Project Report: Predicting Boston Housing Prices

Aravind Battaje

January 25, 2016

1 Project Steps

The project aims to predict housing prices by using regression learning on some known data. On a comprehensive dataset provided by `scikit-learn`, preliminary statistical analysis is performed to find the nature of data/features. The dataset is split after shuffling into two sets for training and testing. The training set is used with a decision tree learner at varying tree depths. For each depth, training and testing errors as a function of training set size are computed and graphed. Another graph to assess model complexity (depth of decision tree) in relation to training and testing errors is also computed. The relation of training size with errors and the approximate optimal tree depth is observed from the graph plots. Finally, a cross-validated grid search is used to find the optimal decision tree depth. This is used to predict the house price for a user-input data.

2 Statistical Analysis and Data Exploration

2.1 Dataset

`scikit-learn`'s Boston housing dataset possesses following characteristics:

Number of features		13
Number of data points		506
Feature descriptions	CRIM	per capita crime rate by town
	ZN	proportion of residential land zoned for lots over 25,000 sq.ft.
	INDUS	proportion of non-retail business acres per town
	CHAS	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
	NOX	nitric oxides concentration (parts per 10 million)
	RM	average number of rooms per dwelling
	AGE	proportion of owner-occupied units built prior to 1940
	DIS	weighted distances to five Boston employment centers
	RAD	index of accessibility to radial highways
	TAX	full-value property-tax rate per $10,000
	PTRATIO	pupil-teacher ratio by town
	B	$1000(B_k - 0.63)^2$ where B_k is the proportion of blacks by town
	LSTAT	% lower status of the population
Target description		median value of owner-occupied homes in $1000s

The above information can be accessed from the `.DESCR` member of Boston dataset object.

2.2 Analysed Characteristics

From the dataset loaded, the minimum, maximum, mean, median and standard deviation values of each feature including that of target were found using relevant *numpy* methods. Table 1 is derived from the analyzed

data. Note for instance from **TARGET** column that the mean of target house prices is 22.53(×$1000).

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS
MIN	0.00632	0	0.46	0	0.385	3.561	2.9	1.1296
MAX	88.9762	100	27.74	1	0.871	8.78	100	12.1265
AVG	3.59376	11.3636	11.1368	0.06917	0.554695	6.28463	68.5749	3.79504
MED	0.25651	0	9.69	0	0.538	6.2085	77.5	3.20745
STD	8.58828	23.2994	6.85357	0.253743	0.115763	0.701923	28.121	2.10363

	RAD	TAX	PTRATIO	B	LSTAT	**TARGET**
MIN	1	187	12.6	0.32	1.73	**5**
MAX	24	711	22	396.9	37.97	**50**
AVG	9.54941	408.237	18.4555	356.674	12.6531	**22.5328**
MED	5	330	19.05	391.44	11.36	**21.2**
STD	8.69865	168.37	2.16281	91.2046	7.134	**9.18801**

Table 1: Boston housing prices dataset analysis

3 Evaluating Model Performance

3.1 Performance Metric

`scikit-learn` provides five different metrics for assessing model performance, namely, `mean_squared_error`, `mean_absolute_error`, `median_absolute_error`, `explained_variance_score` and `r2_score`. `r2_score` and `explained_variance_score` are *score* metrics (greater the better, unlike *error* metric) and they help to assess how well the (training) data has been fit on the regressor. This property might not fair well against finding score of testing data and might lead to over-fitting. `mean_squared_error, mean_absolute_error` and `median_absolute_error` are simpler metrics and they should fair better against over-fitting. Amongst the *error* metrics, `median_absolute_error` and `mean_absolute_error` are closely related, but differ in that the latter is not robust against outliers. Outliers are those data points which are *outside* of the general grouping/trend of a particular characteristic (here prediction error), and they are caused due to errors in measurements. With presence of outliers, `mean_absolute_error` is susceptible to be *pulled* away from the true (without outliers) mean, whereas `median_absolute_error` that picks the *median* from the distribution of errors, is mildly affected as outliers contribute much lesser to the *middle* value of a distribution. `mean_squared_error` behaves similar to `mean_absolute_error` with presence of outliers, but the error is often amplified because of the squaring. The best performance metric to use thus in this project must be `median_absolute_error`.

3.2 Dataset Split

The Boston housing data is split into two sets for training and testing purposes. The training set is used during the learning/fitting and the testing set is used to find the (true) prediction deviation from target values. If there is no split in data, and the learner is modeled only on the given dataset, there are chances of over-fitting as no performance measure can be pegged against *new* data. Effectively, by splitting the data and performing the test step, it is ensured that the model is generalized on the data. The over-fitting for instance is observable in Figure 1, where predicting on training set, which is when the regressor hasn't seen any *new* (or test) data, the training error is very low for any training size. Also, the ratio of split plays a role in model performance. If the training data size is lower than test data size, *optimum* learning might not be possible. It is also bad to have a very small test data size as that might lead to over-fitting. For the project, optimum `TrainSize:TestSize` ratio could be between 50% to 80%.

Figure 1: Decision tree of max. depth = 9

3.3 Grid Search and Cross-Validation

`scikit-learn`'s `GridSearchCV` helps perform an exhaustive grid search on the *parameter grid* of an *estimator*. In the project, a regression based *estimator* (`DecisionTreeRegressor`) is used. `DecisionTreeRegressor` has a parameter called `maxdepth` that controls the maximum allowed tree depth. Given a parameter space, grid search exhaustively evaluates the *estimator* performance for each parameter (here only `maxdepth`). This helps to find the best model to fit the data.

Additionally, `GridSearchCV` uses cross-validation instead of train-test split to learn and evaluate the model performance. Cross-validation is a procedure where the input dataset is split into smaller chunks, followed by learning on most of the chunks, holding out only a few for evaluation. In the most basic form (used here), called *K-Fold* cross-validation, data is split into K *folds*. $K-1$ folds are used for learning and the left out fold is used to evaluate model performance. With several iterations, the fold held out is made to cycle through all available K folds. The total performance measure will be the average of performace metric calculated in each iteration. In the project, a *3-Fold* (default) cross-validated grid search is used, which is also optimum. Higher folds might result in smaller validation set, effectively over-fitting. Anything more than *10-Fold* cross-validation with Boston housing dataset might not be prudent because of the small size of data (506).

4 Analyzing Model Performance

4.1 Learning Curves

In the project, ten training curves are generated with ten different *max depths*. In each of the graphs, the training and testing errors as a function of training size (50 increments) is plotted. The ten plots give an insight into the nature of *learning* with respect to the training size and model complexity (max depth). Generally, it is seen that with increasing training size, training error consistently increases and testing error consistently decreases. With higher model complexity, it is seen that there is a general shift in both error curves towards lesser error values. To pick on specific details, extremes in the series of plots, namely, Figure 2a (max depth 1) and Figure 2b (max depth 10) provide reasonable insights. In Figure 2a, it is quite apparent that the *learning* by the decision tree is insufficient because training and testing errors are quite high and they converge as the training size increases. On the other hand, Figure 2b represents another extreme where

(a) Depth 1

(b) Depth 10

Figure 2: Decision tree learning curves

with sufficient training sizes, testing error decreases, but more importantly, the training error is very close to zero. This indicates that the decision tree has fit the training data quite well (maybe too well, or over-fit).

Some interesting (side) notes for both plots: Rising training data size and rising max depths, result in lesser error with testing data; the curves are quite abrupt in Figure 2a because of the terrible under-fitting; the training errors are very low when the training size is too low as a consequence of over-fitting on the very small size of data.

4.2 Model Complexity

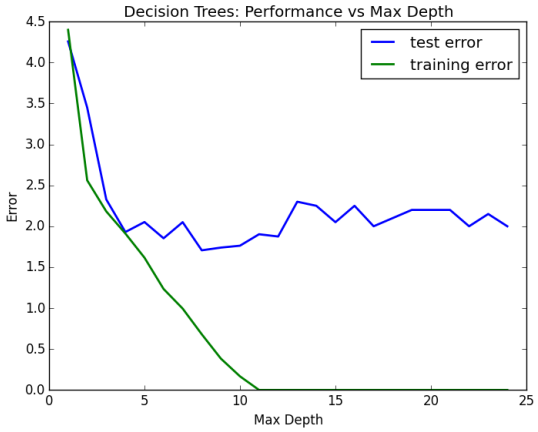

Figure 3: Model complexity graph

The model complexity graph (Figure 3) represents the relation between learning errors and complexity of the decision tree (or max depth). The same data for training and testing as used for the above mentioned learning curves are used here. Upto a certain point (max depth = 10) each learning curves' training and testing errors at a particular (highest) training size relate to each error point in the model complexity graph. Therefore, as it is observed in learning curves, model complexity graph confirms the fact that the training error decreases and tends to zero with higher max depths as a consequence of over-fitting, and that with higher max depths the testing error seems to decrease. However, it is important to note in the model complexity graph that the testing error seems to saturate beyond a certain max depth. More precisely, the

testing error tends to stay around a neighborhood of a minimum, beyond a certain *point* (of generalization). The slight increases in testing error beyond this *point* can be attributed to over-fitting. This could also indicate that with any given set of training data, it is not prudent to achieve a perfect fit without trading off on generalization. Thus, in Figure 3, as the saturation of testing error starts at around a max depth of 5, and training error starts to get too low beyond max depth of 7, the optimum model to select for generalization would be a decision tree of max depth between 5 and 7.

5 Model Prediction

After running 100 trials, the most common model was found to be a decision tree of max depth 6. The prediction of this tree for the sample input `[11.95, 0.00, 18.100, 0, 0.6590, 5.6090, 90.00, 1.385, 24, 680.0, 20.20, 332.09, 12.13]` is `20.76598639`. The average of the predictions, with max depth mostly varying between 5 to 7, was found to be `20.71604903` with a standard deviation of `0.33793076`. It appears that the most common prediction `20.76598639` approximately agrees with the statistics previously derived. For instance, comparing some of the features of the sample input, such as `PTRATIO` and `RM`, against those of statististics, the prediction appears to be consistent. In other words, considering `PTRATIO` (pupil-teacher ratio) and `RM` (average number of rooms per dwelling), which have the least variance among the dataset, and are also slightly lesser than the respective means, it appears natural that the prediction also should be lesser than the average. By this logic, the prediction falls well within the *majority of distribution* of housing prices in the dataset.