6202 Project：Prediction of Boston Housing Price

Jinyi Shang

Chaohui Li

Yu Cao

The George Washington University

Contents

1. Introduction

A house value is simply more than location and square footage. Like the features that make up a person, an educated party would want to know all aspects that give a house its value. This project is going to be focused on solving the problem of predicting house prices for house buyers and house sellers.

In the project, we will evaluate the performance and predictive power of models that has been trained and tested on data collected from houses in suburbs of Boston, Massachusetts. Models trained on this data that is seen as a good fit could then be used to make certain predictions about a house – in particular, its monetary value. The models would prove to be invaluable for someone like a real estate agent who could make use of such information on a daily basis, or someone like you and me who would like to find their dream home with a reasonable price tag.

If you are studying data science, you probably heard of the Boston housing dataset, which is a very ubiquitous dataset. This dataset contains information about 506 census tracts of Boston from the 1970 census. And each of the 506 entries represent aggregated data about 13 features for homes from various suburbs in Boston, Massachusetts, including average number of rooms per dwelling, pupil-teacher ratio, per capita crime rate and so on.

Like ‘hello world’, the Boston Housing Dataset has become part of a common vocabulary. And it will remain so, not only because thoroughly labeled datasets for machine learning are still not that easy to find, but because using the same dataset for decades to test different algorithms has allowed scientists to control for that variable and highlight the differences in algorithm performance. However, the Boston housing dataset is small, especially in today’s age of big data. If the amount of data is too small, the model is likely to appear overfitting, resulting in poor model performance. In order to reduce the impact of this problem on the training model, we intend to optimize the data preprocessing and model selection by, for example, reducing the number of features, cross-validation, and reducing the complexity of the model.

We are going to break everything into logical steps that allow us to ensure the cleanest, most realistic data for our model to make accurate predictions from:

1. Load Data and Packages
2. Preprocessing of Dataset
   1.1.Analyzing
   1.2.Impute Missing Data and Clean Data
   1.3.Feature Transformation/Engineering
3. Modeling and Predictions
   1.1.Linear Regression

1.2.Neural Network
1.3.Random Forest
1.4.Gradient Boost

2. Description of Dataset

2.1 Overview of the dataset.

There are total 14 columns in the dataset. The column “MEDV” is the median of house price of self-occupied house. Other columns are features that are related to the house price.

Table2- 1 The overview of the dataset

#	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

2.2 The statistics information.

“CHAS” is a dummy variable, other variables are all continuous variables. According to the minimum, median and maximum, “CRIM”, “ZN”, “AGE”, “B” and “LSTAT” are skewed distributed. According to the count, some columns exist missing values.

Table2- 2 The statistics information of the dataset

	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

2.3 Missing values.

The columns that exist missing values are “CRIM”, “ZN”, “INDUS”, “CHAS”, “AGE” and “LSTAT”. Then, we summarize the amount of missing values for each column. They are shown as follows:

Table2- 3 The summary of missing values

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

3. Models and Algorithm

3.1 Feature engineering -transformation.

a. Box-Cox transformation.

Box-Cox transformation is a generalized power transformation method which is commonly used in statistical modeling. It is used when continues variables do not satisfy the normal distribution. After the Box-Cox transformation, the correlation between unobservable errors and predictors can be reduced to some extent. The main feature of the Box-Cox transformation is the introduction of a parameter, which is estimated by the data itself to determine the form of data transformation that should be adopted. The Box-Cox transformation can significantly improve the normality, symmetry and variance equality of the data. The characteristic of the Box-Cox transformation is that it can only be applied to samples containing positive values. The transfer function is shown as follows:

$$y^{(\lambda)} = \begin{cases} \dfrac{y^{\lambda} - 1}{\lambda}, & \lambda \neq 0 \\ \log y, & \lambda = 0 \end{cases}$$

We need to determine the transformation parameter λ so that $y^{(\lambda)}$ satisfies

$$y^{(\lambda)} = X\beta + e, e \sim N(0, \sigma^2 I)$$

We use the maximum likelihood method to determine the transformer parameter λ. For fixed λ, β and σ^2, the likelihood function is

$$L(\beta,\sigma^2) = \frac{1}{(\sqrt{2\pi}\sigma)^n} \exp\left\{-\frac{1}{2\sigma^2}\left(y^{(\lambda)} - X\beta\right)'\left(y^{(\lambda)} - X\beta\right)\right\} J$$

$$J = \prod_{i=1}^{n} \left|\frac{dy_i^{(\lambda)}}{dy_i}\right| = \prod_{i=1}^{n} y_i^{\lambda-1}$$

Then we can obtain the maximum likelihood estimation of β and σ^2, the residual

sum of squares and the responding maximum of likelihood function. It is a unary function for λ, so we can easily get the value of λ.

b. Yeo-Johnson transformation.

Yeo-Johnson transformation is also one of the power transformation methods by constructing a set of monotone functions to transform data of random variables. It can reduce the heteroscedasticity of random variables and amplify its normality, making its probability density function form closer to normal distribution. The characteristic of the Yeo-Johnson transformation is that it can be applied to samples containing 0 and negative values, so it is also considered to be the extension of the Box-Cox transform in the real filed. The transfer function is shown as follows:

$$y^{(\lambda)} = \begin{cases} \log(X+1) & if\ \lambda = 0, X > 0 \\ \dfrac{(X+1)^{\lambda} - 1}{\lambda} & if\ \lambda \neq 0, X > 0 \\ -\log(-X+1) & if\ \lambda = 2, X \leq 0 \\ \dfrac{-\left[(-X+1)^{(2-\lambda)} - 1\right]}{2-\lambda} & if\ \lambda \neq 2, X \leq 0 \end{cases}$$

As for the determination of λ, it is the same as the Box-Cox transformation. It also uses the maximum likelihood estimation to determine the value of λ.

Yeo-Johnson transformation is one of the data standardization methods used in the data preprocessing stage of data mining and machine learning. When the normality of the random variables is poor, it is processed using the Yeo-Johnson transformation, which is beneficial to the modeling.

3.2 Standardization.

Standardization is also called zero-mean normalization. After standardization, the mean value of the data is 0 and the standard deviation is 1. The conversion formula is:

$$x_i^{'} = \frac{x_i - \bar{x}}{s}$$

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

The z-score standardization method is suitable for cases when there is outlier data that exceeds the value range. It is currently the most commonly used data standardization method.

3.3 Multiple linear regression.

3.3.1 Definition.

Multiple linear regression is a machine learning algorithm based on supervised learning. It is a statistical approach to modeling the relationship between a dependent variable and a given set of independent variables.

3.3.2 Expression.

$$y = b_0 + b_1 x_1 + \ldots + b_n x_n$$

y is the dependent variable, x_1 ...x_n are multiple independent variables.

Also, we can vectorize the above expression as follows:

$$y = Xb$$

3.3.3 Cost function.

We use the residual sum of squares between the observed targets in the dataset and the targets predicted by the linear approximation as the cost function of the linear regression model.

$$S = \|Xb - y\|^2$$

By differentiating and minimizing S, we can get

$$X^T Xb = X^T y$$

If the matrix $X^T X$ is not singular, then there is a unique solution for b:

$$b = (X^T X)^{-1} X^T y$$

3.4 Neural network.

Neural Networks (Like the network shown in Figure 1) are a set of algorithms, modeled loosely on the human brain, a neural net consists of thousands or even millions of simple processing nodes that are densely interconnected. Most of today's neural nets are organized into layers of nodes, and they're "feed-forward," meaning that data moves through them in only one direction. An individual node might be connected to several nodes in the layer beneath it, from which it receives data, and several nodes in the layer above it, to which it sends data.

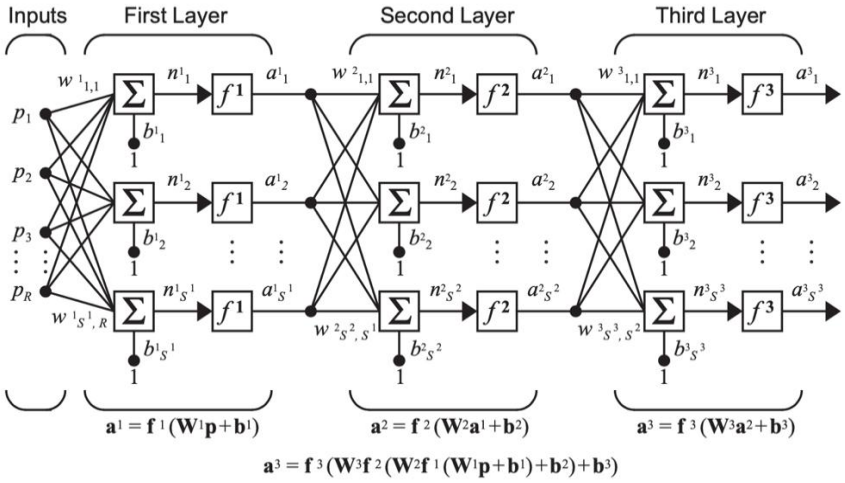

Figure 1. A Three-Layer Network

To each of its incoming connections, a node will assign a number known as a "weight." When the network is active, the node receives a different data item — a different number — over each of its connections and multiplies it by the associated weight. It then adds the resulting products together, yielding a single number, which is called net input, as the Equation 1showing.

$$n = \mathbf{W}\mathbf{p} + b \qquad \text{Equation 1}$$

Then, the neuron output can be calculated as Equation 2, depending on the particular transfer function that is chosen.

$$a = f(\mathbf{W}\mathbf{p} + b)$$ Equation 2

When a neural net is being trained, all of its weights and biases are initially set to random values. Training data is fed to the bottom layer — the input layer — and it passes through the hidden layers, getting multiplied and added together in complex ways, until it finally arrives, radically transformed, at the output layer. During training, the weights and biases are continually adjusted until training data with the same labels consistently yield similar outputs.

3.5 Ensemble methods.

After we use linear regression to fit our Boston housing Data and obtain the result (MSE and r square) as the benchmark for our project, we use neural network. However, there is an obvious problem that the amount of our data is not large enough, which may cause underfitting and return a bad result. Therefore, we try to use ensemble method to strength our learner to get a good result.

Ensemble methods aims to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator. These base estimators, which called weak learner, generally do not have a good performance on a given data. But if they have strong dependency and we can use this dependency to update the weight, for example, Strengthen the learning weight of misclassified data or they don't have strong dependency and we can combine them, we can improve these weak learner to a strong learner. Boosting is the representative method of the former, and the latter is bagging method and random forest. They are two broad categories of ensemble methods.

In our project, we will try these two methods: Gradient Boosting and Random forest to improve the result. Actually, it turns out that with a small sample of data, the performance of ensemble methods is better than our traditional machine learning models.

3.5.1 Random forest

Before we talk about random forest, we need to understand the core of this method—— resampling technique and randomly choosing feature. The first technique we also call it bootstrap methods:

Now Given data set D containing m samples, randomly take a sample from D with replacement into set D' and sampled for m times to get the data set D'. Then, D' have m samples. It's clear that some of the samples in D will show up multiple times, and some of the samples won't.

So we estimate the probability that the sample is never picked in m samples. It will be (1-1/m)^m. set m->∞:

$$\lim_{m \mapsto \infty} \left(1 - \frac{1}{m}\right)^m \mapsto \frac{1}{e} \approx 0.368$$

Which means there are about 36.8% samples in D will never show up in D'.This is bootstrap, which we used to build our resampling data in random forest.

The second variant on random forest is when we choose a feature to classify our data on our base estimator, traditional decision tree will calculate all of information gain and choose the biggest one. In random forest, we first randomly choose k features from all of the features, and use these k features to calculate information gain. This change improve the independency of each tree, which optimized the model. Generally, we will set k=log2 d, where d is the number of all features.

Now, let's summarize the algorithm of random forest:

First, use bootstrap build m number of subset.

For each subset, train a decision tree or regression tree as base estimator.

In each base estimator, randomly select "K" features from total "m" features where k << m

Among the "K" features, calculate the node "d" using the best split point

Split the node into daughter nodes using the best split

Build forest by training "m" number of trees.

3.5.2 Boosting

Boosting is a very powerful learning method, which is also a supervised categorization learning method. It combines many weak learners to produce a strong learner. The performance of a weak classifier is only slightly better than that of random selection, so it can be designed to be very simple and without too much computational cost. Many weak classifiers are combined to form an integrated strong classifier similar to SVM or neural network.

Now the question is how to use boosting to build a rule? We can do it in the following steps:

Step 1: The basic learner should have the same weight for each observation

Step 2: If the first basic learning algorithm makes a wrong prediction, the point has a higher weight in the next basic learning algorithm

Step 3: Iterate step 2 until a predetermined number of learners or a predetermined prediction accuracy is reached.

Finally, the output of a number of weak learner combined into a strong learner, improve the overall prediction accuracy of the model.Boosting is always more concerned with the weak rules that are misclassified.

3.5.2.1 Adaboost.

Here, a famous algorithm of boosting method is Adaboost.

If we have a classification problem, given a training sample set, it is often much easier to derive a rough classification rule (weak classifier) from this classification problem than an accurate classification rule (strong classifier).The promotion method is to get a series of weak classifiers (also known as basic classifiers) by repeated learning from the weak learning algorithm, and then form a strong classifier by combining these weak classifiers. Most of the promotion methods are to change the probability distribution of training data (or the weight distribution of training data) to call the weak learning algorithm to learn a series of weak classifiers for different training distributions. As a result, there are two issues that need to be addressed for the promotion approach:

How to change the weight or probability distribution of training data in each round?

How to combine a weak classifier into a strong classifier?

AdaBoost approach to the first problem is to raise the weights of samples that were misclassified by the previous round of weak classifiers and lower the weights of samples that were correctly classified. After a round of weight increase, the weak classifier will pay more attention to the samples that are not properly classified. As it goes on, the classification problem is "divided and ruled" by a series of weak classifiers. For the second problem, namely the combination of weak classifiers, AdaBoost adopts the weighted majority voting method. Specifically, it is to increase the weight of weak classifiers with small error rate so that they can play a greater role

in voting; on the other hand, it is to reduce the weight of weak classifiers with large error rate so that they can play a smaller role in voting.

3.5.2.2 Gradient boost.

The next is what we used in our problem called Gradient boosting. It's a generalization of AdaBoost.

Gradient boosting involves three elements:

A loss function to be optimized.

A weak learner to make predictions.

An additive model to add weak learners to minimize the loss function.

a. Loss Function.

The loss function used depends on the type of problem being solved. It must be differentiable, but many standard loss functions are supported and you can define your own. For example, regression may use a squared error and classification may use logarithmic loss.

A benefit of the gradient boosting framework is that a new boosting algorithm does not have to be derived for each loss function that may want to be used, instead, it is a generic enough framework that any differentiable loss function can be used.

b. Weak Learner.

Decision trees are used as the weak learner in gradient boosting. Specifically regression trees are used that output real values for splits and whose output can be added together, allowing subsequent models outputs to be added and "correct" the residuals in the predictions. Trees are constructed in a greedy manner, choosing the best split points based on purity scores like Gini or to minimize the loss. Initially, such as in the case of AdaBoost, very short decision trees were used that only had a single split, called a decision stump. Larger trees can be used generally with 4-to-8 levels. It is common to constrain the weak learners in specific ways, such as a maximum number of layers, nodes, splits or leaf nodes. This is to ensure that the learners remain weak, but can still be constructed in a greedy manner.

c. Additive Model.

Trees are added one at a time, and existing trees in the model are not changed.

A gradient descent procedure is used to minimize the loss when adding trees.

Traditionally, gradient descent is used to minimize a set of parameters, such as the

coefficients in a regression equation or weights in a neural network. After calculating error or loss, the weights are updated to minimize that error.

Instead of parameters, we have weak learner sub-models or more specifically decision trees. After calculating the loss, to perform the gradient descent procedure, we must add a tree to the model that reduces the loss (i.e. follow the gradient). We do this by parameterizing the tree, then modify the parameters of the tree and move in the right direction by (reducing the residual loss.

Generally this approach is called functional gradient descent or gradient descent with functions.

3.6 Evaluation index of models.

3.6.1 Mean squared error.

Mean squared error (MSE) of an estimator measures the average of the squares of the errors – that is, the average squared difference between the estimated values and the actual value. MSE is a risk function, corresponding to the expected value of the squared error loss. It is always non-negative, and values closer zero are better.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y}_i)^2$$

3.6.2 R-squared.

R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression. It is the percentage of the response variable variation that is explained by a linear model. In general, the higher the r-squared, the better the model fits the data. The formula is:

$$R^2 = 1 - \frac{unexplained\ variation}{total\ variation}$$

4. Experimental Setup

4.1 Data preprocessing.

4.1.1 Filling the missing values.

a. The distribution of features that exist missing values.

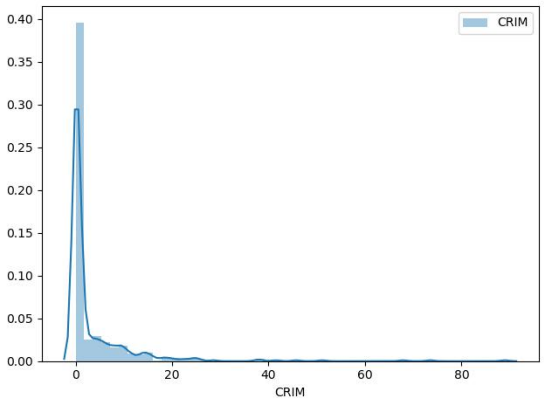

Figure4- 1 The distribution of "CRIM"

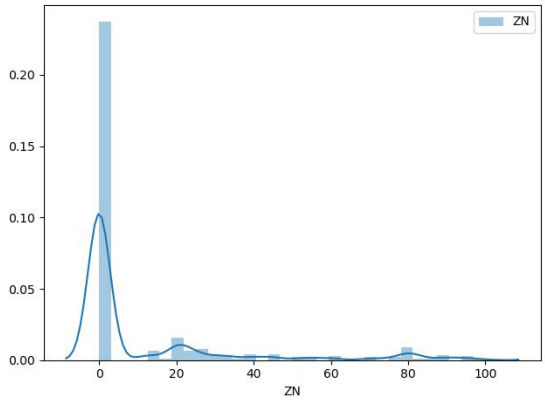

Figure4- 2 The distribution of "ZN"

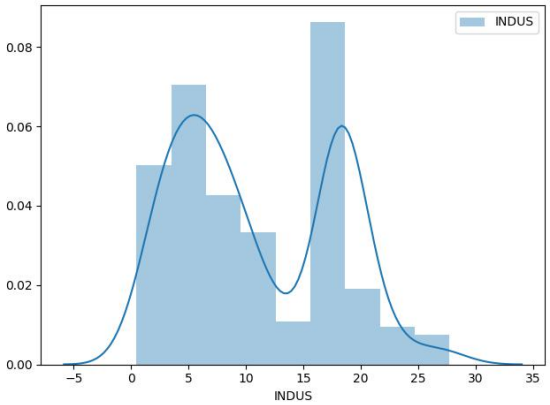

Figure4-3 The distribution of "INDUS"

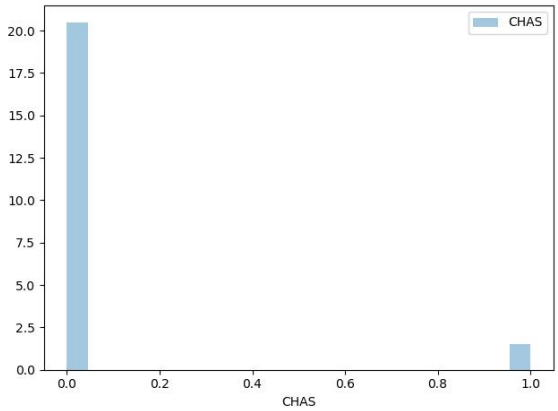

Figure4- 4 The distribution of "CHAS"

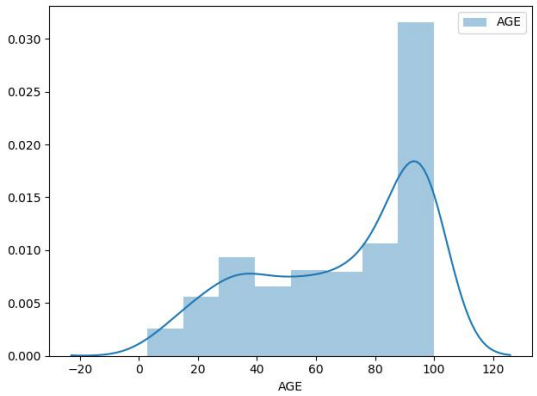

Figure4- 5 The distribution of "AGE"

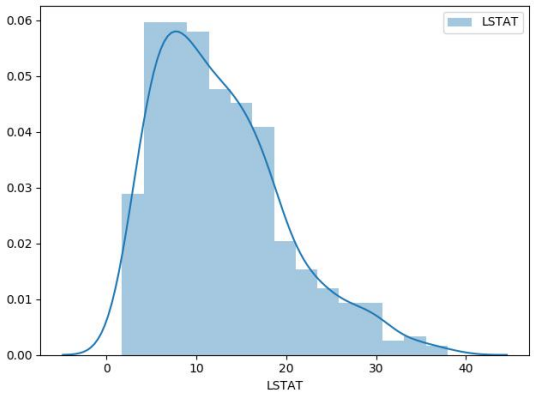

Figure4- 6 The distribution of "LSTAT"

b. Filling methods.

For the "INDUS" feature, its distribution does not show obvious skewness, so we choose the "mean" method to fill the missing values. For the "CRIM", "ZN", "AGE" and "LSTAT" features, their distributions show obvious skewness, so we choose the "median" method to fill the missing values. For the "CHAS" feature, it is categorical data, so we choose the "mode" method to fill the missing values.

In the Python, we use the Imputer package to fill the missing values with different strategies.

4.1.2 Non-linear Transformation and standardization.

For the dataset, it has both positive and negative samples, so we use Yeo-Johnson transformation as our transform function. Because it can be applied to samples containing 0 and negative values. After the process, the dataset became more Gaussian-like.

In the Python, we use the Power Transformer to achieve the non-linear transformation and standardization.

4.1.3 Significance of Features.

We use univariate feature selection to select significant features. It works by selecting the best features based on univariate statistical tests. It can be seen as a preprocessing step to an estimator. The method based on F-test estimate the degree of linear

dependency between two random variables. On the other hand, mutual information methods can capture any kind of statistical dependency, but being nonparametric, they require more samples for accurate estimation.

In the Python, SelectKBest removes all but the k highest scoring features and return the p_value for every feature. For regression problems, we would better use f_regression and mutual_info_regression as our score function. The p_values for each column is shown as follows:

Table4- 1 The significance of features

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

We find that all features are significant because that all p-values are smaller than 0.05.

Then we plot the relationship between the target price and every feature separately and calculate the f-test score and mutual info score. Also, it indicates that all features are significant.

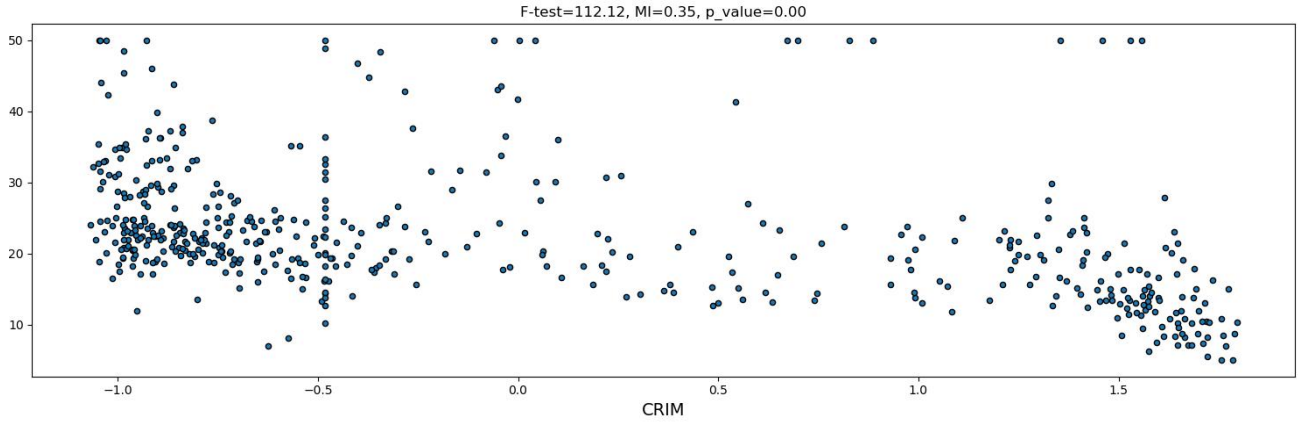

Figure4- 7 The significance of "CRIM"

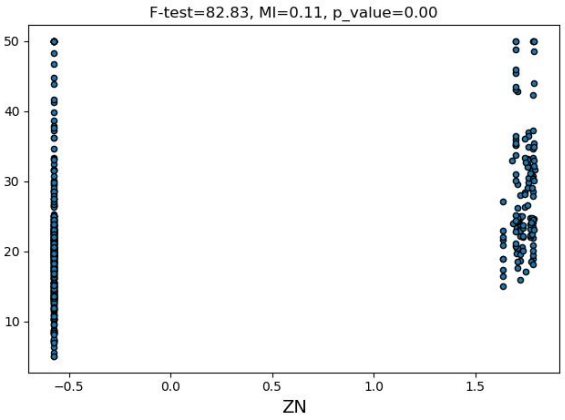

Figure4- 8 The significance of "ZN"

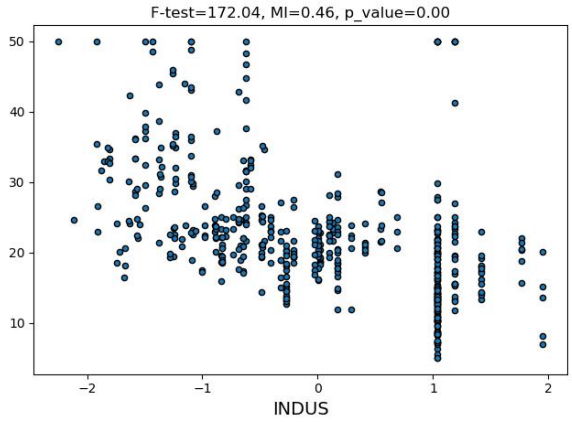

Figure4- 9 The significance of "INDUS"

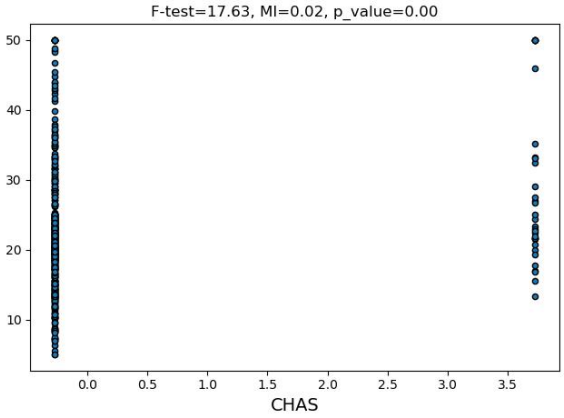

Figure4- 10 The significance of "CHAS"

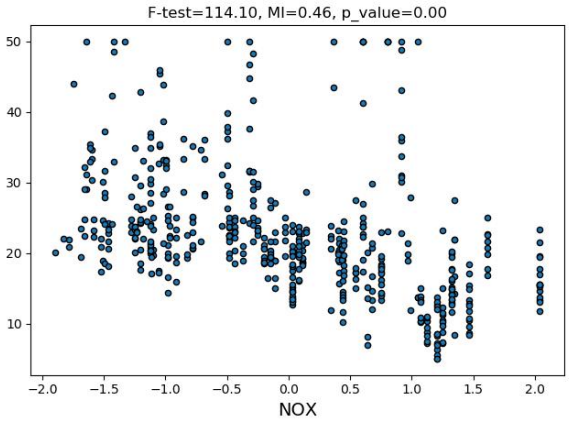

Figure4- 11 The significance of "NOX"

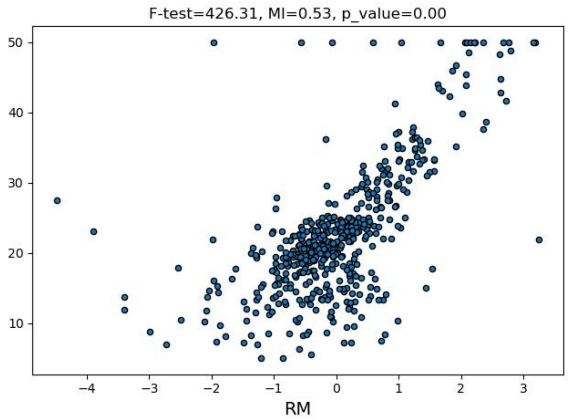

Figure4- 12 The significance of "RM"

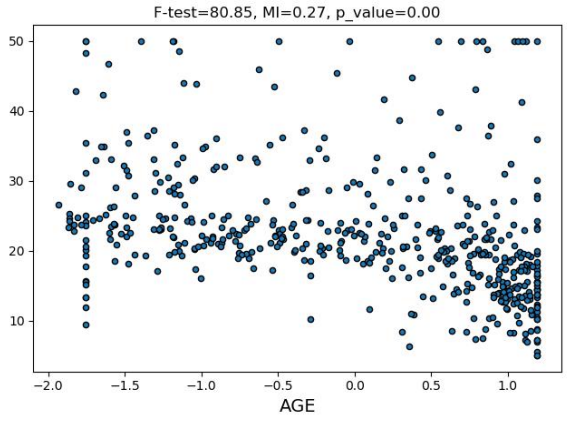

Figure4- 13 The significance of "AGE"

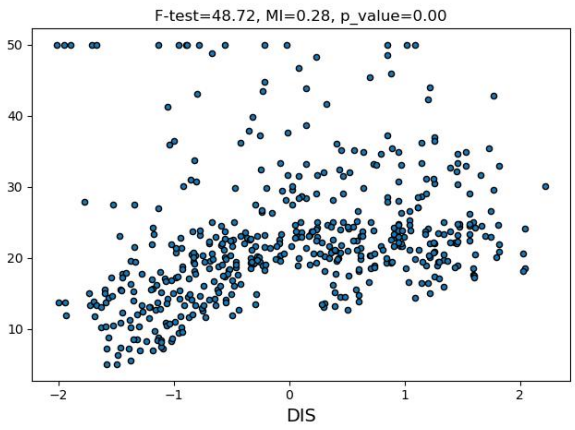

Figure4- 14 The significance of "DIS"

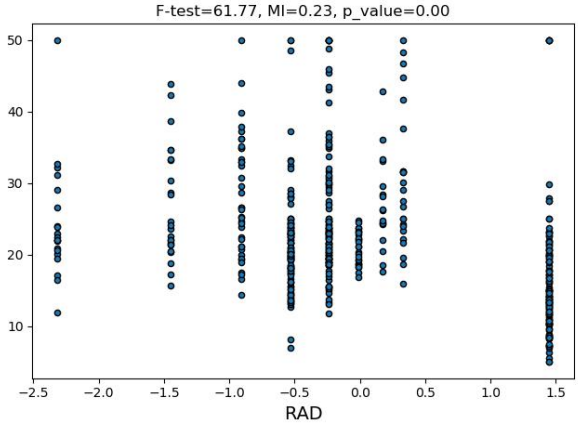

Figure4- 15 The significance of "RAD"

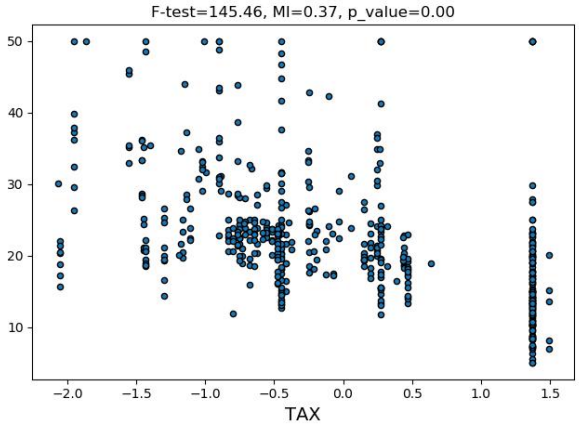

Figure4- 16 The significance of "TAX"

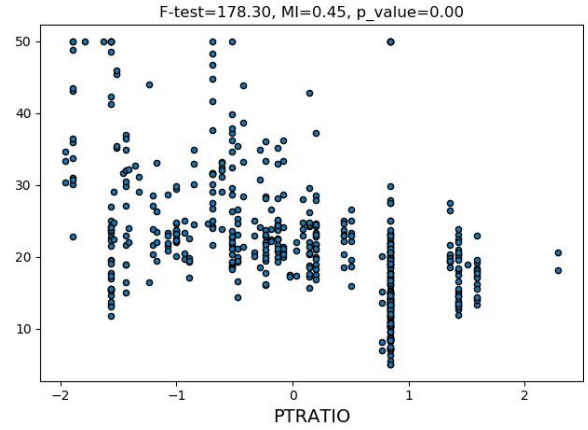

Figure4- 17 The significance of "PTRATIO"

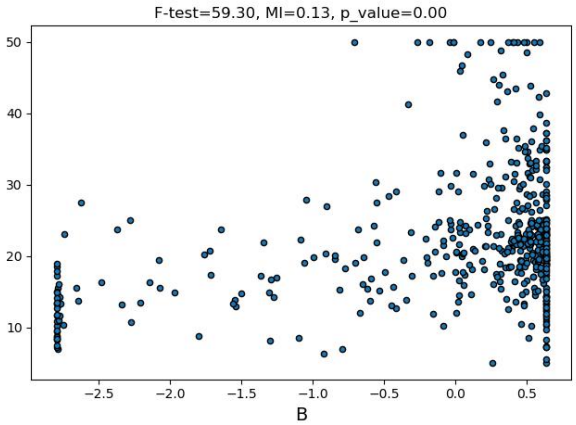

Figure4- 18 The significance of "B"

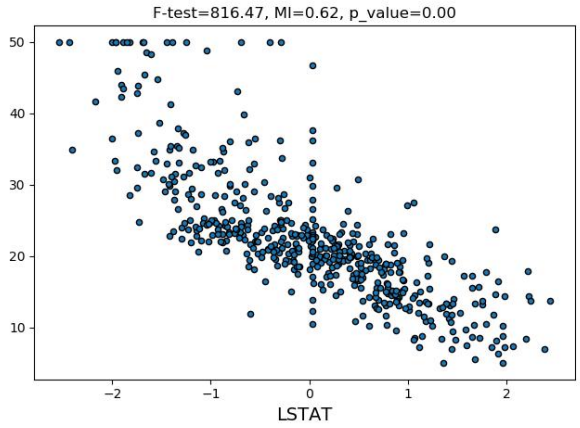

Figure4- 19 The significance of "LSTAT"

4.2 Linear regression.

First, we separate the dataset to train data and test data. Then, we use the Linear Regression to achieve the ordinary least squares. In the python, we set all parameters as default and get the coefficients for features and the intercept. At the same time, we can calculate the mean square error and R square as the index to judge our model's performance. It is also the basis for judging other machine learning models.

4.3 Neural network.

For the case of the project, after preprocessing data, we selected 'MEDV' as target, and rest of variables as features. Then, 'train_test_split()' function is used to split data to train and test data. The parameter 'random_state' of this step in all models is set as 1 so that the results of all models are comparable.

Here, 'LMPRegressor()' is chosen to be used as Neural Network model in PyCharm. It contains many parameters like 'activation', 'solver', 'max_iter', 'batch_size', 'early_stopping' and so on.

To get the best parameters, a new python file is written to adjust the different parameters mentioned above. Here we use both 'GridSearchCV' and 'RandomizedSearchCV' methods to find the best parameters.

'GridSearchCV' is grid search and cross-validation. Grid search, search for parameters, that is, adjust the parameters in order according to the step size within the specified parameter range, use the adjusted parameters to train the learner, and find the parameters with the highest accuracy on the verification set from all the parameters, which is actually a cycle and comparison process. It can guarantee to find the most accurate parameters within the specified parameter range, but this is also the defect of grid search. It requires traversing all possible parameter combinations, which is very time-consuming in the face of large data sets and multiple parameters.

'RandomizedSearchCV' is used in the same way as 'GridSearchCV', but it replaces 'GridSearchCV''s grid search for parameters by randomly sampling in the parameter space. For parameters with continuous variables, 'RandomizedSearchCV' will sample it as a distribution. This is what grid search cannot do. Thus, we use 'RandomizedSearchCV' finally. Its search ability depends on the set 'n_iter' parameter.

Then, we train the model using the best parameters by the customized function 'train_model()' and get MSE and R^2 score, which are used to judge the performance of model.

Finally, the return value ‘loss_curve_’ and prediction are used to plot the results.

4.4 Ensemble method.

After preprocessing, we can directly use the API from sklearn to run our model.

First, we use **from sklearn.model_selection import train_test_split** to split our data with test size as default, random_state=1. Then, use **RandomForestRegressor** and **GradientBoostingRegressor** API separately in **sklearn.ensemble** to run our model.

About the parameter selection, we use **from sklearn.model_selection import GridSearchCV** to choose the best combination of the number of estimators and max_depth.

After training, we will use r square and MSE to judge the performance of our model

5. Results

5.1 Linear regression.

5.1.1 The coefficients of the linear regression model.

Table5- 1 The coefficients of the linear regression model

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
0.1286	0.1582	-0.9226	0.8054	-2.2907	1.6932	-0.0723

DIS	RAD	TAX	PTRATIO	B	LSTAT	Intercept
-3.8692	1.0019	-1.2698	-1.3697	0.1950	-5.2169	22.5933

5.1.2 The performance of the model.

The mean square errors is 20.2388 and R-square is 0.7957, which means that almost 80% change of house price can be explained by the linear regression model.

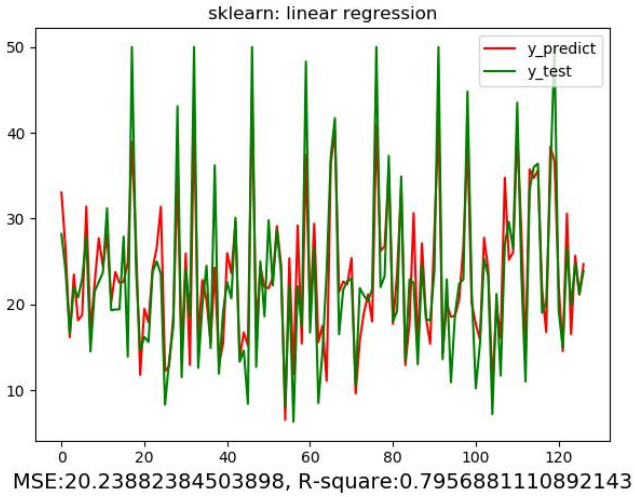

Figure5- 1 The comparison between the predicted value and true value

5.2 Neural network.

In adjusting parameter part, we focus on 'activation', activation function for the hidden layer; 'solver', the solver for weight optimization; 'batch_size', size of minibatches for stochastic optimizers, which is useful when the model is not converge; 'max_iter', the maximum number of iterations, which is also an important parameter for convergence; 'early_stopping', whether to use early stopping to terminate training when validation score is not improving.

After using cross validation, we get the best parameters shown in Table 1.

Table5- 2 Best Parameters

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

Note:The number of sampling without replacement 'n_iter' in 'RandomizedSearchCV' is 350. Best score in 'RandomizedSearchCV' is 0.622.

Parameter 'relu' is the rectified linear unit function as Equation 3 expresses

$$f(x) = \max(0, x) \qquad \text{Equation 3}$$

The advantage of this activation is that the convergence rate of SGD obtained by using ReLU will be much faster than sigmoid/tanh. It can get the activation value

without calculating a lot of complicated operations, compared to sigmoid/tanh.

Parameter ‘sgd’ refers to stochastic gradient descent. Since it is not a loss function on all training data, but a loss function on a certain training data in each iteration, so that the update speed of each round of parameters is greatly accelerated.

In results after training model part, we get MSE and R^2 score to see whether the model is good.

MSE is a risk function, which can be calculated by Equation 4, corresponding to the expected value of the squared error loss. MSE is almost always strictly positive. It is a measure of the quality of an estimator. The closer to zero the value is, the better the model is.

$$MSE = \frac{1}{n}\sum_{i}(y_i - y_{\text{pred}})^2 \qquad \text{Equation 4}$$

Here, MSE of neural network model is 10.352, which is close to zero, indicating that the model performs well.

R squared score is a statistical measure of how close the data pairs in a set are to their fitted regression line. This measure ranges from 0 to 1, indicating the extent to which the dependent variable in a data set is predictable. A R squared of 0 means that the dependent variable cannot be predicted by the independent variable, while 1 means that it can be predicted without error. Thus, the larger R squared score is, the more predictable the dependent variables are. The result of neural network model is 0.896, which also shows that the model performs well.

What’s more, two images are also improved to judge the performance.

Firstly, we plot the image of error and epochs (Figure 2). We can see that as the number of iterations increases, the mean square error of the model gradually decreases. When the number of iterations reaches about 150, the mean square error of the model has been reduced to 0. This indicates that the model has converged at this time, which is the same as the ‘n_iter’ of model.

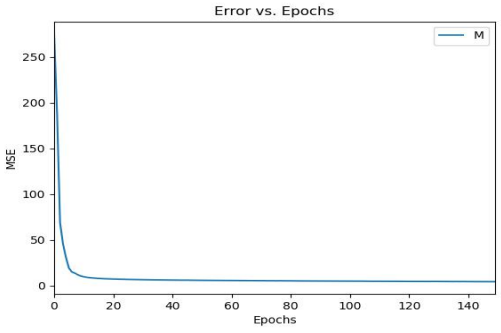

Figure5- 2. Error vs. Epochs

Secondly, we plot the fitted curve and true value curve (Figure 3). The blue line in Figure 3 represents real value of house price, while red line represents the fitted value predicted by neural network model. We can see that the red line and the blue line are roughly coincident, which directs that the performance of model is good and it can predict most house price correctly.

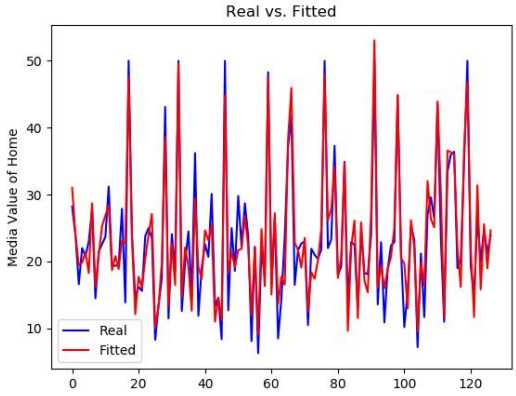

Figure5- 3. Real Value Curve vs. Fitted Curve

5.3 Ensemble methods.

5.3.1 random forest.

Through **GridSearchCV** we choose the combination of parameter like this:

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

And the Mean square error and R square are:

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

What's more, we plot the true price vs simulate price:

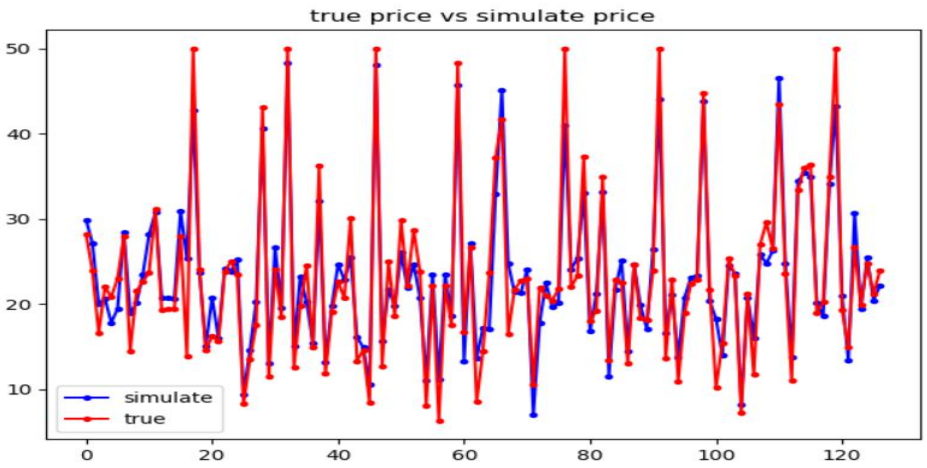

Figure5- 4. Real Value Curve vs. Fitted Curve

5.3.2 gradient boosting

Through **GridSearchCV** we choose the combination of parameter like this:

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

And the Mean square error and R square are:

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

What's more, we plot the true price vs simulate price:

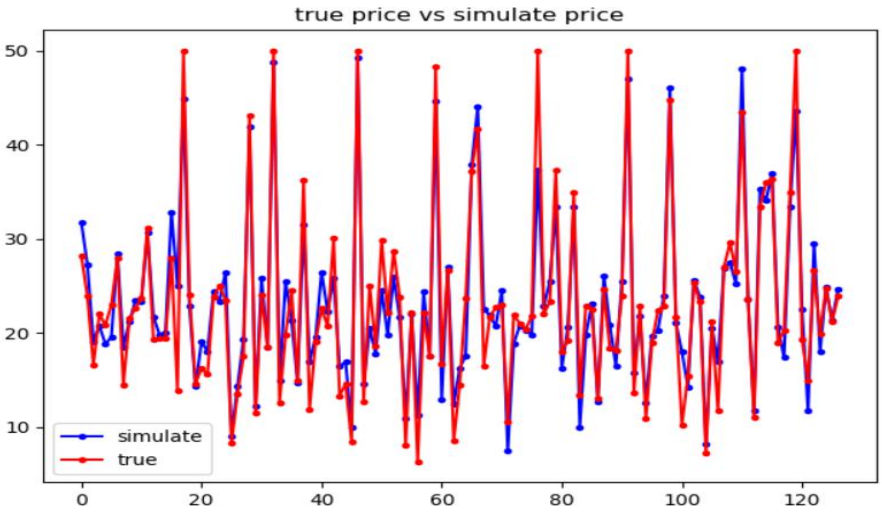

Figure5- 5. Real Value Curve vs. Fitted Curve

6. Conclusion

In this project, we mainly use linear regression as our benchmark and the neural network we learned from this class. We also combine the practical problem to use the ensemble methods, which have a better performance than linear regression and neural network.

Since our evaluation method is MSE and R square, so here's a summary of the model performance:

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

It turns out that if the data size is small, ensemble methods is truly better than other machine learning methods since it's unique resampling technique to strength the data.

Actually, in this data, if we are going to improve the current R square, we should obtain more data. Although neural network's performance is relatively bad, it's still a powerful tools. In our opinion, a sufficient data can certainly make NN better than ensemble methods.

Reference

[1] “A Beginner's Guide to Neural Networks and Deep Learning.” Pathmind, pathmind.com/wiki/neural-network#:~:text=Neural networks are a set,labeling or clustering raw input.

[2] “API Reference¶.” Scikit, scikit-learn.org/stable/modules/classes.html.

[3] Cantaro, M. (2020, January 19). Things You Didn't Know About the Boston Housing Dataset. Retrieved from

https://towardsdatascience.com/things-you-didnt-know-about-the-boston-housing-dataset-2e87a6f960e8

[4] “Code Faster with Line-of-Code Completions, Cloudless Processing.” Kite, kite.com/python/answers/how-to-calculate-r-squared-with-numpy-in-python.

[5] Hagan, Martin T., et al. Neural Network Design. s. n., 2016.

[6] Larry Hardesty | MIT News Office. “Explained: Neural Networks.” MIT News, 14 Apr. 2017, news.mit.edu/2017/explained-neural-networks-deep-learning-0414.

[7] Naya, Gabriel. “Metrics and Python.” Medium, Towards Data Science, 26 Nov. 2019, towardsdatascience.com/metrics-and-python-850b60710e0c.

[8] Samratp. (2018, December 07). Boston Housing Prices - Evaluation & Validation. Retrieved from

https://www.kaggle.com/samratp/boston-housing-prices-evaluation-validation

[9] scikit-learn Machine learning in Python. (n.d.). Retrieved from https://scikit-learn.org/stable/

[10] Martin T.H., Howard B.D., Mark H.B., Orlando D.J.(n.d.). Neuron Network Design.

[11] Sebastian R., Vahid M. (2017). Python Machine Learning Second Edition. Packt Publishing Ltd.