P1: PREDICTING BOSTON HOUSING PRICES

1 Data Exploration

Boston data statistical analysis:

Data size: 506
Number of features 13
Price minimum value: 5.000
Price maximum value: 50.000
Price mean value: 22.533
Price median: 21.2
Price STD: 9.188
Skewness = 1.1048, kurtosis = 1.4686 (frequency of prices)

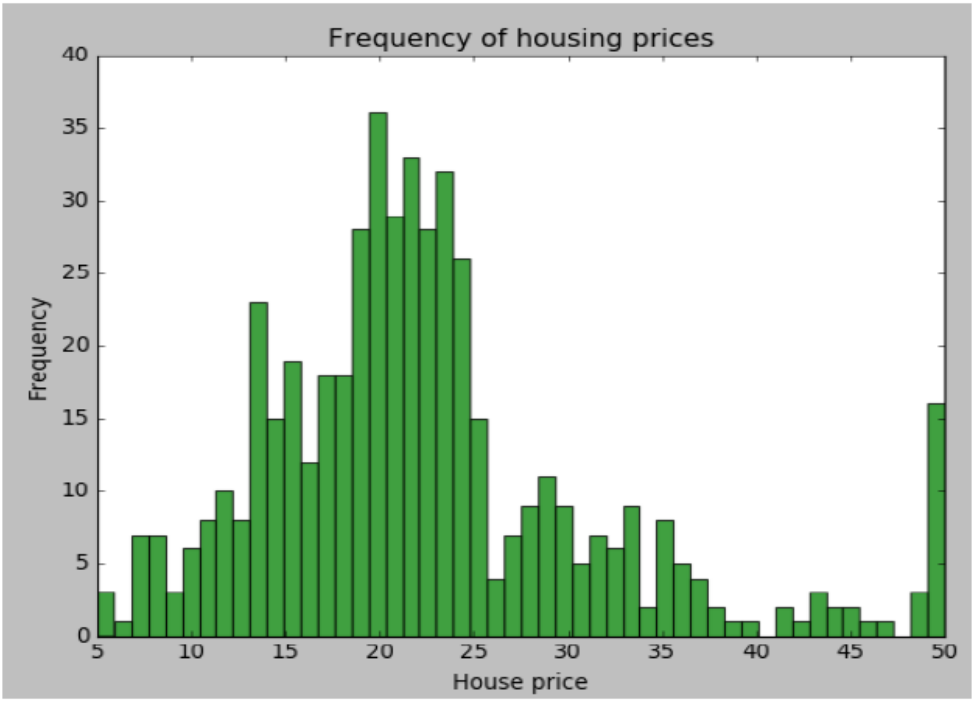


Figure 1. Frequency of housing prices

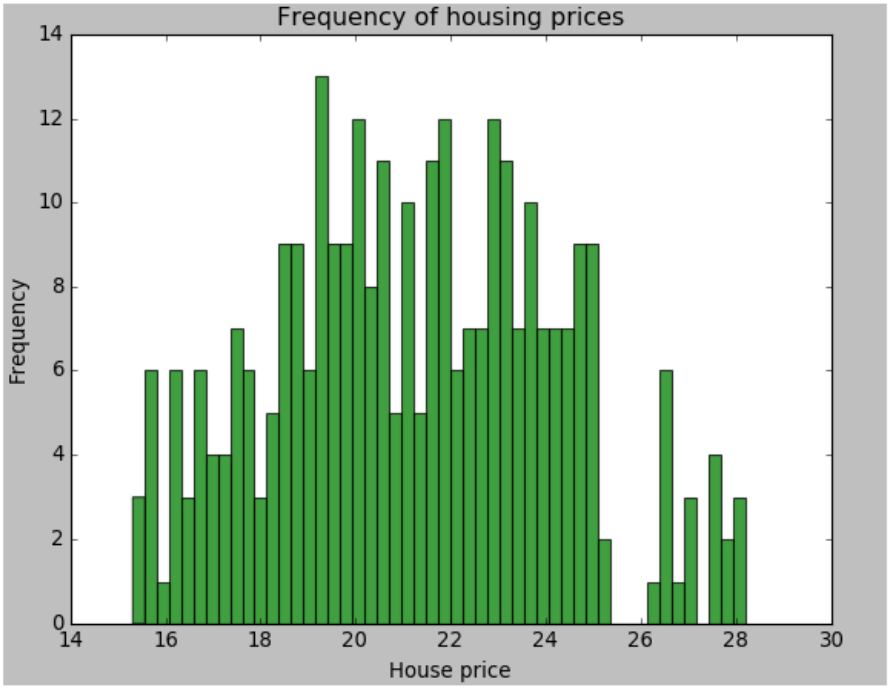


Figure 2. Frequency of housing prices without the outliers (0.25)

2 Evaluating Model Performance

2.1 Performance Metric

For this project, the Mean Square Error (MSE) error scoring metric was used. The reasons are:

1. MSE is a commonly used error measuring metric;
2. MSE has more mathematical properties than, for example, the Mean Absolute Error (MAE).
3. MSE is differentiable and that allows the use of different methods (e.g.,ANN, regression);
4. MSE is more sensitive of errors in data (the difference between the fact and the predicted value) than MAE.

In fact, Fig.1 shows that the number houses within the range of 48-50K exceed 15, a fact we need to take into
account. This means that MSE is more precise for the model building than onthers. This is also true for the houses of the price under 48K.

In graphs of the next page, MAE and MSE metric are very similar (with a single random state for a regressor). If we evaluated the square root of MSE, we would obtain a value which would be slightly larger than the MAE value.
So, I choose MSE because DecisionTree regressor can find the gradient or the slope to minimize the error.

2.2 Testing/Training Split

Boston data set was split into the testing and the training data sets. If we use all the data for training, it would produce a model overfitting problem. If a model was overfitted, it would become too complex and have poor performance in general. The overfitted model would lose its generality and become very sensitive to data errors.
The author used the ShuffleSplit (from sklearn.cross_validation) function to spilt data with 25% of testing data. This function belongs to the holdout cross validation method. See implementation in the split_data function.

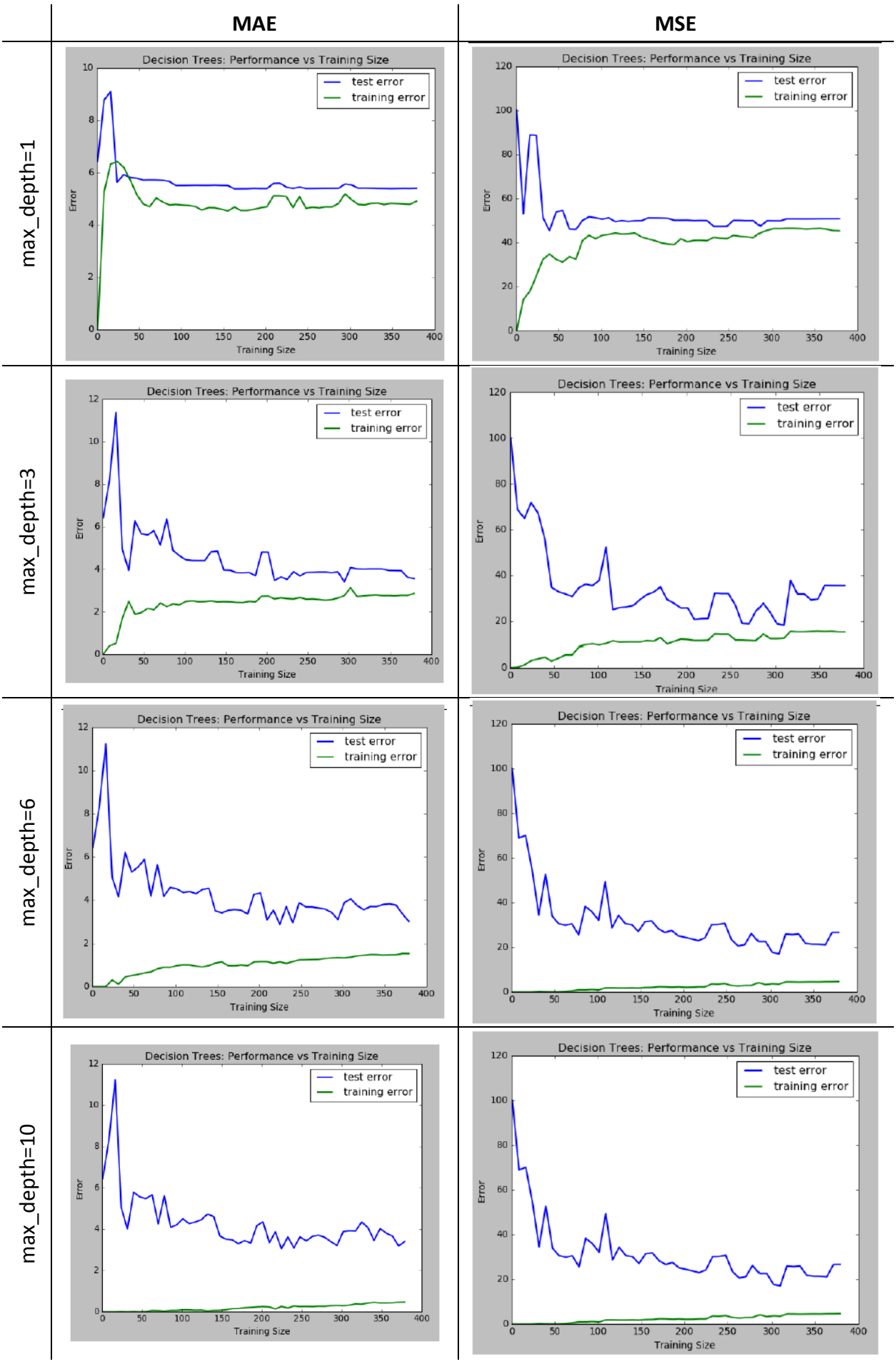


Figure 3. Model's performance for different depth levels

Gridsearch implementation

For the GridSearch function, we need to provide the following variables:

- **estimators.** In this case, I provide the regressor object, which is the DecisionTreeRegressor estimator;
- **estimator's parameters.** Here, I provide 'max_depth':(1,2,3,4,5,6,7,8,9,10) for the DecisionTreeRegressor and check GridSearch the depth of each level;
- **scoring function**. In my case, I provide the my_scorer object, which is defined by the function make_scorer with the parameter greater_is_better=False (for reducing lose) and the performance function performance_metric (particularly, MSE);
- **cv parameter (default 3-folds)**. The number of the Cross-validation folds is used in model fitting. I use 11, because the Boston data is not large, and cross-validation, with a large number of fords, gives more accurate results.

See implementation in the fit_predict_model function.

After the GridSearch application, we would take the best parameter: max_depth of the model. The GridSearch searches best combination of estimator parameters. In my case, the lowest value is the performance metric. As the scoring function is defined by the parameter greater_is_better=False, the values of the score function are negative, and we only need the minimal absolute value.
The minimal value of the scoring function provides the complexity level for the best model. Generally, the curve of the model's complexity is concave. The best level of a model depth can be found at the lowest point of the curve.

2.3 Cross Validation

Cross validation is a model-evaluation method. The main idea is to classify the data into a training set and a testing set (minimum of two folds - holdout method). After the model training, it will be tested for data that has not been observed yet. The holdout method has a disadvantage – the model training and the testing results largely depends on individuals in the samples; and this method provides high variance of error-evaluation. To resolve such a problem, the K-fold method will be used.
Cross validation is useful in the GridSearch, where a small variation of error score value would be present, and the best parameter set is more robust for regression in the classification function.
The author used the CV=11 parameter in the GridSearch, which means using 11 folds for the model evaluation and mean error-calculation. Instead of 3 folds (default), author uses 11 folds, because the size of the Boston data is relatively small. In experiments, the author obtained more robust results, and less error.

3 Analyzing Model Performance

3.1 Learning Curves and Training Analysis

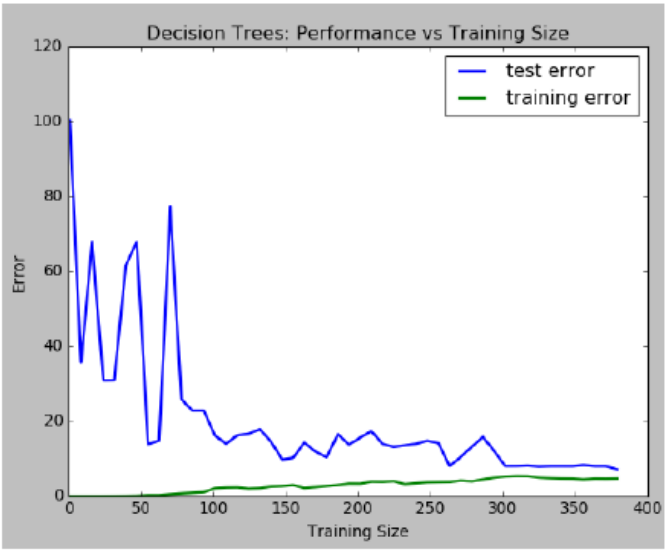


Figure 4. The model's performance vs Training size, max_depth=6

The MSE performance metric, data split, and regressor with random state=1111.

General observation: with an increasing training size, the training error increases and the testing error decreases, until they meet in the middle. Then training size is small, the training error is near zero, due the fact that the model is too simplistic and fits with the data well, yet the size of the training set increases, the training error increases, and the testing error decreases.

3.2 Learning Curves and Bias & Variance Analysis

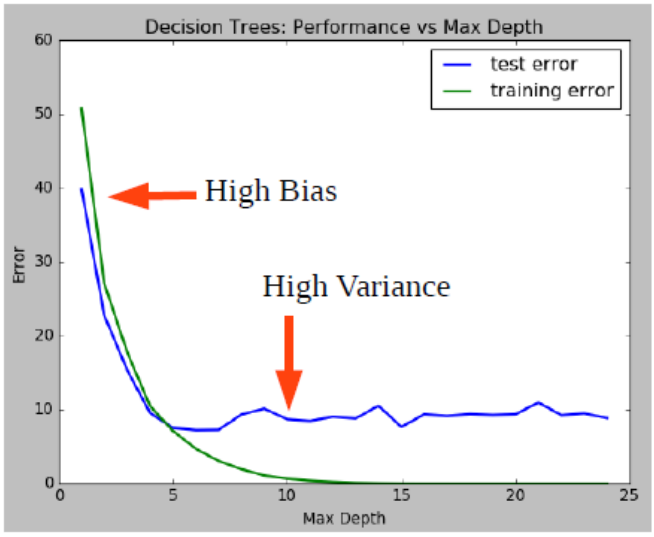


Figure 5. Model's bias and variance

The model in the Figure 5 with max_depth=1 is highly biased (there is a high error for both the training and the testing set), and it is under-fitted or too simplistic. The high biased model is too simple (the DecisionTree of a single level) and will never be a good to fit with the Boston data.
The model with max_depth=10 has high variance, its training error is very low, while the testing error is high. The model is overfitted or too complex. The overfitted model tends to fit the training data very well, hence the low training error, while the model is too sensitive or not general enough for the testing data, hence the high testing error. I think, the useful model is with max_depth=5 or 6.

3.3 Error Curves and Model Complexity

The increasing model complexity (see Figure 4) the training and testing curves converge. Hence, their variance decreases. With the increasing, the model complexity bias decreases. After the optimal level of the model complexity is reached, the error curve slopes to zero and a large gap between curves appears (high variance).
Before max_depth=5, the testing curve and the training curve converge (variance decreases) see Figure 5, yet after the 5-th model becomes overfitted with high variance, the training error approaches the zero value. The best model max_depth is near 5 and 6, where the training and the testing error is reduced as much as possible, yet over 5, they diverge. This means that model becomes overfitted.

3.4 Picking the Optimal Model

In Figure 5, you can see the correlation of the testing and the training errors. Before max_depth=5, the testing and the training curve converge (the earlier model is too general, or it is under-fitted), yet over 5, the model becomes overfitted and very sensitive to errors within the testing set. The best model has the variable max_depth value of 5, where the training and the testing error is reduced as much as possible, and over 5, they diverge. This means that the model becomes overfitted.

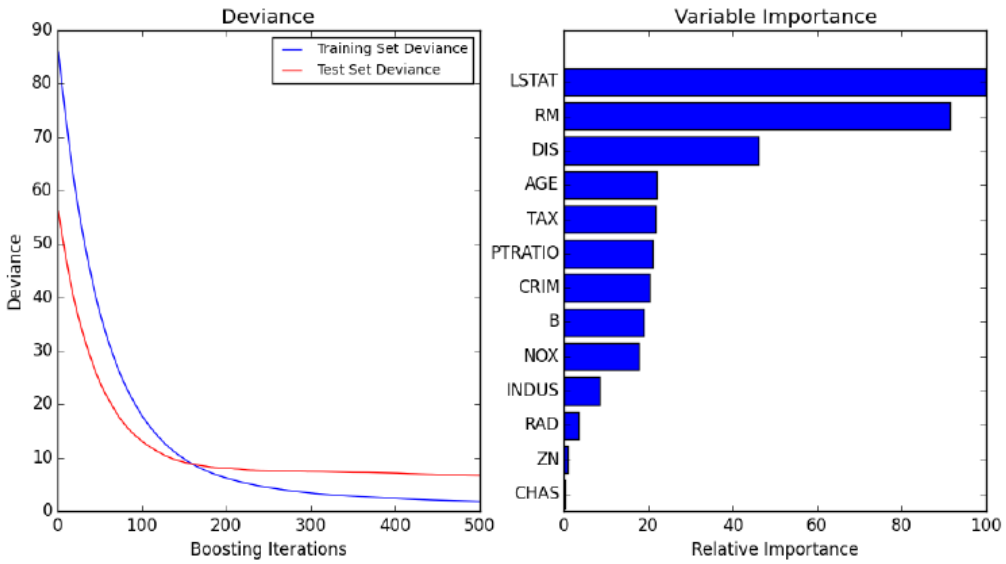
4 Model Prediction

4.1 Predicted Housing Price

The Grid search was used with CV=11 (cross-validation folds).
For the estimator, the following parameters were used:

- 'max_depth':(4,5,6);
- 'max_features': [3,5,6,7,8,9,10] – n regards to the importance of the feature.

The outliers had reduced within the range of 5%. I continuously ran prediction function, 10 times, after which I selected a particular run with the minimal MSE.



from: http://scikit-learn.org/stable/auto_examples/ensemble/plot_gradient_boosting_regression.html

Figure 6. Features importance in Boston dataset

Output:

```
The best score is 17.41 and if we compute square root then error is: +/-4.17K
DecisionTreeRegressor(criterion='mse', max_depth=6, max_features=7,
max_leaf_nodes=None, min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False, random_state=None,
```

```
splitter='best')
Best parameters:
{'max_features': 7, 'max_depth': 6}
Details of SearchGridCV:
mean: -29.44738, std: 39.19877, params: {'max_features': 3, 'max_depth': 4}
mean: -23.63674, std: 20.45759, params: {'max_features': 5, 'max_depth': 4}
mean: -23.21252, std: 19.61036, params: {'max_features': 6, 'max_depth': 4}
mean: -21.55645, std: 18.66106, params: {'max_features': 7, 'max_depth': 4}
mean: -21.94794, std: 18.15543, params: {'max_features': 8, 'max_depth': 4}
mean: -18.87749, std: 13.84446, params: {'max_features': 9, 'max_depth': 4}
mean: -21.94349, std: 20.32550, params: {'max_features': 10, 'max_depth': 4}
mean: -31.11077, std: 31.33363, params: {'max_features': 3, 'max_depth': 5}
mean: -20.44386, std: 18.63846, params: {'max_features': 5, 'max_depth': 5}
mean: -23.31186, std: 23.87093, params: {'max_features': 6, 'max_depth': 5}
mean: -20.72869, std: 17.35518, params: {'max_features': 7, 'max_depth': 5}
mean: -22.69857, std: 20.38567, params: {'max_features': 8, 'max_depth': 5}
mean: -21.47821, std: 18.55807, params: {'max_features': 9, 'max_depth': 5}
mean: -19.94218, std: 18.05281, params: {'max_features': 10, 'max_depth': 5}
mean: -20.20094, std: 16.68052, params: {'max_features': 3, 'max_depth': 6}
mean: -23.69743, std: 22.13642, params: {'max_features': 5, 'max_depth': 6}
mean: -20.63111, std: 18.07963, params: {'max_features': 6, 'max_depth': 6}
mean: -17.41893, std: 10.53325, params: {'max_features': 7, 'max_depth': 6}
mean: -21.47617, std: 22.24779, params: {'max_features': 8, 'max_depth': 6}
mean: -19.83095, std: 13.30438, params: {'max_features': 9, 'max_depth': 6}
mean: -21.97624, std: 15.61239, params: {'max_features': 10, 'max_depth': 6}
Best score
-17.4189304172
And predicted with max_depth=6 and features_num=7
House: [11.95, 0.0, 18.1, 0, 0.659, 5.609, 90.0, 1.385, 24, 680.0, 20.2, 332.09, 12.13]
Prediction: [ 21.35555556]
```

Using the RandomizedSearchCV for the best parameter estimation and acquiring slightly better results (by MSE). Output:

```
Best estimator:
DecisionTreeRegressor(criterion='mse', max_depth=5, max_features=10,
max_leaf_nodes=None, min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False, random_state=None,
splitter='best')
Best parameters:
{'max_features': 10, 'max_depth': 5}
Details of SearchGridCV:
mean: -24.52015, std: 22.30312, params: {'max_features': 3, 'max_depth': 4}
mean: -24.61778, std: 18.79132, params: {'max_features': 3, 'max_depth': 6}
mean: -24.90807, std: 23.81418, params: {'max_features': 5, 'max_depth': 5}
mean: -18.24851, std: 11.21149, params: {'max_features': 9, 'max_depth': 5}
mean: -23.37192, std: 21.00552, params: {'max_features': 7, 'max_depth': 6}
mean: -17.36383, std: 12.60198, params: {'max_features': 10, 'max_depth': 5}
mean: -19.68312, std: 13.05250, params: {'max_features': 6, 'max_depth': 4}
mean: -21.72699, std: 20.56604, params: {'max_features': 9, 'max_depth': 4}
mean: -23.42656, std: 26.25868, params: {'max_features': 3, 'max_depth': 5}
mean: -19.89605, std: 19.01176, params: {'max_features': 10, 'max_depth': 4}
Best score
-17.3638306179
House: [11.95, 0.0, 18.1, 0, 0.659, 5.609, 90.0, 1.385, 24, 680.0, 20.2, 332.09, 12.13]
Prediction: [ 20.4338843]
```

4.2 Comparing the Model Price to the Housing Statistics

In the previous section, the predicted house price was 21.35K:
[11.95, 0.0, 18.1, 0, 0.659, 5.609, 90.0, 1.385, 24, 680.0, 20.2, 332.09, 12.13].

And the model's best MSE here equal to 17.42 IN comparison to the Chapter 1 model, the MSE value is within the STD range. Also, the predicted house price is near median and the mode of the Boston data prices. I think, the predicted value is reasonable. When comparing the model parameters, obtained by using the GridSearchCV, with the values predicted with NearestNeighbors, we can conclude that they are very similar. The **Nearest Neighbors average: 18.24.**

```
Nearest Neighbors average: 18.24
List of evaluations
Max_depth       Num_of_features         MSE     Predicted
[[  5.           8.           13.87521461  20.12444444]
 [  5.          10.           14.31202447  20.59333333]
 [  6.           6.           14.05192923  19.74754098]
 [  4.           7.           13.53871108  20.44672131]
 [  5.           5.           14.05687673  20.60215054]
 [  4.           6.           14.4215583   17.89166667]
 [  5.          10.           13.43630104  20.47711864]
 [  4.          10.           13.80123351  20.02209302]
 [  4.           9.           13.22574183  20.59834711]
 [  5.           9.           13.3115675   20.47068966]
 [  6.           8.           13.07658936  20.47307692]
 [  5.           7.           13.48646741  20.35842697]
 [  5.           3.           14.42335257  20.81111111]
 [  6.           6.           14.06869726  20.4075    ]
 [  6.           9.           14.2012228   20.3974359 ]
 [  5.           8.           13.21212075  20.4       ]
 [  4.           9.           13.03999278  20.38583333]
 [  5.           8.           12.84309708  20.47711864]
 [  4.          10.           14.56396895  20.468     ]
 [  5.           5.           14.44688896  21.01315789]]
MSE: 12.8430970805 Predicted cost: 17.8916666667
```

Figure 7. Debug information for Nearest Neighbors algorithm