Predicting Housing Prices in Boston

Saif Abid

Section 1: Statistical Analysis and Data exploration:

Number of data points:

Provided was 506 data points corresponding to housing prices for the given dataset.

Number of features:

There were 13 features.

Maximum and Minimum housing prices:

The maximum reported price was $50,000, and the minimum was $5000

Mean and median Boston housing prices:

The mean price was $22,532.81, while the median was ~$21200.00

Standard deviation:

The standard deviation was ~$9188.01

Section 2: Evaluating Model Performance

Which measure of model performance is best to use for predicting Boston housing data and analyzing the errors? Why do you think this measurement most appropriate? Why might the other measurements not be appropriate here?

Considering that we are looking at a regression problem and not classification - choosing metrics such as accuracy and recall did not seem appropriate. I narrowed my possible metric choices to *mean squared error (MSE)* or *mean absolute error (MAE).* Of the two, I ended up choosing to use MSE. When trying to analyze errors, the MSE is a good choice because it will emphasize larger errors as opposed to smaller ones due to the square - which also implies our error for any datapoint is always positive so cancellation of errors will not occur. Lastly, another reason why MSE was chosen as opposed to MAE is because the large errors in MAE could get hidden(or not clearly evident) when we take the average. The

emphasis of larger errors will ensure even with averaging if there is very large errors, they will be shown.

Why is it important to split the Boston housing data into training and testing data? What happens if you do not do this?

The reason it's important to split your training data is because otherwise your will have no way to measure your performance accurately. The reason for this is because if you don't split your data, then your only way to 'test' afterwards how well your performance is would be

1) running it against a subset of your already trained data.

2) Don't measure performance and have it run on brand new data.

Both cases are horrible ideas. In the first case when you decided not to split into test/train, you'll be forced to test on data which your model has already been trained on. As a result it will perform really well and give you the perception that it will perform just *as* well on unseen data. This is wrong and it's performance to *new data* is completely unknown. The second case is also a bad idea, because you have no way to determine if your model has any sort of well defined performance. In essence, if you don't measure any performance, than any predictions you make on *new* data become very difficult to justify.

What does grid search do and why might you want to use it?

Grid search is a way to tune our parameters for a given learning model with predefined options. Giving grid search a dictionary of possible values for different model tuning parameters and our learning model, it will run through the various parameters and fit your data according to the combination which performed the *best* (the caller can define a scoring function to dictate what it means to be good performing vs bad)

Why is cross validation useful and why might we use it with grid search?

Cross validation is useful because it helps us avoid overfitting. By taking our data and allowing us to train on a subset of that data and test on another portion of the data which our model has not seen yet, we can ensure that the performance of our model simulates a similar situation of that which it might have to face when it sees brand new unseen data. If you don't cross validate, you'd be testing on the same data your model has just seen to train

on and thus your performance would not tell you anything meaningful, nor will you be able to understand how your model will work on unseen data.

Cross validation is used with grid search because as grid search works its way through various combinations of parameters, it can train a model on a portion of the data given a set of parameters and test it against the remaining data which it has not trained on (cross validation at work here). Doing this sort of cross validation testing with various parameter combinations helps gridsearch ensure that the tuned parameters it's fitting our model with are infact tuned for performance on *unseen* data, not just on performing well with data it's already being trained on.

Section 3 - Analyzing Model Performance:

Look at all learning curve graphs provided. What is the general trend of training and testing error as training size increases?

As training size increases, training error increases and testing error tends to decrease.

Look at the learning curves for the decision tree regressor with max depth 1 and 10 (first and last learning curve graphs). When the model is fully trained does it suffer from either high bias/underfitting or high variance/overfitting?

At depth 1, when the model is fully trained, it will suffer from high bias/underfitting. This can be seen from the fact that the error as training size increases tends be around the same for both training and test data. This makes sense that this is an indicator of underfitting(high bias) because it implies the model was too simple and thus could not perform well on either the test or train data.

At depth 10, the model is overfit and has high variance. This can be seen from the fact that training error is very low(close to 0) at high training sample size, however the test error is still very high. This is a sign of overfitting because it means the model fit the training data extremely well, but it could not generalize enough for unseen data - hence a very high error for its test data.

Look at the model complexity graph. How do the training and test error relate to increasing model complexity? Based on this relationship, which model (max depth) best generalizes the dataset and why?

As model complexity increases, training error decreases to almost ~0. Test error however decreases as complexity increases to a certain max-depth (~6) before it begins to stop decreasing. That said, the best model is the one with max depth 6.

The reason for this is because as model complexity increases, initially both test and train error decrease their error because the model is building/finding better more complex relations between the data to generalize it well. However eventually the test error stops decreasing while the train error continues too - this is because of overfitting. The model is becoming too complex and no longer able to generalize the data well, hence it overfits the training data but fails to perform well on the test data.

Ultimately the best model is then the one which is a balance between the overfitting nature of a really complex model(high variance) and the underfitting nature of a simple model(high bias) - which in our case is the model with max-depth tuned to 6.

Section 4 - Model Prediction:

The predicted Housing price:

$20765.99. Max-depth:6. Found after running model 1000 times to determine the most frequent, best chosen max depth by grid search.

Does this make sense(model justification)?

At a first glance one might think this is a very well defined model since the prediction is close to average and within 1 std deviation from the average. However the average doesn't give us any relevant information on the feature vector which our model predicted this value for. I would like to rather judge the validity of my model based on the learning curves/performance metrics and model complexity chart. When looking at the model complexity graph, we can see a 13-15% error in our test data and a 4-5% error in our training data(for a complexity of ~6). I believe that although there is room for improvement with our error and performance metrics (from either more data to train from or tuning other parameters), the difference in error is still low enough to conclude that the price is still within a reasonable means.