Predicting Boston Housing Prices

Table of Contents

Project Aim

Apply basic machine learning concepts on data collected for housing prices in the Boston, Massachusetts area to predict the selling price of a new home.

Methodology

- Explore the data to obtain important features and descriptive statistics about the dataset.
- Split the data into testing and training subsets, and determine a suitable performance metric for this problem.
- Analyse performance graphs for a learning algorithm with varying parameters and training set sizes to pick the optimal model that best generalizes for unseen data.
- Test this optimal model on a new sample and compare the predicted selling price to your statistics.

Data Source

- https://archive.ics.uci.edu/ml/datasets/Housing

Software and Libraries

- Python 2.7
- NumPy 1.10
- scikit-learn 0.17
- iPython Notebook (with iPython 4.0)

Statistical Analysis and Data Exploration

Data points (houses) collected	506
Number of features are present for each house	13
Minimum housing price	5
Maximum housing price	50
Mean housing price	22.533
Median housing price	21.2
Standard deviation of house price	9.188

Description of three significant features

There are 506 houses in this dataset. The median price is USD21,200. The median price refers to the price at which half house prices are more than USD21,200 and half are less USD21,200. The standard deviation is USD9,188. Assuming that the house prices follow a normal distribution (of mean = 0 and Variance =1), 68% of the house prices will be contained between USD13,345 and USD31,721 (i.e. avg= USD22,533 +/- USD9,188). 95% of the house prices will be contained within 2 standard deviations from the mean. This is USD4,157 and USD40,909 (i.e. avg= USD22,533 +/- (2*USD9,188)).

Listing of the values corresponding to the above features, for the client's feature set CLIENT_FEATURES

CRIM corresponds to the value 11.95. RM corresponds to the value 5.609. LSTAT corresponds to the value 12.13.

Evaluating Model Performance

Discussion of the purpose of splitting the data into training and testing subsets

To give an estimate of the performance on an independent dataset and serves as check on overfitting.

Listing of the most appropriate performance measures for predicting housing prices and analyzing error

- Accuracy
- Precision
- Recall
- F1 score
- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)

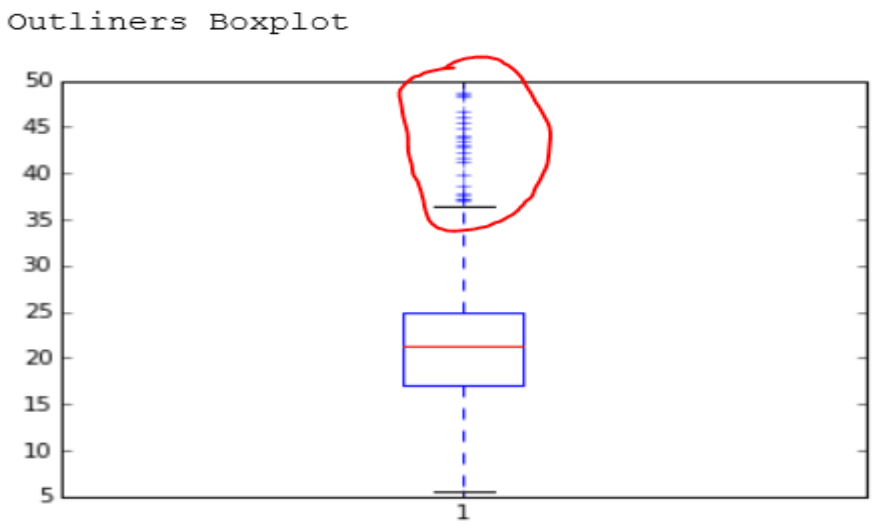
The following scoring methods are for classification tasks only. Therefore, they do not fit the bill in the regression case. When we try to run the below metrics, a runtime throws an exception message: 'continuous is not supported'

- accuracy_score(y_true,y_predict)
- precision_score(y_true,y_predict)
- recall_score(y_true,y_predict)
- f1_score(y_true, y_predict, average='weighted')

The only two metrics that are compatible with the regression analysis are the MAE and MSE.

- MAE favours general accuracy at the expense of occasional wide misses
- MSE favours less overall accuracy at the price to reduce the likelihood/size of wide misses

In other words, MSE is more sensitive to the outliers as it emphasises the large errors by taking square of the error. As shown by the boxplot below, there are a large number of outliners; therefore MAE is preferred over MSE.



Explanation of the grid search algorithm and its applicability

Grid search algorithm[1] is a way of systematically working through multiple combinations of parameter tunes, cross-validating as it goes to determine which tune gives the best performance.
Grid search allows the following to be specified:

- the parameters list
- the ranges that should be explored for each parameter
- the scoring function that needs optimizing.

It then iteratively searches the combinations of those parameters and gives us the values of the parameters that optimize the scoring function.

Explanation of cross-validation and how is it performed on a model

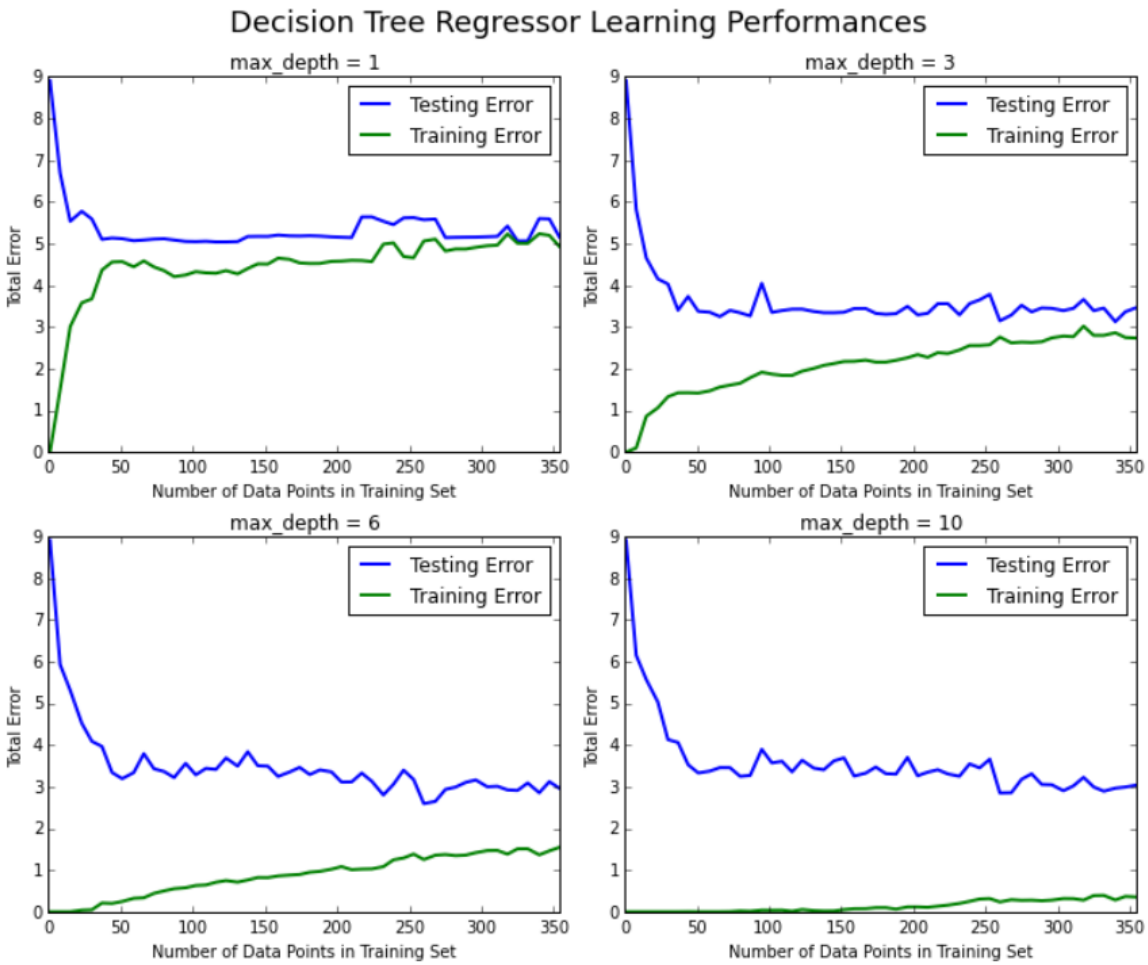
The point of cross-validation is to avoid supervised machine learning algorithms to training and test on the same dataset. The point is to retain a portion of the dataset as the training set, and the rest as the test set. Cross-validation enables to maximise the usage of both the training set and the test set, and ensure the entire dataset is used through the training and test phase.
The methodology is as follows:

- Split linearly the dataset in buckets by k factor (e.g. k=10, hence 10 buckets)
- Decide the percentage of buckets that are used for training (e.g. 70%). The rest is used as test data (i.e. 30% in this example)
- Iteratively use each of the training bucket and remaining test bucket as input data in the machine learning algorithm
- Finally, average the results

The grid search specialises in tuning the machine learning input parameters. By feeding all the dataset in the training set (eventually), the cross-validation helps with the tuning of these parameters. Conversely, if a section of the full dataset was only used for testing purpose, there is a chance the shape of the test data would differ from the training data. The grid search would have no opportunity to learn from the test set, and improve on the parameters tuning. Overall, Grid Search with Cross-Validation Scheme would make the brute force parameter search better in the way that each model would be trained with best learning result and best validation.

[1] There are other techniques that could be used for hyperparameter optimization in order to save time, such as RandomizedSearchCV (http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.RandomizedSearchCV.html#sklearn.grid_search.RandomizedSearchCV). In this case, instead of exploring the whole parameter space, just a fixed number of parameter settings is sampled from the specified distributions. This proves useful when we need to save time but is not necessary in cases in cases like ours where the data set is relatively small.

Analysing Model Performance



Graph Description 1

The chosen learning curve graph is one where max_depth = 4.

Training Errors: There is a sharp increase (approx. 100%) of the number of errors when the number of data points increases from 0 to approx. 50. From this point on, the number of errors remains relatively stable around the 4 and 5 marks.

Testing Errors: There is a sharp decrease (approx. 55%) of the number of errors when the number of data points increases from 0 to approx. 40. From approx. 40 to approx. 200 data points the number of errors remains stable at around 5. There is a small bit in the error count between approx. 200 and aprrox. 260 data point to 6. After that, it stays around the 5 mark on average.

Graph Description 2

Looking at the test errors where max depth = 1 and the model is using the full training set, both the number of test and training errors meet at same level of error, i.e. 5. This would indicate model suffer from a high bias and a low variance. Conversely, when max depth = 10, the training error is close to 0, while the testing error is close to 3. This would indicate a small bias and a high variance.

9) From the model complexity graph, describe the training and testing errors as the max depth increases. Based on your interpretation of the graph, which max depth results in a model that best generalizes the dataset? Why?

As a max depth increase, the training error curve displays a strong negative convexity, starting at 5 error for max depth = 1 and ending with 0 errors when max depth = 13. The testing error curve does not show the same curvature. There is a sharp decrease of errors until max depth = 5 (from 5 to 3 errors), then it plateaus at approx 3. The optimum max depth is where both the test and training errors attain their respective minimum. Therefore, max depth = 12 looks like a good number, the number of test error is approx 3. and the number of training errors is close to 0.

The hashed area in red shows overfitting, as the training errors go down to zero with max depth = 12 while the training test total errors stay around 3. The yellow crossing axis show the potential optimum point for the maximum depth (i.e. 4) where Testing errors are at their global minimum, i.e. 3.

Model Prediction

Display of the optimal max depth for the model using grid search

The max depth provided is 4.

Best selling price for the client's home / Comparison of the best selling price to the statistics generated on the dataset

Using the parameter-tuned model, it looks like the best predicted client's home value is USD21,630. It lives within one standard deviation (USD9,188) from the mean (USD22,533).[2]

Discussion of the model relevance

As the predicted value is very close to the current mean, it looks like the predicted model is a good fit to predict future selling prices in this area.

[2] To assess if your prediction is reasonable, besides from comparing it with the median, the mean and checking if it is included in one standard deviation range, to use SKlearn to find the nearest neighbours of the feature vector and see how your result compares with them. Please see http://scikit-learn.org/stable/modules/neighbors.html#finding-the-nearest-neighbors for further details.

```
from sklearn.neighbors import NearestNeighbors
def find_nearest_neighbor_indexes(x, X):  # x is your vector and X is the data set.
  neigh = NearestNeighbors( n_neighbors = 10 )
  neigh.fit( X)
  distance, indexes = neigh.kneighbors( x )
  return indexes
indexes = find_nearest_neighbor_indexes(x, X)
sum_prices = []
for i in indexes:
   sum_prices.append(city_data.target[i])
neighbor_avg = np.mean(sum_prices)
print "Nearest Neighbors average: " +str(neighbor_avg)
```