# Predicting Boston Housing Prices

## §1. Statistical Analysis and Data Exploration

The Boston house prices dataset contains data for 506 houses. For each house, data is recorded for 13 separate features, in particular:

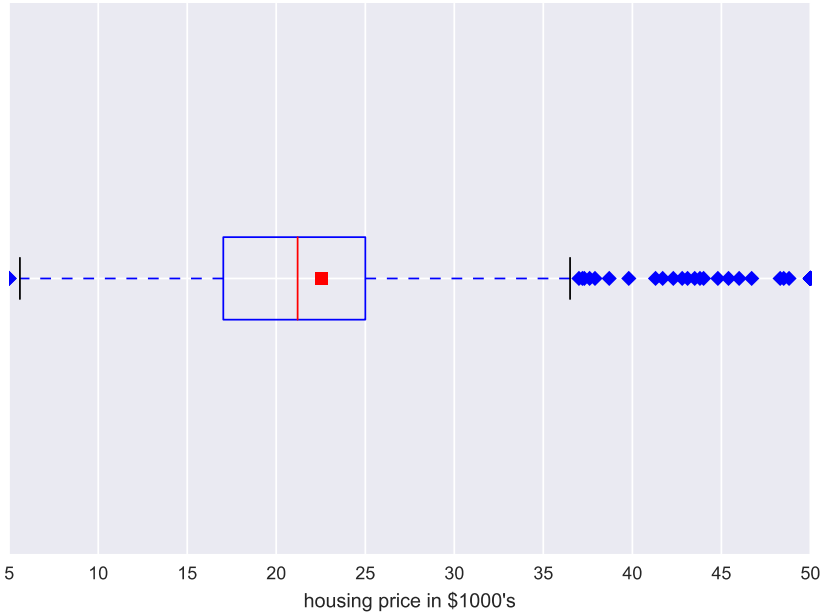
1. per capita crime rate by town
2. proportion of residential land zoned for lots over 25,000 sq.ft.
3. proportion of non-retail business acres per town
4. Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. nitric oxides concentration (parts per 10 million)
6. average number of rooms per dwelling
7. proportion of owner-occupied units built prior to 1940
8. weighted distances to five Boston employment centres
9. index of accessibility to radial highways
10. full-value property-tax rate per $10,000
11. pupil-teacher ratio by town
12. $1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by town
13. % lower status of the population

**Minimum and maximum house prices:** $5,000 and $50,000 respectively.
**Mean and median prices:** $22,533.80 and $21,200 respectively.
**Standard deviation of prices:** $9,188.

Figures 1 and 2 provide visualisations of the distribution of prices in the sample.

Figure 1: Distribution of housing prices in sample

Figure 2: Box and whisker plot of housing prices in sample




In Figure 2 the whiskers are positioned 1.5 standard deviations away from the box and the mean is displayed with a red square marker. We observe the distribution of housing prices to be positively skewed.

# §2. Evaluating Model Performance

## §2.1 Choosing a model performance metric

We are interested in predicting housing prices on a continuum so our problem is one of regression rather than classification, as such classification metrics are not suitable for this problem. In the scikit-learn libraries there are five regression metrics available, these are shown in Table 1.

**Table 1:** Scikit-learn regression metrics

| | |
|---|---|
| `explained_variance_score` | Explained variance regression score function |
| `mean_absolute_error` | Mean absolute error regression loss |
| `mean_squared_error` | Mean squared error regression loss |
| `median_absolute_error` | Median absolute error regression loss |
| `r2_score` | $R^2$ (coefficient of determination) regression score function. |

Explained variance and $R^2$, though measures of model performance, are not suitable for predicting Boston housing data and analysing the errors. In order to calibrate the model to the training data we require a metric which is appropriately related to the distance between the prediction and target and these metrics are not. In particular, we note that the distance from the target can be arbitrarily large while these metrics are bounded. We are then left with three metrics, the median and mean absolute errors and the mean squared error.

The median absolute error is not a good metric to use since increasing distance from the target need not increase the metric. In particular, when we are fitting to an array of data points, if we increase any, or even all, of the errors greater than the median error, the median does not change. The metric then considers these to be equally good fits and does not discriminate against larger errors appropriately.

The mean absolute error is also not a particularly good metric to use either since it does not distinguish between the case where we have lots of similar sized errors and the case where larger errors are offset by smaller ones. Consider the simplified case where we are fitting to two data points and as such have two errors. In the first fit the two errors are equal and in the second fit we increase one of the errors and decrease the other by the same amount. The mean error for both these fits is the same and so by this metric we do not prefer one fit over the other. From this example we can also see how, even for very simple problems, using this metric might result in solutions which are not unique.

The best choice is mean squared error. Squaring each error means we effectively penalise errors in proportion to their size. Furthermore, in the case where we are fitting a regression model which is linear in the parameters, the mean squared error is globally concave ensuring the existence of a unique solution [Least Squares Optimisation].

## §2.2 Splitting the Boston housing data into training and testing data

We split the housing data into training and testing data in order to have an independent set of data to validate our model against. Without evaluating the performance on an independent set of testing data we are at risk of overfitting our model to the training data.

## §2.3 Grid search

Grid search systematically works through every combination of specified parameter sets, fitting and scoring our model with a metric of our choice, eventually settling on the combination of parameters which result in the best score. An obvious example of where grid search is useful is when we want to find the parameters of a model for which its performance is optimal.

## §2.4 Cross validation with grid search

As mentioned above, we want to ensure that our model doesn't simply make accurate predictions on the data we trained it on but also generalises well to unseen data. In order to do this, we use cross validation. In simple cross validation we split our data into separate training and testing sets, we then check and compare the error of our models (each determined by a particular choice of model parameters) on both the training dataset and the testing dataset. The optimal model has a small error on the training data and minimal error on the testing data.

We can go further and improve on simple cross validation by using *k*-fold cross validation. Here we randomly shuffle our dataset and split it into *k* parts. We then cross validate our model *k* times, each time using a different test set. Our performance metric is then averaged over the *k* experiments. *k*-fold cross validation allows us to smooth over the variations that result from how we split our dataset for training and testing in simple cross-validation, giving us a more accurate reflection of our model performance. Unlike simple cross validation, it also enables us to use all of our data for both training and testing. This is of particular importance when data is limited in size.

A model typically has both parameters we wish to 'learn' from our training data and parameters we wish to fix (for example parameters which determine the complexity of the model). We want to choose these fixed parameters such that our testing error is minimal. Combining cross validation with grid search allows us to do this. In particular the scikit-learn function GridCrossCV combines *k*-fold cross validation (*k*=3 by default) with grid search in order to find the parameters which best generalise the model.

# §3. Analysing Model Performance

We calibrate scikit-learn's Decision Tree Regressor to our training data and compare how it performs on the training and testing data for a range of values of the max depth.

## §3.1 Learning Graphs

Figure 3: Max depth = 1

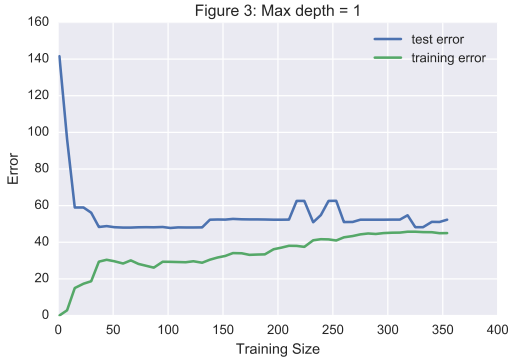



Figure 4: Max depth = 2

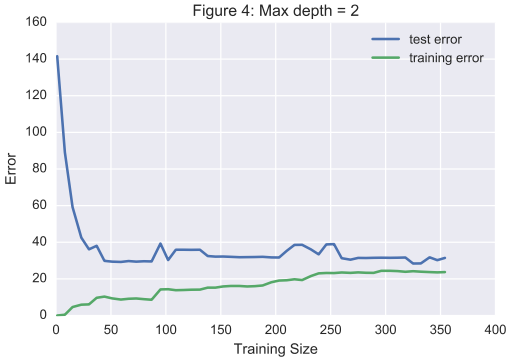



Figure 5: Max depth = 3

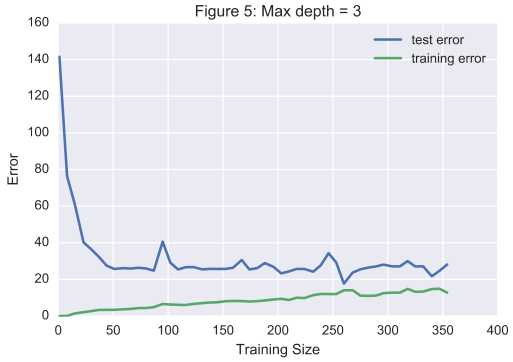



Figure 6: Max depth = 4

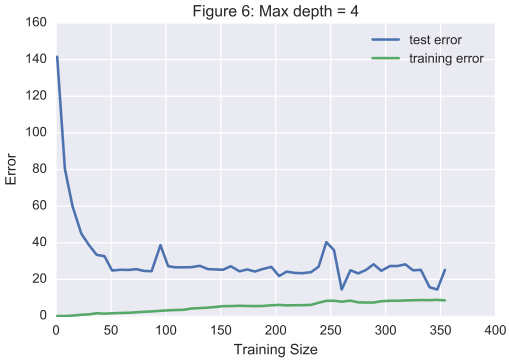
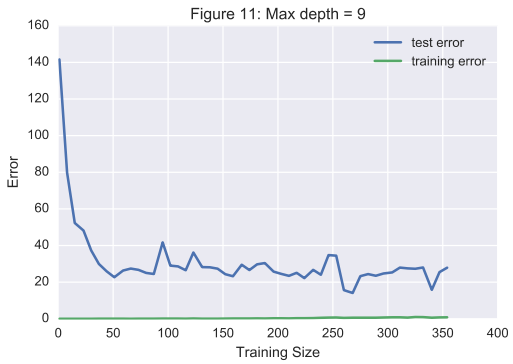


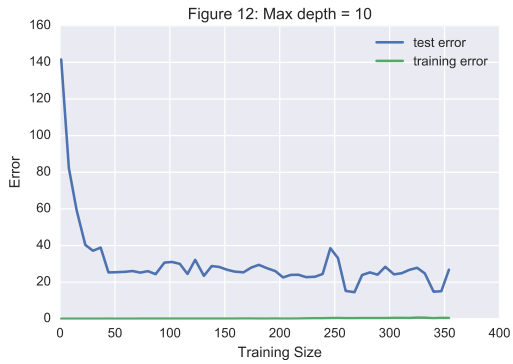

Figure 7: Max depth = 5


Figure 8: Max depth = 6


Figure 9: Max depth = 7


Figure 10: Max depth = 8




### Trend of training and testing error as training size increases

Figures 3 – 12 show that as training size increases, the general trend of training error is increasing while the testing error is generally decreasing (with noise) eventually levelling off. We note also that the training error is below the test error in all cases.

### Comparing learning curves for the decision tree regressor with max depth 1 and 10

We compare Figures 3 and 12 when fully trained. The max depth 1 regressor suffers from high bias/underfitting; the training and testing errors converge (from below and above respectively) as we increase the training data but both remain high (see Figure 3). The max depth 10 regressor suffers from high variance/overfitting; the training error increases but remains very low while the testing error decreases but high in comparison (see Figure 12).

## §3.2 Model Complexity

### Choosing max depth which best generalizes the dataset

As we increase the complexity of the model, the training error decreases steadily toward zero; the testing error however, after an initial drop, levels off. On levelling off we notice noise in the testing error but otherwise the error remains roughly constant.

Based on the complexity graph I expect the model with max depth 4 best generalises the data. This is the lowest complexity model for which the test error has fallen to the levelled off error level (which I consider to be a line through the centre of the noise). Beyond this we are increasing the complexity of the model and the cost of fitting it with no gain in the prediction performance on unseen data and – we are simply overfitting our model to the training data.

# §4. Model Prediction

## §4.1 Justifying our chosen model complexity level

The results of running the model 20 times are displayed in Table 2. We note that for 11 out of the 20 runs, max depth 4 gave the lowest mean squared error on the test data. The table also shows the results of calculating the average and standard deviation of the mean squared error over all 20 runs. We note also that the max depth 4 model gave the lowest average and standard deviation of mean squared error. This confirms max depth 4 as the best parameter choice for the model.

**Table 2:** Results over 20 iterations of 3-fold cross validation

| Max depth | Housing Price | Frequency | Average MSE | Std Dev MSE |
|---|---|---|---|---|
| 1 | $ 19,933.72 | 0 | 68.08 | 25.53 |
| 2 | $ 23,349.80 | 0 | 46.72 | 21.03 |
| 3 | $ 22,905.20 | 0 | 44.97 | 13.29 |
| 4 | $21,629.74 | 11 | 38.07 | 9.28 |
| 5 | $20,967.76 | 1 | 40.00 | 12.21 |
| 6 | $20,765.99 | 0 | 41.04 | 8.84 |
| 7 | $19,997.47 | 4 | 39.86 | 9.89 |
| 8 | $18,816.67 | 1 | 40.78 | 10.67 |
| 9 | $19,327.27 | 1 | 41.49 | 10.02 |
| 10 | $20,720 | 2 | 40.72 | 9.87 |

## §4.2 Sanity checking our prediction result

The price predicted by the max depth 4 model is $21,629.74. This price falls between the 52.4th and 52.6$^{th}$ percentile of our sample distribution. The price is between the mean and median prices and if we look at our histogram in Figure 1 we can see it falls in the range of the mode. Based on this, the price produced by our chosen model seems plausible.

Using the utilities in the scikit-learn neighbours module we find the ten nearest neighbours, by features, to our clients house. The mean and standard deviation of the prices of these 10 houses are $21,520 and $10,300 respectively. We note that our clients predicted house price is very close to this mean, and well within the one standard deviation, further validating our model's predicted price.