Udacity.com: Machine Learning Course

Boston Housing Price Prediction

Student: Diego Fanesi

Introduction

Problem Description: The dataset includes data from 506 house prices organized in 13 features. We have to build a machine learning to predict the house pricing using this data.

Data Analysis: The prices in the dataset go from 5 to 50 with a mean price of 22.532, which is fairly close to the median 21.2. The dataset is characterized by a Standard Deviation of 9.188.

Choosing the right performance metric function

The scikit-learn library provides a lot of different metrics. The Boston house pricing prediction is a regression problem because we try to predict a continuous value estimating the relationship between other variables with the output, therefore we need a regression metric function. The functions provided are 5:

- Explained variance score
- Mean absolute error (MAE)
- Mean squared error (MSE)
- Median absolute error
- The coefficient of determination

I tested all of them and they provide similar results in our case: the best max_depth found is between 4 and 6 and the final prediction is 20.5 < p(x) > 21.8 depending on the costing function chosen and a random factor due to the random shuffling before splitting the data between training and test set.

Explained variance score: This metric is based on the following formula

$$\texttt{explained\_variance}(y, \hat{y}) = 1 - \frac{Var\{y - \hat{y}\}}{Var\{y\}}$$

The function has a maximum value of 1 which is reached when the prediction has the same value of the correct value. The variance of the prediction is normalized by the variance of the y vector so the values have a generally small value. In our case the values were always included between -1 and 1.

Mean absolute error:

$$\text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i| \,.$$

This is one of the most used cost functions for regression problems. It generally works well and the best value is 0 which is also the best possible value. Lower values represent a better fit of the data and the function is linear.

Mean squared error:

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

This scoring function is similar to the previous one but the squared values add an interesting difference. The function has higher values for differences > 1 and substantially lower for differences < 1, with 0 as best possible value. This means that high deviation from the real value will be penalized much more than lower deviations. This will make the prediction error converge quickly in the area of -1 < y'-y < 1, with a substantially slower variation after that. This function might be helpful for cases with high variance and low amount of data because the type of function needs less regularization compared to the previous one.

Median absolute error: This metric is based on the median value, this means that a single example with an exceptional value that is very far from the average might negatively influence all the other predictions. This function is the right choice if we want to make sure that our prediction is never too far from the real value. In our case we rather have one vary bad prediction and have a very good prediction for all others rather than all fairly bad predictions just because one house has been significantly overpriced.

The coefficient of determination: This function is similar to the explained variance in terms of results and characteristics.

The final choice: We have 506 instances to divide between cross validation and training sets with values between 5 and 50 and a standard deviation of 9.1, which means that our model will be probably overfitting because of the low amount of data available and an error contained between -1 and 1 is more than acceptable. We could use the MSE but the squared value in the metric will cause the function to be not so stable, meaning that fitting a regularization parameter might be difficult. For these reasons I decided to use MAE.

Splitting data in multiple sets

The dataset provided need to be split at least in two sets where one is the training set and another one is the test set. The training set is used to set the parameters of the machine learning function, in this case the regression tree. This means that the tree structure will be built to fit the training set as good as possible, therefore we cannot test final results of our machine learning with the training set because it would give us too optimistic compared to the real performances of the machine on new data. Therefore, we split the data in two sets, a training set and a test set making sure we use most of the data (70%) for the training and we reserve some unused data to test the performances of the machine on brand new data.

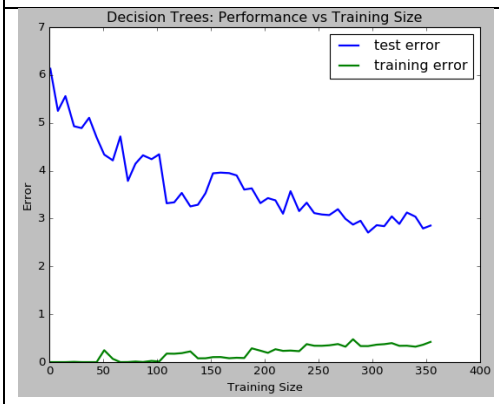
Cross Validation and GridSearch

The algorithm of the cross validation process divides the dataset in k-folds and select one fold at the time to be the cross validation set. The model will be trained on all the other folds except the selected one and the process will iterate through the dataset selecting a different fold each time and retraining the model again on the new dataset. The results obtained will be averaged to get final value. This methodology is used to fit parameters and to test the model, in our case is used to fit the max_depth parameter with the gridsearch function. The default k value for gridsearch is 3 which means the whole dataset will be divided in three parts, the model will be trained on two of them and the remaining one will be used as cross validation. This means that approximately 30% of the data will be excluded from the training and used for cross validation. However, we don't have a lot of data and 70% of the data might be not enough for training the model. I decided to increase the k value to 5 so the cross validation will decrease from 30% to 20%, the training model size will grow from 70% to 80% and the cross validation process will run for 5 times instead of 3, providing better results in case of low amount of data.

Max_depth represents the maximum number of levels the tree can reach, that implies the number of partitions our regression tree will create, determining their dimension. The maximum depth defines the complexity of the model and can be used as regularization parameter that can change how our model behaves in terms of under fitting or over fitting. We have to set this parameter on data that hasn't been used for the model training otherwise we will obtain a choice that might be optimal for the training set but the model might not generalize well. Setting this parameter to the training error will make our model overfitting the training data and performing poorly on new data.

Learning curves and training analysis

The software provides a graph of the average prediction error in the training set and in the test set. The function in run multiple times in a training set of increasing size and every time the function is tested on the test set. The average errors are reported into a graph which is reported below:

Max_depth=10	Max_depth=4	Max_depth=1
		

Increasing the training set size it becomes more difficult to perfectly fit the training set, therefore the training error increases. On the other hand the test error becomes smaller with the increasing "experience" of the machine.

This graph is particularly interesting to analyze how the learning machine is performing in the data set:

- If the two lines get very close with the increasing of the training set but both the training error and the test error are high, then the model is under fitting.
- If the test error stays high but the training error is very low then the model is over fitting the training set.

We can see those concepts in practice changing the depth of the regression tree.

We can see how the model is over fitting the training data with a max_depth=10. The model is behaving quite well with max_depth=4. However, we can see that the difference between test and training error is still substantial. This can be probably fixed collecting some more training data. The model is under fitting the training data with max_depth=1, because both the training and the test error are fairly high.

To better understand how the complexity of the model impacts the performances of the machine we can look at the complexity model reported below:

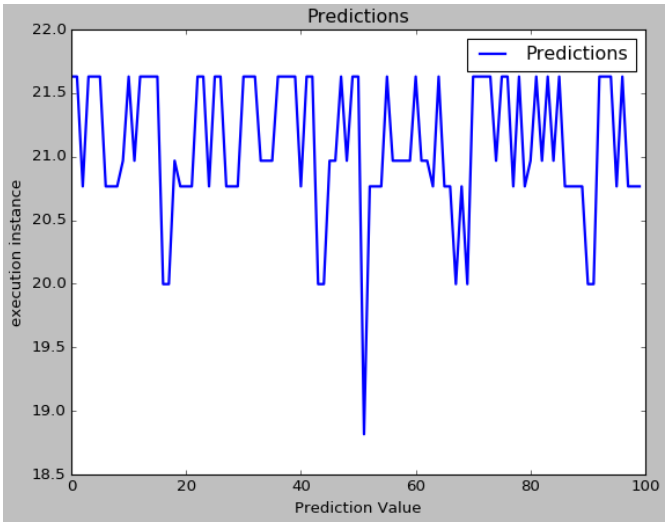
With the increasing of the complexity we can see how the model fits the training data every time better. However the performances get worse and worse after max_depth=5 because the model can perfectly fit every single instance of the training example but even a single value that is exceptionally high or low will impact the overall model. We can say that when the model is overfitting is very specific for the training data but does not generalize well on new data.

To choose a good max_depth value I would consider the difference between the training error and the test error, and the value of the test error compared to the values obtained for other max_depth value close to the point where the test error function starts stabilizing.

The best choice seems to be max_depth=5 because the test error and the training error are not too far and the test error has a good local minimum at that value.

The final price prediction

To get a better understanding on how does the random shuffling affect the final prediction of the machine and what would be the most correct prediction, I run the model in a cycle for 100 times, shuffling and retraining the model again every time. I collected the results in a list and I made a graph that I included in this document (graph on the left).



Furthermore I calculated the mode rounding the values at 3 decimals. The mode was 21.63, therefore I believe this is the right prediction. The prediction is fairly close to the mean and the median of the data and the standard deviation is higher than the difference between the prediction and the mean, therefore we can conclude that the prediction is correct.