Gilad Gressel
December 7th, 2015

# Project 1, Prediction Boston Housing Prices

## Statistical Analysis and Data Exploration

The housing data contains the following statistics. There are 506 individual samples (homes), each home has 13 features. The total number of features for the entire dataset is 6578. The minimum amount that a home sold for $5,000. The maximum amount a home was sold for is $50,000. The mean price of a home is $22,533. The median price of a home is $21,200. The standard deviation of price for this data set is $9,188

## Evaluation Model Performance

Choosing the best performance metric

This problem is one of regression, therefore our performance metric should evaluate how closes the line fits the data. These types of metrics tend to be distance based (distance from real-values to predicted values). Sci-Kit learn offers three such metrics that are easy to use, mean-squared-error (MSE), mean-absolute-error (MAE), and median-absolute-error. Of these three, either of the first two seem suitable. Median-absolute-error would be helpful if we had significant outliers, however upon visual inspection of the housing prices, this does not seem to be the case.

Between the two remaining choices, there are subtle differences. Due to squaring, MSE amplifies points farther away causing a greater penalty, conversely MAE will treat each point equally. Using MSE as we will eventually optimize our classifier to make smaller mistakes in general (as large differences between the prediction and target value will be penalized more). MAE would not penalize wider “misses” so we may end up with a classifier that does well but occasionally has large errors on single points.

For this dataset either one is fine, however I chose MSE because in order to penalize the larger mistakes. As a real-estate agent, I’d want all my predictions to be ballpark, and avoid really large mistakes on any one client's house-sale. One angry client could ruin my business, while smaller mistakes here and there won’t affect the business as much. Thus MSE is the better choice.

Splitting the data-set

We have to split the data-set into two sections in order to test our trained classifier. If we test our trained classifier on the data that was used to train it, of course it would do perfectly, it built

its model looking at those examples! A classifier must be tested on unseen data, in this way we can be sure that our classifier has *learned* something and is able to generalize to new data.

Cross-Validation

Cross Validation (CV) is a technique which is used to retain data for both training and testing. The problem that many machine learning problems face is shortage of data. As we have seen in the learning graphs, as the training size increases the accuracy of the model increases. When it comes to creating training / testing splits we may face a shortage of data for our training (or testing). CV is a good solution for this. It splits the data into *n* folds and iteratively trains *n-1* folds as one training set, then tests on the remaining fold. This process repeats itself until all folds have been used as a test set (and all folds have been used as part of the training data as well). CV records metrics for each fold and takes the mean of them at the end. In this manner we are able to use all our data for both training and testing.

GridSearchCV

Gridsearch is a method which systematically tests many different parameters of your classifier and retains knowledge of the best parameters. This is an awesome tool because when we are building trained models, we have many different classifier specific parameters, called hyper-parameters, which we need to test. Gridsearch simply automates this process.
We use gridsearch to find the best parameter choice for max_depth in the tree model.

The aim of gridsearch is to find the best parameters for a given model. However if we limit gridsearch to single testing set, we may accidentally overfit our model if the testing set is imbalanced in anyway. A good solution is to use cross-validation with gridsort, in this manner the parameters will be optimized on the entire data set and any random anomalies due to random splitting will be removed.

**Analyzing Model Performance**

The general trend of training and test error as training sizes is that error initially goes up for training and error decreases for testing. Both flatline around 8-9% error rate.

Max depth 1 displays serious bias / underfitting, the error is quite high for training. This is due to the large generalizations the tree is making. Max depth 10 displays high variance / overfitting, training error is virtually non-existent, but testing is much higher. Whenever we see extremely low training error compared to testing, we have a strong case for overfitting.

Looking at the model complexity graph we see that training error drops as depth increases and begins to flat-line around max depth 10. The testing error also drops as model complexity increases (depth) but begins to oscillate horizontally around depth 4 - 5. Ideally we want our

training error to be close to the testing error, if this is possible then we can trust future predictions of unseen data.

The ideal spot seems to be around max depth 4 or 5, this is a place where training error is low and also close to testing error.  This is later confirmed by grid search.

**Model Prediction**

The final model makes a prediction on unseen data (our clients house).  Note that gridsearch chose max_depth = 4, which fits well with the learning / complexity graphs.

DecisionTreeRegressor(compute_importances=None, criterion=mse, max_depth=4,
max_features=None, min_density=None, min_samples_leaf=1,
min_samples_split=2, random_state=None, splitter=best)

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: [ 21.62974359]

The prediction price is 21.69, however this is generally irrelevant because I have no domain expertise or knowledge of the house to understand if it's reasonable.  What is a better "sanity check" is to compare the statistics from the overall predictions the model makes to ground truth given by the original housing prices.

Therefore I compare three sets of data below.  We have the ground-truth provided by housing_prices, a gridsearch prediction that used 100% of the data to train, and a trained classifier on 70% of the data testing on 30%.

| | ground truth | cv-grid search | 70 train / 30 test |
|---|---|---|---|
| min price house | 5 | 8.87 | 9.129 |
| max price house | 50 | 50 | 50 |
| mean price | 22.53 | 22.53 | 23.21 |
| median price | 21.2 | 21.66 | 20.39 |
| standard deviation | 9.188 | 9.009 | 8.992 |

Looking at each row we see that the numbers are all quite close, therefore I believe it is a valid model.