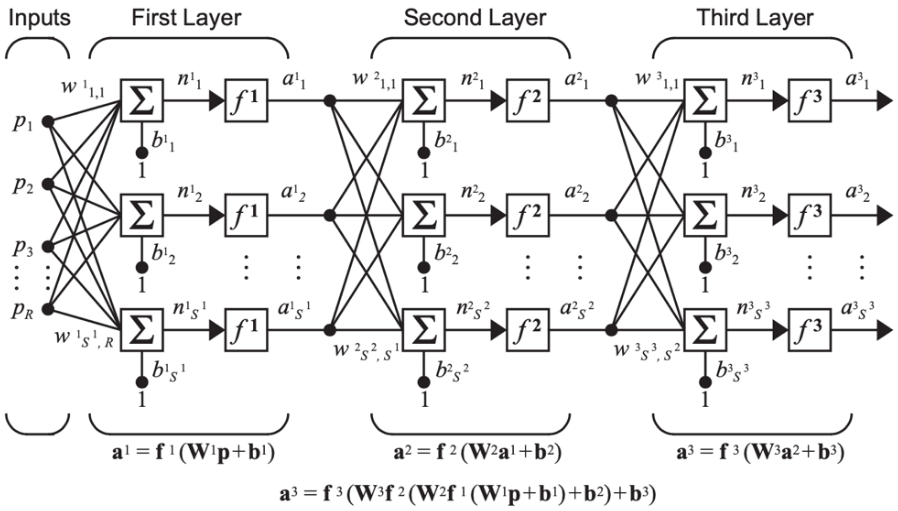



```python
# linear regression
lr1 = LinearRegression()
lr1.fit(x_train, y_train)
y_pred1 = lr1.predict(x_test)
plt.plot(range(len(y_test)), y_test, 'b', label='Real')
plt.plot(range(len(y_pred1)), y_pred1, 'r', label='Fitted')
plt.legend()
plt.title('sklearn: linear regression')
plt.savefig('Real vs. Fitted_LR.png')
plt.show()
print('The coefficients of linear regression model are:', lr1.coef_)
print('The intercept of linear regression model is:', lr1.intercept_)
print("MSE:", metrics.mean_squared_error(y_test, y_pred1))
print("R^2:", r2_score(y_test, y_pred1))
```

PART 05

Neural Network

Neural Network



Three-Layer Network

Neural Networks are a set of algorithms, modeled loosely on the human brain.

$$n = \mathbf{W}\mathbf{p} + b$$

Net Input

$$a = f(\mathbf{W}\mathbf{p} + b)$$

Neuron Output

Parameters

Model:

MLPRegressor()

Parameters:

activation	activation function for the hidden layer
solver	the solver for weight optimization
batch_size	size of minibatches for stochastic optimizers
max_iter	the maximum number of iterations
early_stopping	whether to stop when validation score is not improving

CV

```python
def selepara_RSCV(Model, params, X, Y):

    n_iter = np.arange(200, 400, 50)

    score = 0

    for i in n_iter:

        clf = RandomizedSearchCV(Model, params, n_iter=i, n_jobs=-1, cv=5)

        grid_result = clf.fit(X, Y)

        if grid_result.best_score_ > score:
            score = grid_result.best_score_
            parameters = grid_result.best_params_
            n = i

    return n, score, parameters
```

```python
# Parameters
hls = (100,)

optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']

param_random = dict(
    activation=['identity', 'logistic', 'tanh', 'relu'],
    solver=['lbfgs', 'sgd', 'adam'],
    max_iter=np.arange(7000, 15000, 1000),
    batch_size=np.arange(10, 200, 10),
    early_stopping=[False, True])

best_n_iter, best_score, paras = selepara_RSCV(NN, param_random, features, target)
```

- RandomizedSearchCV:

  grid search for parameters by randomly sampling in the parameter space

  Its search ability depends on the set ‘n_iter’ parameter.

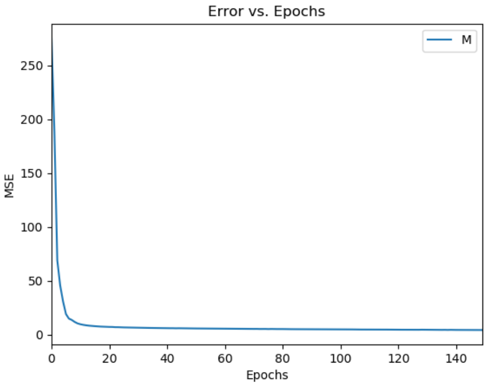
- Best Parameters:

Parameters	activation	solver	batch_size	max_iter	early_stopping
Value	relu	sgd	70	11000	True

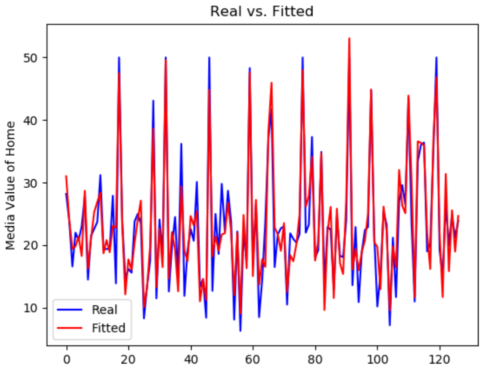
Results

MSE & R^2: MSE: 10.352. R^2: 0.896

Error vs. Epochs



Real vs. Fitted




```python
# Train the model
nn = MLPRegressor(hidden_layer_sizes=hls,
                  activation=paras['activation'],
                  solver=paras['solver'],
                  max_iter=paras['max_iter'],
                  early_stopping=paras['early_stopping'],
                  batch_size=paras['batch_size'],
                  random_state=50)
MSE, R2, Loss_curve, prediction = train_model(nn)

print('MSE:', MSE)
print('R2:', R2)

print('Number of iteration when the model converges:', nn.n_iter_)

# Plot results

# Error and Epochs
pd.DataFrame(Loss_curve).plot()
plt.title('Error vs. Epochs')
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.legend('MSE')
plt.savefig('Error vs. Epochs.png')
plt.show()

# Real vs Fitted
plt.plot(range(len(y_test)), y_test, c='b')
plt.plot(range(len(prediction)), prediction, c='r')
plt.title('Real vs. Fitted')
plt.ylabel('Media Value of Home')
plt.legend(['Real', 'Fitted'])
plt.savefig('Real vs. Fitted.png')
plt.show()
```

PART 06

Ensemble method

Ensemble method

Reason:

Data size → Bad generalization

Ensemble method

01 Bagging

random forest

02 Boosing

adaboost, Gradient boosting

Ensemble method

01 Random forest

How to improve the ability of generalization?

Resampling technique(Bootstrap)

Randomly select subfeature

Ensemble method

Bootstrap:

data set D : m samples,

randomly take a sample from D
with replacement into set D'

data set D':m samples.

Ensemble method

Randomly select subfeature

Traditional: k of features

information gain

RF : subfeature set

Ensemble method

RF Performance:

```
{'max_depth': 10, 'n_estimators': 100}
```

```
MSE: 9.3766947500555
R square: 0.905341820710926
```

Ensemble method

02 Gradient boosting

All of data

Estimator

weight value

Adaboost

Given dataset:

$$T = \{(x_1, y_1), (x_2, y_2), \cdots (x_N, y_N)\},\ y \in \{-1, +1\}$$

and base estimator $G_m(x)$

initialize weight $w_i^{(1)} = \frac{1}{N}\ , \quad i = 1, 2, 3, \cdots N$

for m=1 to M:

train G(x):

$$G_m(x) = \underset{G(x)}{\arg\min} \sum_{i=1}^{N} w_i^{(m)} \mathbb{I}(y_i \neq G(x_i))$$

calculate error rate

$$\epsilon_m = \frac{\sum\limits_{i=1}^{N} w_i^{(m)} \mathbb{I}(y_i \neq G_m(x_i))}{\sum\limits_{i=1}^{N} w_i^{(m)}}$$

update parameter

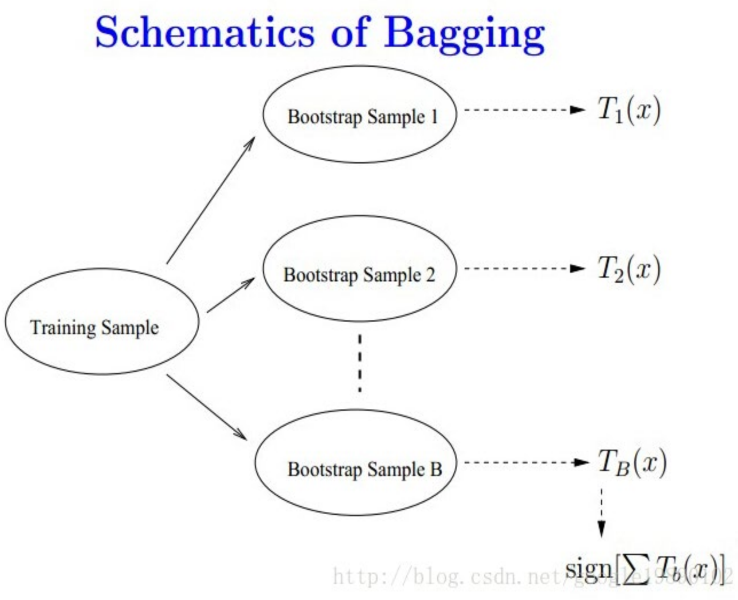
$$\alpha_m = \frac{1}{2} ln \frac{1-\epsilon_m}{\epsilon_m}$$

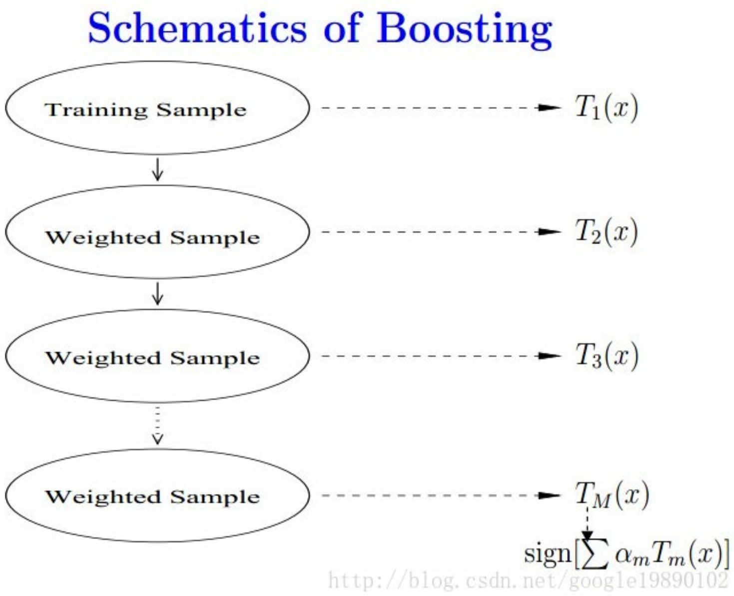
$$w_i^{(m+1)} = \frac{w_i^{(m)} e^{-y_i \alpha_m G_m(x_i)}}{Z^{(m)}}\ , \qquad i = 1, 2, 3 \cdots N$$

final model:

$$G(x) = sign\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$$

Schematics


Schematics of Bagging
Training Sample
Bootstrap Sample 1
Bootstrap Sample 2
Bootstrap Sample B
$T_1(x)$
$T_2(x)$
$T_B(x)$
$\text{sign}[\sum T_b(x)]$
http://blog.csdn.net/google19890102


Schematics of Boosting
Training Sample
Weighted Sample
Weighted Sample
Weighted Sample
$T_1(x)$
$T_2(x)$
$T_3(x)$
$T_M(x)$
$\text{sign}[\sum \alpha_m T_m(x)]$
http://blog.csdn.net/google19890102

Ensemble method

GB Performance:

```
{'min_samples_leaf': 3, 'n_estimators': 150}
```

```
MSE: 9.126177702162598
R square: 0.9078708022194981
```

PART 07

Summary

Model	MSE	R square
Linear regression	20.239	0.796
Neural network	10.352	0.896
Random forest	9.376	0.905
Gradient boosting	9.126	0.907

Ensemble method is better.

PART 08

Q&A