Housing Price Regression and Prediction Based on Gradient Boosting CART Decision Tree

Abstract

Housing price prediction is a common subject in regression forecast. Unfair housing prices occur commonly because different people buy houses with the same value at different prices. To reduce the probability of this issue, we designed a housing price prediction programme in Python based on an optimized classification and regression decision tree (CART). This report introduces the application of classification, regression, pruning strategy and gradient boosting in CART, and demonstrate the data pre-processing and parameters adjusting in gradient boosting decision tree (GBDT). To verify the efficiency of the trained GBDT, we compared the mean squared errors of the prediction results of classical CART tree and the CART tree optimized by applying gradient boosting. The implementation results show that the GBDT applying appropriate parameters achieves the better results.

I. Introduction

The housing price issue concerns the national economy and people's livelihood, and has a significant impact on national economic development and social stability. Housing prices have always been the focus of governments.

In recent years, with the economic development, the real estate industry has developed rapidly and become a pillar industry to stimulate domestic demand and promote economic growth. Taking China's national conditions as the background, with the rapid rise of real estate prices, China's real estate market bubbles are getting bigger and bigger, real estate developers and local governments get huge benefits from it, but ordinary citizens often face difficulties in buying houses, expensive houses, and uncertain housing prices. How to restrain and regulate the housing price trend, prevent the housing bubble, and stabilize the real estate market has become a common problem faced by many countries.

As an object with both commodity and investment attributes, real estate involves many resource problems and interest relations, especially in the aspects of scarce

resources and optimal allocation. Therefore, housing price forecast is particularly important. In order to maintain the sustainable and healthy development of the real estate market, we need to predict the basic development direction of the real estate. In addition, with the diversified development of social economy and the continuous improvement of social market structure system, the application field of housing price prediction will gradually expand.

There are a variety of factors that influence house prices, and the ways and degrees of these factors also vary. Some factors can be measured by mathematical models, while others can only be judged quantitatively by experience. So, forecasting house prices is somewhat difficult. The traditional prediction of commodity house price is divided into two categories. One is qualitative prediction method, such as judgment prediction method and evaluation method. The other is quantitative prediction method, such as regression analysis, state transfer method, grey prediction model, time series analysis, etc., can be used to predict the trend and development of housing prices [1].

Traditional house price analysis methods are already very powerful, but with the epidemic of novel Coronavirus, the unstable factors in the real estate industry are increasing. Therefore, more robust house price prediction methods are needed to guide the development of the real estate industry, provide data for government decision-making and provide direction for people to purchase houses. In this context, we choose housing price forecast as our research topic.

The remainder of this report is organized as follows. Section II describes the methodologies we used for housing price prediction. Section III describes the experimental results. Section IV concludes this report.

II. Methodology

2.1 CART Decision Tree

Classification and regression trees (CART) [2] are a kind of decision tree that can achieve both regression and classification. Unlike the ID3 and C4.5 algorithms, CART only supports binary trees.

The inputs and outputs of CARTs are listed in Chart 1.

CART Type	Input	Output
Classification Tree	Discrete data	Class of each sample
Regression Tree	Discrete data	Prediction (from a continuous range)

Chart 1. Inputs and outputs of CARTs

In the housing price prediction programme designed by our team, the following background is covered:

2.1.1 Classification Process of CART

The attribute selection index of CART classification tree is GINI. GINI reflects the uncertainty of the samples. The smaller GINI, the less differences between the samples. The process of classification is a process of uncertainty reduction, namely, data purification. Therefore, in the implementation of CART classification tree, the attribute with the least GINI value will finally be the classification attribute.

For node t in a tree, the value of $GINI(t)$ is calculated by the following equation:

$$GINI(t) = 1 - \sum [p(C_k|t)]^2$$

where C_k is the k^{th} class of an attribute, $p(C_k|t)$ is the conditional probability that the samples in node t.

Take the case in Figure 1 as an example to explain the process of GINI calculation. Figure 1 shows the result of a survey. The original dataset is stored in *Node D*, and were split equally into *Node D1* and *Node D2*. The attribute of the samples is whether they agree with the survey topic or not. The classes in this case are *Agree* and *Disagree*.

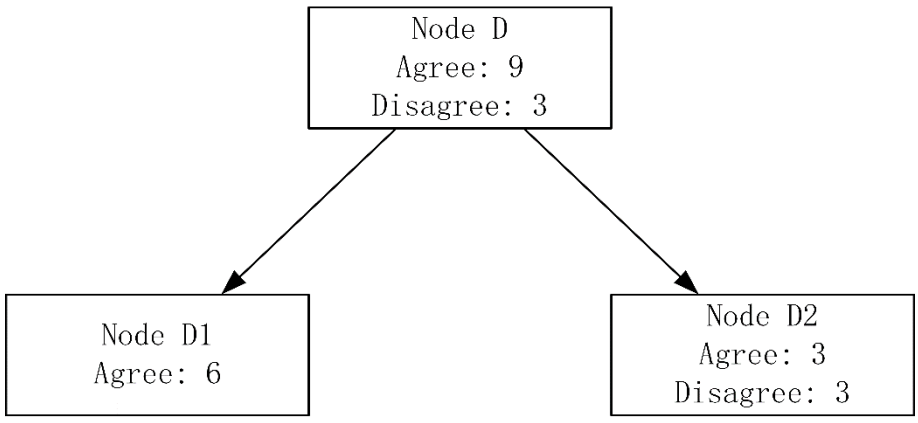

Figure 1. An example of classification tree

Suppose that C_1 refers to *Agree* class and C_2 refers to *Disagree* class, and the

attribute they belong to is *attribute A*. For *Node D1* and *Node D2*, their GINI values toward *attribute A* can be calculated as below:

1) $GINI(D1) = 1 - [p(C_1|D1)^2 + p(C_2|D2)^2] = 1 - [1^2 + 0] = 0$
2) $GINI(D2) = 1 - [p(C_1|D1)^2 + p(C_2|D2)^2] = 1 - [0.5^2 + 0.5^2] = 0.5$

For *Node D*, its GINI value towards the attribute A is the normalized sum of $GINI(D1)$ and $GINI(D2)$, which can be written as

$$GINI(D,A) = \frac{D1}{D}GINI(D1) + \frac{D2}{D}GINI(D2) = \frac{6}{12}GINI(D1) + \frac{6}{12}GINI(D2)$$
$$= 0.5 * 0 + 0.5 * 0.5 = 0.25$$

Thus, for *Node D*, the GINI value of *attribute A* is 0.25. The selection of classification attribute requires calculation of GINI value of every attribute so that the least GINI can be found.

2.1.2 Regression Process of CART

The difference between CART regression tree and CART classification tree is that classification tree adopts the class with higher probability of occurrence in leaf nodes for classification, while regression tree adopts the mean value in leaf nodes as the prediction index. To establish the CART regression tree, we use the following loss function:

$$Loss\big(y, f(x)\big) = (f(x) - y)^2$$

where *x* refers to the eigenvalues, *y* refers to the target values in the training set, *f(x)* refers to the mean value of target values after data division. Take the case in Chart 2 as an example of calculation explanation.

x	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
y	1.12	1.54	1.87	2.05	2.28	2.56	2.84	3.02

Chart 2. An example of regression tree dataset

Supposed that the maximum depth of the regression decision tree is 2, then it will be generated by the following steps:

Step 1. Sort the eigenvalues and take the average of two adjacent eigenvalues, and consider them as the possible dividing points (Chart 3).

s	1.5	2.5	3.5	4.5	5.5	6.5	7.5

Step 2. Calculate the squared loss of each possible dividing point by the following equation:

$$L(s) = \sum (y_i - R_1)^2 + \sum \left(y_j - R_2\right)^2$$

where $i \in (1, s) \cap Z$, $j \in (s, total\ number\ of\ y) \cap Z$. For R_1 and R_2 we have

$$R_1 = \frac{\sum y_i}{[s]},\ i \in (1, s) \cap Z$$

$$R_2 = \frac{\sum y_j}{total\ number\ of\ y - [s]},\ j \in (s, total\ number\ of\ y) \cap Z$$

For example, for division point $s_1 = 1.5$, we have

$$R_1 = \frac{1.12}{[1.5]} = \frac{1.12}{1} = 1.12$$

$$R_2 = \frac{1.54 + 1.87 + 2.05 + 2.28 + 2.56 + 2.84 + 3.02}{8 - [1.5]} = \frac{16.16}{8 - 1} = \frac{16.16}{7} = 2.31$$

$$\therefore\ L(s_1) = (1.12 - 1.12)^2 + [(1.54 - 2.31)^2 + (1.87 - 2.31)^2 + (2.05 - 2.31)^2 + (2.28 - 2.31)^2 + (2.56 - 2.31)^2 + (2.84 - 2.31)^2 + (3.02 - 2.31)^2] = 1.70$$

The loss function results of all points are shown in Chart 3.

x	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
y	1.12	1.54	1.87	2.05	2.28	2.56	2.84	3.02
s	**1.5**	**2.5**	**3.5**	**4.5**	**5.5**	**6.5**	**7.5**	
$\boldsymbol{L(s)}$	1.70	1.10	0.91	0.82	0.93	1.36	2.09	

Chart 3. Loss function results

Therefore, the division point of Step 2 is $s_4 = 4.5$. Two sub-trees are generated based on this division point. Then we have

x	**1**	**2**	**3**	**4**
Sub-tree 1	1.12	1.54	1.87	2.05
x	**5**	**6**	**7**	**8**
Sub-tree 2	2.28	2.56	2.84	3.02

Chart 4. Sub-trees

Step 3. & 4. Generate the sub-trees according to the above rules, which is shown in Figure 2. The target data and predictions are listed in Chart 5.

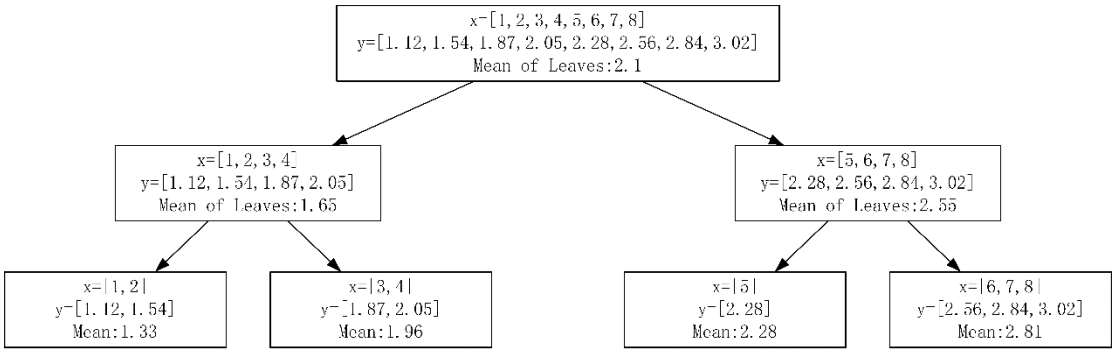

Figure 2. Regression decision tree (maximum depth = 2)

x	1	2	3	4	5	6	7	8
Target	1.12	1.54	1.87	2.05	2.28	2.56	2.84	3.02
Prediction	1.33	1.33	1.96	1.96	2.28	2.81	2.81	2.81

Chart 5. Target data and predictions

2.1.3 Cost-Complexity Pruning

To prevent over-fitting of the training sets, the CART algorithm applies cost-complexity pruning (CCP) [3] strategy. Cost refers to the number of misclassification samples increased when factor tree T_t replaced by leaf nodes during pruning. Complexity represents the number of leaf nodes reduced by T_t of subtree after pruning. The CCP core algorithm can be concluded by the following equation:

$$\alpha = \frac{R(t) - R(T_t)}{|N| - 1}$$

where t is a node and T_t is a subtree of node t. The root of T_t is node t.

α is the parameter that evaluate the association between cost and complexity, $R(t)$ refers to cost of node t, and $|N|$ is the number of leaves in subtree T_t.

$R(t)$ can be calculated by $R(t) = \sum_{i=1}^{m} r_i(t) * p_i(t)$, where $r_i(t)$ is the error rate of node i, and $p_i(t)$ is the proportion of samples in node i to total samples in the tree.

Figure 3 shows a sample tree for CCP practice.

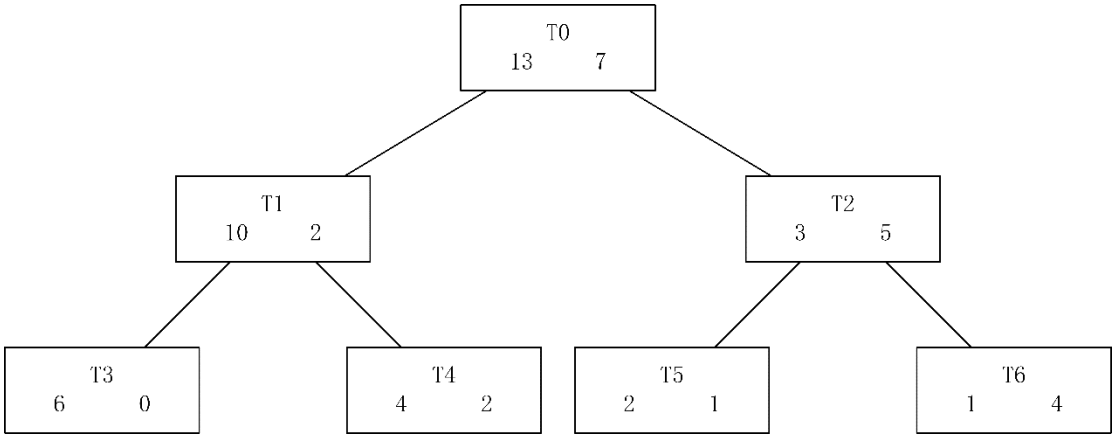

Figure 3. CCP sample

Step 1. Calculate α value of all non-leaf nodes from the button to the top. In this case, suppose that there are 100 samples in total. Then we have

$$R(t) = \frac{7}{20} * \frac{20}{100} = \frac{7}{100}$$

If the subtree of T_1 is pruned, we have

$$R(T_1 t) = R_5(t) + R_6(t) = \frac{1}{3} * \frac{3}{100} + \frac{4}{5} * \frac{5}{100} = \frac{5}{100}$$

$$\therefore \alpha_1 = \frac{\frac{7}{100} - \frac{5}{100}}{3-1} = \frac{\frac{2}{100}}{2} = 0.01$$

If the subtree of T_2 is pruned, then we have

$$R(T_2 t) = R_3(t) + R_4(t) = 0 + \frac{2}{6} * \frac{6}{100} = \frac{2}{100}$$

$$\therefore \alpha_2 = \frac{\frac{7}{100} - \frac{2}{100}}{3-1} = \frac{\frac{5}{100}}{2} = 0.025$$

Step 2. Conduct the different pruning plan. From *Step 1*, we generate a series of pruned tree and store them in a pruning set $\{T_0, T_1, T_2\}$ (T_0 refers to the original tree, or also called complete decision tree). The possible results of pruning are shown in Figure 4 and 5.

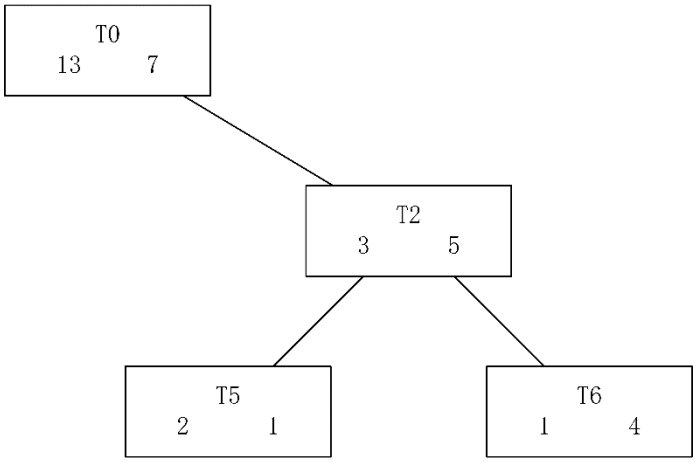

Figure 4. Pruning plan 1

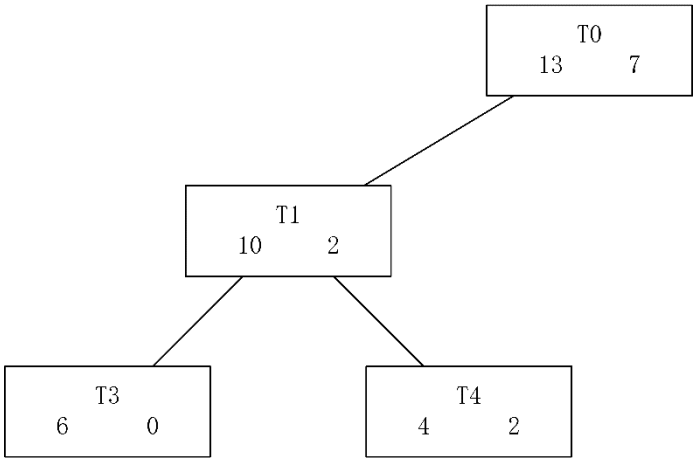

Figure 5. Pruning plan 2

According to the results in Step 1, $\alpha_1 < \alpha_2$, thus, the subtree of T_1 will be pruned. Therefore, the final decision tree in this case will be the one shown in Figure 4.

Step 3. Verification of the pruning plan. The best pruned tree T_{best} is a pruned tree with least nodes and satisfying the equation:

$$E_i \leq E' + SE(E')$$

where E_i is the error rate of pruned tree T_i, E' is the minimum of E_i. $SE(E')$ can be calculated by

$$SE(E') = \sqrt{\frac{E' * (N - E')}{N}}$$

where N refers to the size of pruning set. In this case, $E_1 = \frac{17}{36} = 0.472$, $E_2 = \frac{11}{44} = 0.25$. Therefore, $E' = 0.25$, $SE(E') = \sqrt{\frac{0.25*(3-0.25)}{3}} = 0.479$, $E' + SE(E') = 0.729$.

The final pruning plan satisfies the conditions, so it is verified.

2.2 Boosting Decision Tree

Boosting is a general method of producing an accurate classifier on the basis of weak and unstable classifiers [4]. The great advantage of this method is that boosting often does not suffer from overfitting. Boosting decision tree (BDT) is an integrated learning method based on CART decision tree. In this method, an effect of a strong learner (classifier) is integrated by iterating the output decision trees of several weak learners (Figure 6).

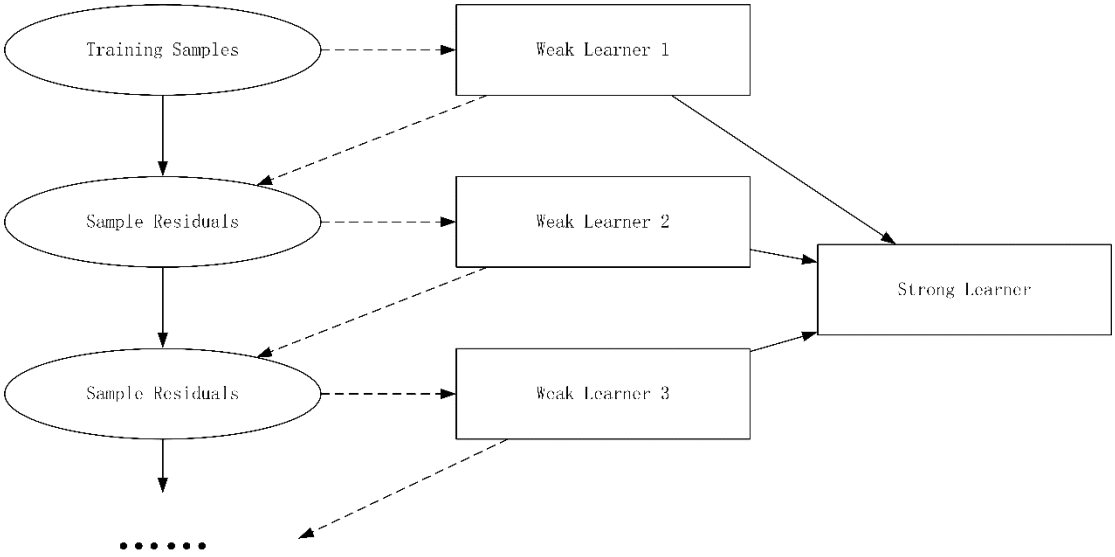

Figure 6. BDT model

A learner with a classification accuracy of 60%-80% is called a weak learner. When the classification accuracy of a learner is more than 90%, it can be considered as a strong learner. Generally, weak learners perform slightly better than random predictions, but not with much accuracy.

The algorithm of BDT is demonstrated in Chart 6.

ALGORITHM: Boosting Decision Tree

Input: Training dataset $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$

Output: Corresponding BDT

1 Initialize: $f_0(x) = 0$

2 Set domain of $m = 1, 2, \dots, M$

3 For each sample (x_i, y_i), calculate residual error

$$r_{m,i} = y_i - f_{m-1}(x_i), i = 1,2, \dots, N$$

4 Train a regression decision tree according to residual set $\left(x_i, r_{m,i}\right)_{i=1,2,\dots,N}$, then we have $T(x; \Theta_m)$

5 Update $f_m(x) = f_{m-1}(x) + T(x; \Theta_m)$

6 Repeat the iteration stage in step 4&5 and get the final BDT

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m)$$

Chart 6. Algorithm of BDT

In the expression of final BDT, $T\left(x; \Theta_m\right)$ refers to the decision tree, Θ_m is the parameters of the decision tree, and M is the number of intermediate decision trees.

As stated earlier, the loss evaluation of BDT is squared loss function. The formula below is used to interpret the squared loss function in BDT scenario:

$$L\big(y, f_{m-1}(x) + T(x, \Theta_m)\big) = \big(y - f_{m-1}(x) - T(x, \Theta_m)\big)^2$$

The parameters of the next tree are determined by empirical risk minimization. In other words, make it as small as possible to find the optimal partition point. The mathematical expression of this process is:

$$\widehat{\Theta_m} = arg\ min_{\Theta_m} \sum_{i=1}^{N} L\big(y, f_{m-1}(x) + T(x, \Theta_m)\big)$$

2.3 Gradient Boosting Decision Tree

A common optimization method for CART is to apply gradient boosting. Compared with traditional Boosting, gradient boosting (GB) gradually approximates the local minimum by iteratively selecting the basis function in the direction of the negative gradient of a target loss function. GBDT differs from BDT in that residuals are replaced by gradients, and each base learner has a corresponding parameter weight. Friedman proposed an approximation method using the fastest descent, using the negative gradient of the loss function to fit the base learner [5] which can be described as

$$-\left[\frac{\partial L\left(y_i, F(x_i)\right)}{\partial F(x_i)}\right]_{F(x)=F_{t-1}(x)}$$

where $L\left(y_i, F(x_i)\right) = \frac{1}{2}\left(y_i - F(x_i)\right)^2$. The value of gradient can be gotten by taking the derivative that

$$\frac{\partial L\left(y_i, F(x_i)\right)}{\partial F(x_i)} = F(x_i) - y_i$$

The residual is the negative of the gradient, which is

$$r_{ti} = y_i - F_{t-1}(x) = -\left[\frac{\partial L\left(y_i, F(x_i)\right)}{\partial F(x_i)}\right]_{F(x)=F_{t-1}(x)}$$

Each tree of gradient boosting decision tree (GBDT) algorithm learns the residual of the sum of all the previous trees. This residual is the sum of the predicted value to get the actual value.

For example, suppose that the actual age of *person A* is 18. Assume that the prediction of the first tree is 12, then we know $residual_1$ is 6. In the initialize stage of the second tree, the age of *person A* is set as 6. Assume that the prediction of the second tree is 5, then we have $residual_2$ is 1. Thus, the age of *person A* will be initialized as 1 in the third tree. Suppose that the prediction with 0 residual is 2c, then the final prediction of the age of *person A* is $18 - 2 = 16$. The algorithm of gradient boosting can be demonstrated as Chart 7.

ALGORITHM: Gradient Boosting

Input: Training dataset $Data = (x_1, y_1), (x_2, y_2), \dots, (x_n, y_n), y_i \in \{+1, -1\}$

Output: Gradient and weight

1 Initialize: $F_0(x) = arg\ min_{h_0} \sum_{i=1}^{N} L\left(y_i, h_0(x)\right)$

2 Growth of augmented matrix:

for t=1 to T do

$$\widetilde{y_i} = -\left[\frac{\partial L\left(y_i, F(x_i)\right)}{\partial F(x_i)}\right]_{F(x)=F_{t-1}(x)}, i = 1,2, \dots\ N$$

3 Calculate the negative gradient:

$$\omega_t = arg\,min_{\omega_t} \sum_{i=1}^{N} \left(\widetilde{y_i} - h_t(x, w_t)\right)^2$$

4 Fit residuals to obtain the base learner:

$$\alpha_t = arg\,min_{\alpha_t} \sum_{i=1}^{N} L\left(y_i, f_{t-1}(x_i) + \alpha_t h_t(x; w_t)\right)$$

5 Obtain the weight of the base learner:

$$F_t(x) = F_{t-1}(x_i) + \alpha_t h_t(x; w_t)$$

6 Update the gradient and weight

Chart 7. Gradient boosting algorithm

Figure 7 shows the dataflow structure of gradient boosting algorithm.

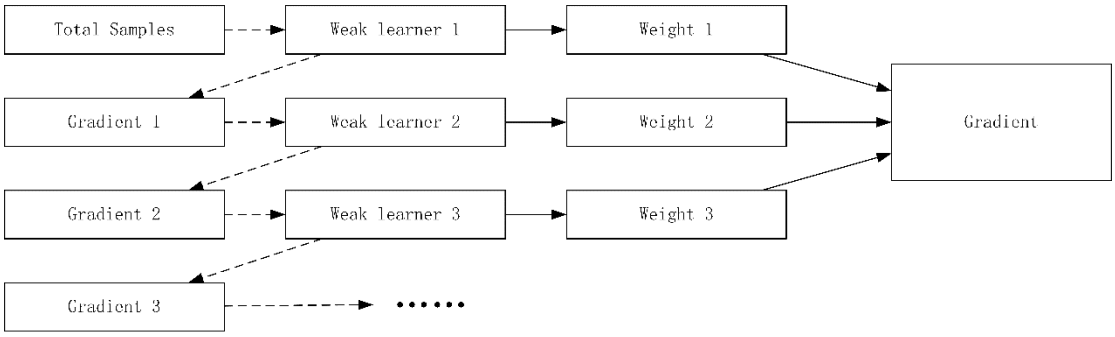

Figure 7. Gradient boosting dataflow

In the process of GBDT, CART regression tree divides the space into *K* domains that are non-intersect, and determines the output c_k of each domain. The mathematical expression of the outputs is as follows:

$$F(x) = \sum_{k=1}^{K} c_k I\,(x \in R_k)$$

The process of GBDT regression is demonstrated in Chart 8.

ALGORITHM: Gradient Boosting Decision Tree

Input: Training dataset $D = (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n), y_i \in \{+1, -1\}$

Output: Corresponding GBDT

1 Initialize: $f_0(x) = arg\,min_{h_0} \sum_{i=1}^{N} L\left(y_i, h_0(x)\right) = arg\,min_c \sum_{i=1}^{N} L(y_i, c)$

2 Calculate the negative gradient by augmented matrix:

for t=1 to T do

$$\widetilde{y_i} = -\left[\frac{\partial L\left(y_i, F(x_i)\right)}{\partial F(x_i)}\right]_{F(x)=F_{t-1}(x)}, i = 1,2, \dots N$$

3 Fitting residuals to obtain the regression tree. Leaf domain of the t^{th} tree is:

$$h_t(x) = \sum_{k=1}^{K} c_k I\ (x \in R_{tk})$$

4 Update the total domain of GBDT:

$$F_t(x) = F_{t-1}(x_i) + h_t(x) = F_{t-1}(x) + \sum_{k=1}^{K} c_k I\ (x \in R_{tk})$$

5 Obtain the additive model of GBDT:

$$F(x) = \sum_{k=1}^{K} c_k I\ (x \in R_k)$$

Chart 8. GBDT regression process

2.4 Datasets

In this work, we use the `fetch_California_housing` dataset [6] [7] in scikit-learn library [8]. This dataset contains information about randomly selected used and new construction and agricultural equipment sold in the state of California in the United States (sales prices are in US dollars). It is a medium dataset with 20640 cases. This sample dataset contains 10 attributes, the first 8 and 10th of which are used as feature inputs to predict California housing prices, and the 9th attribute is used as a label to be predicted, shown in Table 1.

Feature	Description
longitude	A measure of how far west a house is; a higher value is farther west.
latitude	A measure of how far north a house is; a higher value is farther north.
housingMedianAge	Median age of a house within a block; a lower number is a newer building.
totalRooms	Total number of rooms within a block.
totalBedrooms	Total number of bedrooms within a block.

population	Total number of people residing within a block.
households	Total number of households, a group of people residing within a home unit, for a block.
medianIncome	Median income for households within a block of houses (measured in tens of thousands of US Dollars).
medianHouseValue	Median house value for households within a block (measured in US Dollars)
oceanProximity	Location of the house w.r.t ocean/sea.

Table 1. The features description of fetch_California_housing dataset

2.6 Data Pre-processing

According to dataset analysis shown in Figure 8, the first nine attributes of this data set are continuous and the last attribute is discrete. To allow the discrete type data participate in regression, a split and then combined data processing needs to be applied. Meanwhile, there are 207 missing values under the *total_bedroom* attribute, and we filled in these blanks with a median substitution strategy. For data normalization, we applied min-max scalar strategy with its default settings from scikit-learn library.

```
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
```

Figure 8. Attributes information

The total of samples is 20640, and data consists of 9 feature attributes. First, the

sample dataset is divided into 2 parts: feature X and label Y. Then the dataset is split up into testing set and training set according to the ratio of 1:5.

III. Verification

3.1 Regression Tree Parameter Tuning

A series of trials were implemented to determine the best fitting and pruning parameters shown in Table 2 [9].

Parameter	Description
learning_rate	Learning rate shrinks the contribution of each tree by *learning_rate*. There is a trade-off between *learning_rate* and *n_estimators*. Values must be in the range *(0.0, inf)*.
n_estimators	The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance. Values must be in the range *[1, inf)*.
subsample	The fraction of samples to be used for fitting the individual base learners. If smaller than 1.0 this results in Stochastic Gradient Boosting. *subsample* interacts with the parameter *n_estimators*. Choosing *subsample* < *1.0* leads to a reduction of variance and an increase in bias. Values must be in the range *(0.0, 1.0]*.
min_samples_split	The minimum number of samples required to split an internal node.
min_samples_leaf	The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least *min_samples_leaf* training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.
max_depth	Maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. Tune this parameter for best performance; the best value depends on the interaction of the input variables. Values must be in the range *[1, inf)*.

Table 2. Fitting and pruning parameters

Three indicators are used as the aid of parameters tuning: coefficient of determination r^2 in fitting process, mean squared error of fitting prediction, and time complexity. r^2 refers to that number% of the dependent variables derived from the regression equation can be explained by independent variables. Thus, it is used to judge the degree of data fitting. r^2 can be calculated as

$$r^2 = \frac{\sum(\hat{y_i} - \overline{y})^2}{\sum(\hat{y_i} - \overline{y})^2 + \sum(y_i - \hat{y_i})^2} = \frac{\sum(\hat{y_i} - \overline{y})^2}{\sum(y_i - \overline{y})^2}$$

while the mean squared error (MSE) can be calculated as

$$MSE = \frac{\sum(y_i - \hat{y_i})^2}{n}$$

The priority of the parameter tuning is: *max_depth* > *min_samples_leaf* > *min_samples_split* > *subsample* > *learning_rate* = *n_estimators*. During the parameter adjusting process, the above parameters are pre-set to default values and gradually replaced by more suitable values.

The first stage is max_depth adjusting. The followings are the testing data:

max_depth	**3(default)**	**5**	**7**	**9**	**11**	**13**	**15**
r^2	0.86	0.84	0.91	0.96	0.99	1.00	1.00
MSE	43669	45535	35135	23678	12177	4598	1184
Time (s)	1.79	1.29	1.39	1.98	2.18	2.57	2.97

The final *max_depth* value is set to 13 because 13 is the minimum depth taken when the training set is fully fitted:

The second stage is to find the best *min_samples_leaf*. The followings are the testing data:

min_samples_leaf	**1(default)**	**2**	**3**	**4**
r^2	1.00	1.00	0.99	0.99
MSE	4543	6390	9267	12646
Time (s)	2.48	2.45	2.38	2.57

The most appropriate value of *min_samples_leaf* is 1. Increasing this value will increase the error or degree of fit.

The third stage is to find the best *min_samples_split*. The followings are the testing data:

min_samples_split	2(default)	3	4	5
r^2	1.00	1.00	1.00	1.00
MSE	4643	5410	5631	6214
Time (s)	2.48	2.57	2.77	2.67

The most appropriate *min_samples_split* is 2. Increasing this value will increase error.

The fourth stage is to find the best *subsample*. The followings are the test data:

subsample	1(default)	0.9	0.8	0.75
r^2	1.00	1.00	1.00	1.00
MSE	4543	5062	5613	6510
Time (s)	2.48	2.38	2.28	1.98

The final *subsample* value is set to 0.8 because in this case the increase in error is small, but the operation time is reduced.

The last stage is to adjust *learning_rate* and *n_estimators*. We tested the following combinations:

learning_rate	0.1	0.1	0.25	0.25	0.5	0.5	0.75	0.75	1.0	1.0
n_estimators	100	300	100	300	100	300	100	300	100	300
r^2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
MSE-train	6273	431	1075	0.80	100	0.00	13	0.00	18	0.00
MSE-test (k)	48	48	52	52	60	86	74	70	94	90

According to the test data, four of these combinations are over-fitting. We finally choose the combination (0.1, 300) because there was no over-fitting or under-fitting in this case.

3.2 Verification

To verify the efficiency improvement of CART decision tree, we tried 5 times to avoid extreme situations. The outputs of each trial are shown below.

3.2.1 Trial 1

CART Training Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-117.03	32.71	...	103000.0	145631.404959
1	-118.16	33.77	...	382100.0	311790.0
2	-120.48	34.66	...	172600.0	241434.615385
3	-117.11	32.69	...	93400.0	123873.972603
4	-119.8	36.78	...	96500.0	86660.0
...	...	...	...	...	...
16507	-117.96	33.78	...	229200.0	260708.333333
16508	-117.43	34.02	...	97800.0	111543.58209
16509	-118.38	34.03	...	222100.0	191482.608696
16510	-121.96	37.58	...	283500.0	241011.404145
16511	-122.42	37.77	...	325000.0	325000.0

CART Testing Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-119.01	36.06	...	47700.0	49571.428571
1	-119.46	35.14	...	45800.0	73938.461538
2	-122.44	37.8	...	500001.0	451800.2
3	-118.72	34.28	...	218600.0	236240.666667
4	-121.93	36.62	...	278000.0	255131.429319
...	...	...	...	...	...
4123	-117.22	33.36	...	263300.0	294600.0
4124	-120.83	35.36	...	266800.0	255131.429319
4125	-122.05	37.31	...	500001.0	425687.26087
4126	-119.76	36.77	...	72300.0	45300.0
4127	-118.37	34.22	...	151500.0	212631.756757

GBDT Training Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-117.03	32.71	...	103000.0	102914.046957
1	-118.16	33.77	...	382100.0	381943.019468
2	-120.48	34.66	...	172600.0	172365.709338
3	-117.11	32.69	...	93400.0	93231.065407
4	-119.8	36.78	...	96500.0	95958.686731
...	...	...	...	...	...
16507	-117.96	33.78	...	229200.0	229765.880416
16508	-117.43	34.02	...	97800.0	97700.882746
16509	-118.38	34.03	...	222100.0	220758.262542
16510	-121.96	37.58	...	283500.0	283002.774619
16511	-122.42	37.77	...	325000.0	324763.537513

GBDT Testing Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-119.01	36.06	...	47700.0	49403.219214
1	-119.46	35.14	...	45800.0	71976.593065
2	-122.44	37.8	...	500001.0	446326.280377
3	-118.72	34.28	...	218600.0	240280.927775
4	-121.93	36.62	...	278000.0	282928.197141
...	...	...	...	...	...
4123	-117.22	33.36	...	263300.0	248498.996885
4124	-120.83	35.36	...	266800.0	238061.291313
4125	-122.05	37.31	...	500001.0	487269.298538
4126	-119.76	36.77	...	72300.0	74661.638186
4127	-118.37	34.22	...	151500.0	167518.104066

```
CART Regression Evaluation:
Fitting score of training set=0.86
Fitting score of testingset=0.62
Mean squared error of training=43858.77
Mean squared error of testing=70317.63
GBDT Regression Evaluation:
Fitting score of training set=1.00
Fitting score of testing set=0.82
Mean squared error of training=460.27
Mean squared error of testing=48603.86
```

3.2.2 Trial 2

CART Training Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-117.03	32.71	...	103000.0	127096.296296
1	-118.16	33.77	...	382100.0	252432.525
2	-120.48	34.66	...	172600.0	135792.682927
3	-117.11	32.69	...	93400.0	91333.333333
4	-119.8	36.78	...	96500.0	78920.289855
...	...	...	...	...	...
16507	-117.96	33.78	...	229200.0	262550.0
16508	-117.43	34.02	...	97800.0	126162.5
16509	-118.38	34.03	...	222100.0	288116.0
16510	-121.96	37.58	...	283500.0	281925.0
16511	-122.42	37.77	...	325000.0	296696.464286

CART Testing Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-119.01	36.06	...	47700.0	57164.197531
1	-119.46	35.14	...	45800.0	70041.666667
2	-122.44	37.8	...	500001.0	472935.6
3	-118.72	34.28	...	218600.0	281925.0
4	-121.93	36.62	...	278000.0	267185.714286
...	...	...	...	...	...
4123	-117.22	33.36	...	263300.0	160980.0
4124	-120.83	35.36	...	266800.0	241726.433962
4125	-122.05	37.31	...	500001.0	500001.0
4126	-119.76	36.77	...	72300.0	79533.333333
4127	-118.37	34.22	...	151500.0	197012.815625

GBDT Training Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-117.03	32.71	...	103000.0	103206.918744
1	-118.16	33.77	...	382100.0	381971.303571
2	-120.48	34.66	...	172600.0	172358.624249
3	-117.11	32.69	...	93400.0	92910.387253
4	-119.8	36.78	...	96500.0	95620.248784
...	...	...	...	...	...
16507	-117.96	33.78	...	229200.0	229158.148008
16508	-117.43	34.02	...	97800.0	97639.27655
16509	-118.38	34.03	...	222100.0	221962.189502
16510	-121.96	37.58	...	283500.0	282835.248535
16511	-122.42	37.77	...	325000.0	324847.205049

GBDT Testing Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-119.01	36.06	...	47700.0	51328.244382
1	-119.46	35.14	...	45800.0	74229.434545
2	-122.44	37.8	...	500001.0	444909.45686
3	-118.72	34.28	...	218600.0	240242.182704
4	-121.93	36.62	...	278000.0	248786.62968
...	...	...	...	...	...
4123	-117.22	33.36	...	263300.0	231448.030432
4124	-120.83	35.36	...	266800.0	237574.68685
4125	-122.05	37.31	...	500001.0	502037.319491
4126	-119.76	36.77	...	72300.0	75077.22527
4127	-118.37	34.22	...	151500.0	162293.432152

```
CART Regression Evaluation:
Fitting score of training set=0.84
Fitting score of testingset=0.65
Mean squared error of training=45973.86
Mean squared error of testing=67380.80
GBDT Regression Evaluation:
Fitting score of training set=1.00
Fitting score of testing set=0.82
Mean squared error of training=478.55
Mean squared error of testing=48996.43
```

3.2.3 Trial 3

CART Training Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-117.03	32.71	...	103000.0	132016.949153
1	-118.16	33.77	...	382100.0	271200.0
2	-120.48	34.66	...	172600.0	245424.390244
3	-117.11	32.69	...	93400.0	90425.0
4	-119.8	36.78	...	96500.0	107396.721311
...	...	...	...	...	...
16507	-117.96	33.78	...	229200.0	228340.0
16508	-117.43	34.02	...	97800.0	112455.454545
16509	-118.38	34.03	...	222100.0	144963.043478
16510	-121.96	37.58	...	283500.0	253620.454545
16511	-122.42	37.77	...	325000.0	325000.0

CART Testing Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-119.01	36.06	...	47700.0	55816.831683
1	-119.46	35.14	...	45800.0	73940.0
2	-122.44	37.8	...	500001.0	491409.041667
3	-118.72	34.28	...	218600.0	264136.440678
4	-121.93	36.62	...	278000.0	279017.924528
...	...	...	...	...	...
4123	-117.22	33.36	...	263300.0	206869.876712
4124	-120.83	35.36	...	266800.0	232564.285714
4125	-122.05	37.31	...	500001.0	375125.0
4126	-119.76	36.77	...	72300.0	75595.683453
4127	-118.37	34.22	...	151500.0	179266.666667

GBDT Training Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-117.03	32.71	...	103000.0	102649.080813
1	-118.16	33.77	...	382100.0	382092.794992
2	-120.48	34.66	...	172600.0	172553.431163
3	-117.11	32.69	...	93400.0	93170.033314
4	-119.8	36.78	...	96500.0	96252.929229
...	...	...	...	...	...
16507	-117.96	33.78	...	229200.0	229571.052269
16508	-117.43	34.02	...	97800.0	97850.292665
16509	-118.38	34.03	...	222100.0	222926.702774
16510	-121.96	37.58	...	283500.0	282808.318829
16511	-122.42	37.77	...	325000.0	324895.53709

GBDT Testing Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-119.01	36.06	...	47700.0	54767.201878
1	-119.46	35.14	...	45800.0	65193.345872
2	-122.44	37.8	...	500001.0	459901.874479
3	-118.72	34.28	...	218600.0	245078.227296
4	-121.93	36.62	...	278000.0	263143.851086
...	...	...	...	...	...
4123	-117.22	33.36	...	263300.0	241351.796145
4124	-120.83	35.36	...	266800.0	239873.454817
4125	-122.05	37.31	...	500001.0	488783.432638
4126	-119.76	36.77	...	72300.0	72551.377915
4127	-118.37	34.22	...	151500.0	170048.119682

```
CART Regression Evaluation:
Fitting score of training set=0.85
Fitting score of testingset=0.66
Mean squared error of training=44202.43
Mean squared error of testing=67131.53
GBDT Regression Evaluation:
Fitting score of training set=1.00
Fitting score of testing set=0.82
Mean squared error of training=468.71
Mean squared error of testing=48803.51
```

3.2.4 Trial 4

CART Training Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-117.03	32.71	...	103000.0	116663.636364
1	-118.16	33.77	...	382100.0	382100.0
2	-120.48	34.66	...	172600.0	220392.307692
3	-117.11	32.69	...	93400.0	82200.0
4	-119.8	36.78	...	96500.0	94627.835052
...	...	...	...	...	...
16507	-117.96	33.78	...	229200.0	246260.0
16508	-117.43	34.02	...	97800.0	104123.684211
16509	-118.38	34.03	...	222100.0	268422.727273
16510	-121.96	37.58	...	283500.0	281300.0
16511	-122.42	37.77	...	325000.0	325000.0

CART Testing Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-119.01	36.06	...	47700.0	57267.099567
1	-119.46	35.14	...	45800.0	81005.063291
2	-122.44	37.8	...	500001.0	298221.428571
3	-118.72	34.28	...	218600.0	245750.0
4	-121.93	36.62	...	278000.0	218012.5
...	...	...	...	...	...
4123	-117.22	33.36	...	263300.0	185686.27451
4124	-120.83	35.36	...	266800.0	240842.105263
4125	-122.05	37.31	...	500001.0	500001.0
4126	-119.76	36.77	...	72300.0	81005.063291
4127	-118.37	34.22	...	151500.0	205790.697674

GBDT Training Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-117.03	32.71	...	103000.0	103134.531488
1	-118.16	33.77	...	382100.0	382004.700602
2	-120.48	34.66	...	172600.0	172416.13665
3	-117.11	32.69	...	93400.0	93281.243723
4	-119.8	36.78	...	96500.0	95337.925555
...	...	...	...	...	...
16507	-117.96	33.78	...	229200.0	229396.198727
16508	-117.43	34.02	...	97800.0	97940.099009
16509	-118.38	34.03	...	222100.0	221635.521446
16510	-121.96	37.58	...	283500.0	282670.015128
16511	-122.42	37.77	...	325000.0	324896.684406

```
CART Regression Evaluation:
Fitting score of training set=0.88
Fitting score of testingset=0.61
Mean squared error of training=40746.64
Mean squared error of testing=71633.69
GBDT Regression Evaluation:
Fitting score of training set=1.00
Fitting score of testing set=0.82
Mean squared error of training=465.88
Mean squared error of testing=48068.58
```

3.2.5 Trial 5

CART Training Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-117.03	32.71	...	103000.0	123908.064516
1	-118.16	33.77	...	382100.0	422533.444444
2	-120.48	34.66	...	172600.0	153157.142857
3	-117.11	32.69	...	93400.0	138319.59799
4	-119.8	36.78	...	96500.0	92215.2
...	...	...	...	...	...
16507	-117.96	33.78	...	229200.0	229566.666667
16508	-117.43	34.02	...	97800.0	108260.240964
16509	-118.38	34.03	...	222100.0	176289.949749
16510	-121.96	37.58	...	283500.0	287152.941176
16511	-122.42	37.77	...	325000.0	325000.0

CART Testing Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-119.01	36.06	...	47700.0	60676.993865
1	-119.46	35.14	...	45800.0	72942.352941
2	-122.44	37.8	...	500001.0	500001.0
3	-118.72	34.28	...	218600.0	284088.395349
4	-121.93	36.62	...	278000.0	251940.816327
...	...	...	...	...	...
4123	-117.22	33.36	...	263300.0	214939.784091
4124	-120.83	35.36	...	266800.0	283600.0
4125	-122.05	37.31	...	500001.0	500001.0
4126	-119.76	36.77	...	72300.0	72942.352941
4127	-118.37	34.22	...	151500.0	193000.0

GBDT Training Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-117.03	32.71	...	103000.0	103084.351882
1	-118.16	33.77	...	382100.0	381862.679029
2	-120.48	34.66	...	172600.0	172533.772096
3	-117.11	32.69	...	93400.0	93124.772294
4	-119.8	36.78	...	96500.0	95598.782736
...	...	...	...	...	...
16507	-117.96	33.78	...	229200.0	229274.227851
16508	-117.43	34.02	...	97800.0	98174.609801
16509	-118.38	34.03	...	222100.0	221095.402626
16510	-121.96	37.58	...	283500.0	282773.514313
16511	-122.42	37.77	...	325000.0	325051.700709

GBDT Testing Prediction

	longitude	latitude	...	actual_median_house_value	predicted_median_house_value
0	-119.01	36.06	...	47700.0	47774.332875
1	-119.46	35.14	...	45800.0	65556.355828
2	-122.44	37.8	...	500001.0	473520.841483
3	-118.72	34.28	...	218600.0	256624.705358
4	-121.93	36.62	...	278000.0	261889.653655
...	...	...	...	...	...
4123	-117.22	33.36	...	263300.0	241333.014586
4124	-120.83	35.36	...	266800.0	254773.366766
4125	-122.05	37.31	...	500001.0	502837.751246
4126	-119.76	36.77	...	72300.0	68479.421354
4127	-118.37	34.22	...	151500.0	173959.616485

```
CART Regression Evaluation:
Fitting score of training set=0.87
Fitting score of testingset=0.64
Mean squared error of training=41635.23
Mean squared error of testing=68934.45
GBDT Regression Evaluation:
Fitting score of training set=1.00
Fitting score of testing set=0.82
Mean squared error of training=459.15
Mean squared error of testing=47936.91
```

IV. Conclusion

This work compares the housing price predictions produced by CART decision tree and GBDT decision tree. Our research and practice are aimed at getting better accuracy of housing price prediction. In this work, more precise prediction is done by GBDT regressor. All of these predictions can only be possible by analysing different features such as position, specifications, and income. In addition to comparing the efficiency of the two decision trees, we find that different built-in parameters have different effects on GBDT efficiency. The parameter adjustment process also plays an important role in the accuracy of regression.

Reference

[1] Q. Ma, “Discussion on the forecasting method of housing price,” 2014.

[2] L. Breiman, H. J. Friedman, R. A. Olshen and C. J. Stone, Classification and regression trees, Routledge, 2017.

[3] ML Wiki, “Cost-Complexity Pruning,” [Online]. Available: http://mlwiki.org/index.php/Cost-Complexity_Pruning. [Accessed 18 June 2022].

[4] L. Breiman, “Bagging predictors,” *Machine learning,* vol. 24, no. 2, pp. 123-140, 1996.

[5] J. H. Friedman, “Greedy function approximation: A gradient boosting machine,” *Annals of Statistics,* p. 1189–1232, 2000.

[6] scikit learn, “sklearn.datasets.fetch_california_housing,” 22 Mar 2022. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_housing.html. [Accessed 24 Jun 2022].

[7] R. K. Pace and R. Barry, “Sparse spatial autoregressions[J],” *Statistics & Probability Letters,* vol. 33, no. 3, pp. 291-297, 1997.

[8] Pedregosa et al., “Scikit-learn: Machine Learning in Python,” *JMLR 12,* pp. pp. 2825-2830, 2011.

[9] scikit learn, "sklearn.ensemble.GradientBoostingRegressor," 2022. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html. [Accessed 25 Jun 2022].

Appendix – Source Code in Python

```
# GBDT Implementation
import numpy as np
import pandas as pd
from pkg_resources import get_build_platform
import sklearn
from sklearn.model_selection import train_test_split
from sklearn import pipeline
from sklearn.metrics import mean_squared_error
from sklearn import preprocessing
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelEncoder, MinMaxScaler,
OneHotEncoder, scale
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score, mean_absolute_error,
mean_squared_error
import matplotlib.pyplot as plt

# ignore system warnings
import warnings
warnings.filterwarnings('ignore')

# import dataset
# load dataset
cal = pd.read_csv(r'housing.csv', encoding='utf-8')
# # print information of dataset
# cal.info()
```

```
# data cleaning
# Separate numerical (continuous) type data
numerical_cols = list(cal.select_dtypes(include =
'float64').columns)
# Remove prediction column
numerical_cols.remove('median_house_value')
# Separate categorized (discrete) data
cat_cols=list(cal.select_dtypes(include = 'object').columns)

# Split testing set and training set randomly and in 1:5 ratio
x_train, x_test, y_train, y_test =
train_test_split(cal.drop('median_house_value', axis = 1)
, cal['median_house_value'], test_size=0.2, random_state=42)

# data pre-processing: substitude missing value with median
value
num_process_pipe = Pipeline([('imputer',
SimpleImputer(strategy='median')), ('scaler',
MinMaxScaler())])
cat_process_pipe = Pipeline([('ohe', OneHotEncoder())])
preprocessing_pipe = ColumnTransformer([('num_cols2',
num_process_pipe, numerical_cols), ('cat_cols',
cat_process_pipe, cat_cols)])

# apply pre-processing
preprocessing_pipe.fit(x_train)

# CART modeling by scikit-learn
```

```
CART=Pipeline([('preprocessing', preprocessing_pipe),
('model', DecisionTreeRegressor(
    criterion='mse' # tree split strategy of regression
    # pruning parameters
    , max_depth = 13
    , min_samples_split=2   # minimum sample number of the
regression tree when splitting
    , min_samples_leaf=1    # minimum sample number on the
leaf node of the regression tree
    , max_features='log2' # maximum number of candidate
features for selecting split features
    # in this case, max_features=log2(n_features), which
decreases the squared error and increases deviation

    , random_state=None # No random seed is set
))])

# GBDT modeling by scikit-learn
GBDT=Pipeline([('preprocessing', preprocessing_pipe),
('model', GradientBoostingRegressor(
    loss='ls'   # least squared loss
    , learning_rate=0.1   # step length
    , n_estimators=300  # iteration times
    , subsample= 0.8    # the smaller subsample, the stronger
gradient boosting
    , criterion= 'mse'  # tree split strategy of regression

    # pruning parameters
    , min_samples_split=2   # minimum sample number of base
learner tree when splitting
```

```
    , min_samples_leaf=1    # minimum sample number on the
leaf node of the base learner tree
    , max_depth=13  # max depth of base learner

    , init=None # initialize loss estimator
    , random_state=None # No random seed is set
    , max_features='log2'   # maximum number of candidate
features for selecting split features
    # in this case, max_features=log2(n_features), which
decreases the squared error and increases deviation
    , verbose=1 # print learning process
    , warm_start=False))])

# set output target
tar =
['longitude','latitude','housing_median_age','total_rooms','to
tal_bedrooms'
        ,'population','households','median_income','ocean_prox
imity','actual_median_house_value','predicted_median_house_val
ue']

# apply CART model
CART.fit(x_train, y_train)
# make CART prediction
CART_y_train_prediction = CART.predict(x_train)
CART_y_test_prediction = CART.predict(x_test)
# convert prediction to attribute matrix
CART_train_prediction = np.column_stack((x_train, y_train,
CART_y_train_prediction))
```

```
CART_test_prediction = np.column_stack((x_test, y_test,
CART_y_test_prediction))
# transform the prediction results into dataframes
df_CART_p1 = pd.DataFrame(CART_train_prediction, columns=tar)
df_CART_p2 = pd.DataFrame(CART_test_prediction, columns=tar)
# fitting evaluation
CART_score1 = CART.score(x_train, y_train)
CART_score2 = CART.score(x_test, y_test)
# calculate mean squared error of training and testing process
CART_mse1 = np.sqrt(mean_squared_error(y_train,
CART_y_train_prediction))
CART_mse2 = np.sqrt(mean_squared_error(y_test,
CART_y_test_prediction))

# apply GBDT model
GBDT.fit(x_train, y_train)
# make GBDT prediction
GBDT_y_train_prediction = GBDT.predict(x_train)
GBDT_y_test_prediction = GBDT.predict(x_test)
# convert prediction results to attribute matrix
GBDT_train_prediction = np.column_stack((x_train, y_train,
GBDT_y_train_prediction))
GBDT_test_prediction = np.column_stack((x_test, y_test,
GBDT_y_test_prediction))
# transform the prediction results into dataframes
df_GBDT_p1 = pd.DataFrame(GBDT_train_prediction, columns=tar)
df_GBDT_p2 = pd.DataFrame(GBDT_test_prediction, columns=tar)
# fitting evaluation
GBDT_score1 = GBDT.score(x_train, y_train)
GBDT_score2 = GBDT.score(x_test, y_test)
```

```
# calculate mean squared error of training and testing process
GBDT_mse1 = np.sqrt(mean_squared_error(y_train,
GBDT_y_train_prediction))
GBDT_mse2 = np.sqrt(mean_squared_error(y_test,
GBDT_y_test_prediction))

# output CART prediction results
print("CART Training Prediction")
print(df_CART_p1)
print("CART Testing Prediction")
print(df_CART_p2)
# output GBDT prediction results
print("GBDT Training Prediction")
print(df_GBDT_p1)
print("GBDT Testing Prediction")
print(df_GBDT_p2)
# output results
print('CART Regression Evaluation:')
print('Fitting score of training set=%0.2f' % (CART_score1))
print('Fitting score of testingset=%0.2f' % (CART_score2))
print('Mean squared error of training=%0.2f' %(CART_mse1))
print('Mean squared error of testing=%0.2f' %(CART_mse2))
print('GBDT Regression Evaluation:')
print('Fitting score of training set=%0.2f' % (GBDT_score1))
print('Fitting score of testing set=%0.2f' % (GBDT_score2))
print('Mean squared error of training=%0.2f' %(GBDT_mse1))
print('Mean squared error of testing=%0.2f' %(GBDT_mse2))
# plot histogram of the predictions
# GBDT_hist_test = plt.hist(GBDT_y_test_prediction,bins=10)
# plt.show()
```