boston_housing

May 28, 2018

0.1 Project: Predicting Boston Housing Prices

Welcome to the first project of the Machine Learning Engineer Nanodegree! In this notebook, some template code has already been provided for you, and you will need to implement additional functionality to successfully complete this project. You will not need to modify the included code beyond what is requested. Sections that begin with **'Implementation'** in the header indicate that the following block of code will require additional functionality which you must provide. Instructions will be provided for each section and the specifics of the implementation are marked in the code block with a 'TODO' statement. Please be sure to read the instructions carefully!

In addition to implementing code, there will be questions that you must answer which relate to the project and your implementation. Each section where you will answer a question is preceded by a **'Question X'** header. Carefully read each question and provide thorough answers in the following text boxes that begin with **'Answer:'**. Your project submission will be evaluated based on your answers to each of the questions and the implementation you provide.

> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. In addition, Markdown cells can be edited by typically double-clicking the cell to enter edit mode.

0.2 Getting Started

In this project, you will evaluate the performance and predictive power of a model that has been trained and tested on data collected from homes in suburbs of Boston, Massachusetts. A model trained on this data that is seen as a *good fit* could then be used to make certain predictions about a home — in particular, its monetary value. This model would prove to be invaluable for someone like a real estate agent who could make use of such information on a daily basis.

The dataset for this project originates from the UCI Machine Learning Repository. The Boston housing data was collected in 1978 and each of the 506 entries represent aggregated data about 14 features for homes from various suburbs in Boston, Massachusetts. For the purposes of this project, the following preprocessing steps have been made to the dataset: - 16 data points have an `'MEDV'` value of 50.0. These data points likely contain **missing or censored values** and have been removed. - 1 data point has an `'RM'` value of 8.78. This data point can be considered an **outlier** and has been removed. - The features `'RM'`, `'LSTAT'`, `'PTRATIO'`, and `'MEDV'` are essential. The remaining **non-relevant features** have been excluded. - The feature `'MEDV'` has been **multiplicatively scaled** to account for 35 years of market inflation.

Run the code cell below to load the Boston housing dataset, along with a few of the necessary Python libraries required for this project. You will know the dataset loaded successfully if the size of the dataset is reported.

```
In [28]: # Import libraries necessary for this project
         import numpy as np
         import pandas as pd
         from sklearn.cross_validation import ShuffleSplit

         # Import supplementary visualizations code visuals.py
         import visuals as vs

         # Pretty display for notebooks
         %matplotlib inline

         # Load the Boston housing dataset
         data = pd.read_csv('housing.csv')
         prices = data['MEDV']
         features = data.drop('MEDV', axis = 1)

         # Success
         print("Boston housing dataset has {} data points with {} variables each.".format(*data.
```

```
Boston housing dataset has 489 data points with 4 variables each.
```

0.3 Data Exploration

In this first section of this project, you will make a cursory investigation about the Boston housing data and provide your observations. Familiarizing yourself with the data through an explorative process is a fundamental practice to help you better understand and justify your results.

Since the main goal of this project is to construct a working model which has the capability of predicting the value of houses, we will need to separate the dataset into **features** and the **target variable**. The **features**, `'RM'`, `'LSTAT'`, and `'PTRATIO'`, give us quantitative information about each data point. The **target variable**, `'MEDV'`, will be the variable we seek to predict. These are stored in `features` and `prices`, respectively.

0.3.1 Implementation: Calculate Statistics

For your very first coding implementation, you will calculate descriptive statistics about the Boston housing prices. Since `numpy` has already been imported for you, use this library to perform the necessary calculations. These statistics will be extremely important later on to analyze various prediction results from the constructed model.

In the code cell below, you will need to implement the following: - Calculate the minimum, maximum, mean, median, and standard deviation of `'MEDV'`, which is stored in `prices`. - Store each calculation in their respective variable.

```
In [29]: # Converting to numpy array for easy operations
         prices_np = np.array(prices)

         # Minimum price of the data
         minimum_price = prices_np.min()
```

```
    # Maximum price of the data
    maximum_price = prices_np.max()

    # Mean price of the data
    mean_price = prices_np.mean()

    # Median price of the data
    median_price = np.median(prices)

    # Standard deviation of prices of the data
    std_price = prices_np.std()

    # Show the calculated statistics
    print("Statistics for Boston housing dataset:\n")
    print("Minimum price: ${}".format(minimum_price))
    print("Maximum price: ${}".format(maximum_price))
    print("Mean price: ${}".format(mean_price))
    print("Median price ${}".format(median_price))
    print("Standard deviation of prices: ${}".format(std_price))
```

```
Statistics for Boston housing dataset:

Minimum price: $105000.0
Maximum price: $1024800.0
Mean price: $454342.9447852761
Median price $438900.0
Standard deviation of prices: $165171.13154429474
```

0.3.2 Question 1 - Feature Observation

As a reminder, we are using three features from the Boston housing dataset: `'RM'`, `'LSTAT'`, and `'PTRATIO'`. For each data point (neighborhood): - `'RM'` is the average number of rooms among homes in the neighborhood. - `'LSTAT'` is the percentage of homeowners in the neighborhood considered "lower class" (working poor). - `'PTRATIO'` is the ratio of students to teachers in primary and secondary schools in the neighborhood.

** Using your intuition, for each of the three features above, do you think that an increase in the value of that feature would lead to an **increase** in the value of `'MEDV'` or a **decrease** in the value of `'MEDV'`? Justify your answer for each.**

Answer:

1 – Would you expect a home that has an 'RM' value(number of rooms) of 6 be worth more or less than a home that has an 'RM' value of 7?

Ans : A home with greater value of RM (number of rooms) will increase the price (MEDV) of that home. A home with 7 rooms will have higher price than those with 6 rooms.

2 –Would you expect a neighborhood that has an 'LSTAT' value(percent of lower class workers) of 15 have home prices be worth more or less than a neighborhood that has an 'LSTAT' value of 20?

Ans : The neighborhood with 15 'LSTAT' value will be worth more than the neighborhoods with 20 'LSTAT' values. A neighborhood with greater value of LSTAT (lower class workers) will decrease the price (MEDV) of homes in that neighborhood.

3 –Would you expect a neighborhood that has an 'PTRATIO' value(ratio of students to teachers) of 10 have home prices be worth more or less than a neighborhood that has an 'PTRATIO' value of 15

Ans : The neighborhood who has a 'PTRATIO' value less will be worth more than those who has a high 'PTRATIO' value. So , the neighborhood with 'PTRATIO' of 10 is worth than 'PTRATIO' of 15. People with low income or under age of 25 years may prefer a young neighborhood.

0.4 Developing a Model

In this second section of the project, you will develop the tools and techniques necessary for a model to make a prediction. Being able to make accurate evaluations of each model's performance through the use of these tools and techniques helps to greatly reinforce the confidence in your predictions.

0.4.1 Implementation: Define a Performance Metric

It is difficult to measure the quality of a given model without quantifying its performance over training and testing. This is typically done using some type of performance metric, whether it is through calculating some type of error, the goodness of fit, or some other useful measurement. For this project, you will be calculating the *coefficient of determination*, R2, to quantify your model's performance. The coefficient of determination for a model is a useful statistic in regression analysis, as it often describes how "good" that model is at making predictions.

The values for R2 range from 0 to 1, which captures the percentage of squared correlation between the predicted and actual values of the **target variable**. A model with an R2 of 0 is no better than a model that always predicts the *mean* of the target variable, whereas a model with an R2 of 1 perfectly predicts the target variable. Any value between 0 and 1 indicates what percentage of the target variable, using this model, can be explained by the **features**. *A model can be given a negative R2 as well, which indicates that the model is* ***arbitrarily worse*** *than one that always predicts the mean of the target variable.*

```
In [30]: # TODO : Import 'r2_score'
         from sklearn.metrics import r2_score

         def performance_metric(y_true, y_predict):
             """ Calculates and returns the performance score between
                 true and predicted values based on the metric chosen. """

             # TODO: Calculate the performance score between 'y_true' and 'y_predict'
             score = r2_score(y_true, y_predict)

             # Return the score
             return score
```

0.4.2 Question 2 - Goodness of Fit

Assume that a dataset contains five data points and a model made the following predictions for the target variable:

True Value	Prediction
3.0	2.5
-0.5	0.0
2.0	2.1
7.0	7.8
4.2	5.3

Run the code cell below to use the `performance_metric` function and calculate this model's coefficient of determination.

```
In [31]: # Calculate the performance of this model
         score = performance_metric([3, -0.5, 2, 7, 4.2], [2.5, 0.0, 2.1, 7.8, 5.3])
         print("Model has a coefficient of determination, R^2, of {:.3f}.".format(score))
```

```
Model has a coefficient of determination, R^2, of 0.923.
```

- Would you consider this model to have successfully captured the variation of the target variable?
- Why or why not?

Answer:
The coefficient of determination R^2 is the proportion of the variance in the dependent variable that is predictable from the independent variable In the above case, the model has a coefficient of determination, R^2 of 0.923
Therefore, the model has successfully captured the variation of the target variable, The model has captured 92% variation in the target variable and has a fairly strong correlation

0.4.3 Implementation: Shuffle and Split Data

Your next implementation requires that you take the Boston housing dataset and split the data into training and testing subsets. Typically, the data is also shuffled into a random order when creating the training and testing subsets to remove any bias in the ordering of the dataset.

```
In [32]: # TODO: Import 'train_test_split'
         from sklearn.cross_validation import train_test_split

         # TODO: Shuffle and split the data into training and testing subsets
         X_train, X_test, y_train, y_test = train_test_split(features, prices, test_size = 0.2,

         # Success
         print("Training and testing split was successful.")
```

```
Training and testing split was successful.
```

0.4.4 Question 3 - Training and Testing

- What is the benefit to splitting a dataset into some ratio of training and testing subsets for a learning algorithm?

Answer:

When we evaluate the model on the unseen data, we would need the true values for that unseen data for us to compare the model predictions with. A model which does great in training set will end up doing worse in testing set and the model which does a little good in training set ends up doing better in testing set, this gives us less error margins and thus it proves that this model is better. If we train the model on the whole available data set, it would memorize all the true values and will have a perfect accuracy and we will not be able to check its performance on unseen data set.

Splitting off the dataset allow us to train our model on one subset and validate the model on another subset.

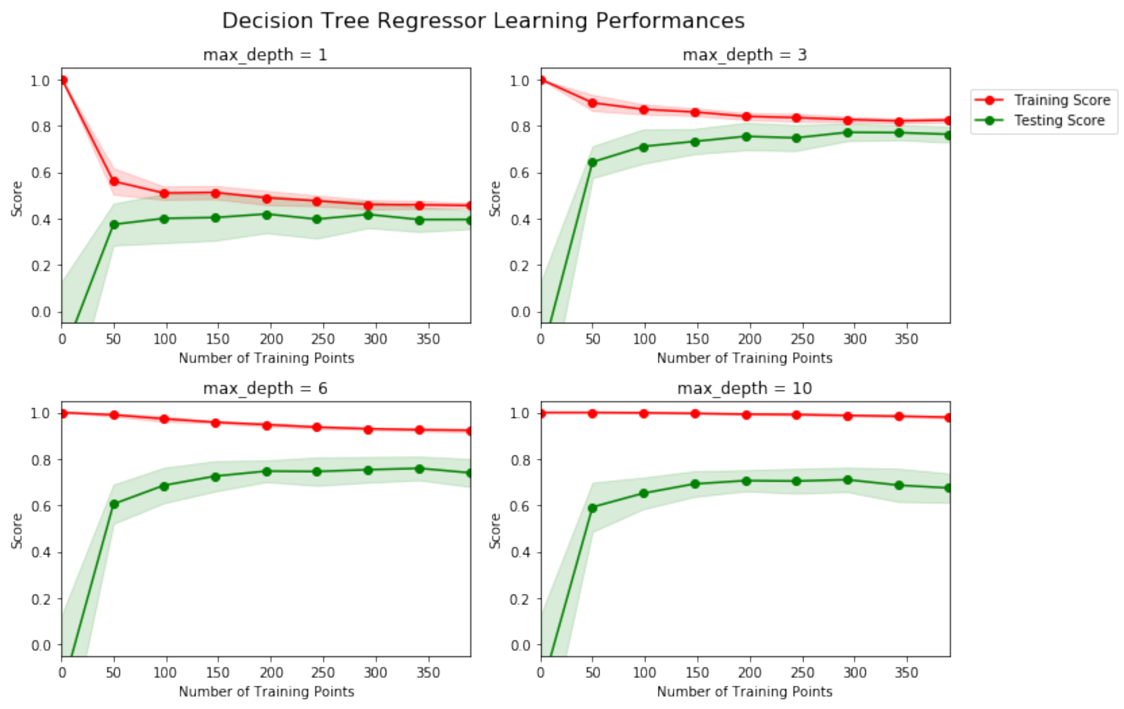
0.5 Analyzing Model Performance

In this third section of the project, you'll take a look at several models' learning and testing performances on various subsets of training data. Additionally, you'll investigate one particular algorithm with an increasing `'max_depth'` parameter on the full training set to observe how model complexity affects performance. Graphing your model's performance based on varying criteria can be beneficial in the analysis process, such as visualizing behavior that may not have been apparent from the results alone.

0.5.1 Learning Curves

The following code cell produces four graphs for a decision tree model with different maximum depths. Each graph visualizes the learning curves of the model for both training and testing as the size of the training set is increased. Note that the shaded region of a learning curve denotes the uncertainty of that curve (measured as the standard deviation). The model is scored on both the training and testing sets using R2, the coefficient of determination.

```
In [33]: # Produce learning curves for varying training set sizes and maximum depths
         vs.ModelLearning(features, prices)
```



0.5.2 Question 4 - Learning the Data

- Choose one of the graphs above and state the maximum depth for the model.
- What happens to the score of the training curve as more training points are added? What about the testing curve?
- Would having more training points benefit the model?

Answer:

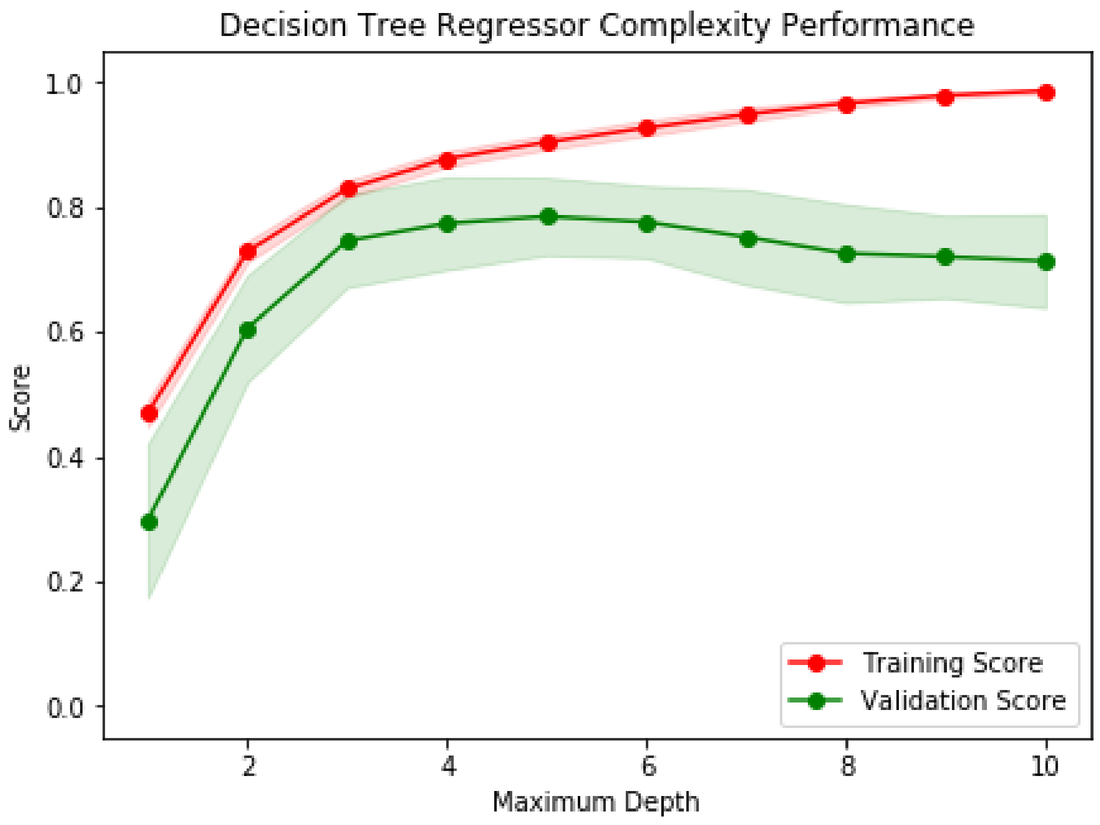
I am choosing the first graph with max depth = 1

We can see that, as more training points are being added, The score of the training curve decreases because the model becomes harder to fit into a larger sample. The score of the testing curve increases, but it seems to level off around 350 points because the model reaches its maximum effectiveness. Since the testing curve level off around 350 points,adding more training points would not affect the model further in any case.

0.5.3 Complexity Curves

The following code cell produces a graph for a decision tree model that has been trained and validated on the training data using different maximum depths. The graph produces two complexity curves — one for training and one for validation. Similar to the **learning curves**, the shaded regions of both the complexity curves denote the uncertainty in those curves, and the model is scored on both the training and validation sets using the `performance_metric` function.

```
In [34]: vs.ModelComplexity(X_train, y_train)
```



0.5.4 Question 5 - Bias-Variance Tradeoff

- When the model is trained with a maximum depth of 1, does the model suffer from high bias or from high variance?
- How about when the model is trained with a maximum depth of 10? What visual cues in the graph justify your conclusions?

Answer:
When the model is trained with max depth of 1, we observe an underfitting, hence it suffers from High Bias. The two lines are very close together but the overall the performance is very poor for both training and testing data which is a good indicator of high bias.
When the model is trained with max depth of 10, we observe an overfitting, hence it suffers from High Variance. The size of the gap between the training score and the validation score lines has a significant amount of gap between the two that serves as a visual cue to justify this conclusion.

0.5.5 Question 6 - Best-Guess Optimal Model

- Which maximum depth do you think results in a model that best generalizes to unseen data?
- What intuition lead you to this answer?

Answer:
The model with maximum depth of **4** has the highest score for the model's ability to generalize unseen data. From the graph we observe, minimum distance between points, of the two curves, is at max depth = 4

0.6 Evaluating Model Performance

In this final section of the project, you will construct a model and make a prediction on the client's feature set using an optimized model from `fit_model`.

0.6.1 Question 7 - Grid Search

- What is the grid search technique?
- How it can be applied to optimize a learning algorithm?

Answer:
Grid search is a technique to find good values for the model parameter. This works by defining a grid over the model parameters and then evaluating model performances for each point on the grid (using validation set instead of training set). We can then choose the point that works the best. It also uses brute force to create models for different combination of parameters, and is typically performed alongside with cross-validation to determine which set of parameters gives the best performance through any kind of scoring metric. It is also called hyperparameter optimization technique that can further help to improve the performance of a model by finding the optimal combination of hyperparameter values.

As we have already applied above for the complexity curve, In that task, we did not know which value of max_depth is best for the decision tree applied to the Boston housing dataset. We defined a grid over the max_depth values 1 to 10 and evaluated decision tree performance for each value from 1 to 10. This allowed us to estimate the relationship between max_depth and the scores.

0.6.2 Question 8 - Cross-Validation

- What is the k-fold cross-validation training technique?
- What benefit does this technique provide for grid search when optimizing a model?

Answer:
K-fold cross validation technique is a measure taken to minimize the error while training our dataset. It basically divides the dataset into n random bins, and then the testing is done on one of the n bins(selected randomly) while other are part of the training set. This process of testing and training is repeated n times to cover all the bins. The final answer consist of the average of the n output from all the n bins.

In this technique we randomly break the data into small sets, called k-buckets. Here 'k' denotes the number of sets we make from the original dataset.

1. We break the data into k-buckets.
2. Then we train our model k times, each time using a different bucket (as our training set)

3. Then we take average of the results to get the final

Benefits - This technique is used to avoid overfitting, increasing algorithm's accuracy in classifying any new previously unknown data point. We are able to use all data for training and testing which is useful for smaller sample sizes. The same data will also never be reused for testing unlike other cross-validation techniques

0.6.3 Implementation: Fitting a Model

Your final implementation requires that you bring everything together and train a model using the **decision tree algorithm**. To ensure that you are producing an optimized model, you will train the model using the grid search technique to optimize the `'max_depth'` parameter for the decision tree. The `'max_depth'` parameter can be thought of as how many questions the decision tree algorithm is allowed to ask about the data before making a prediction. Decision trees are part of a class of algorithms called *supervised learning algorithms*.

In addition, you will find your implementation is using `ShuffleSplit()` for an alternative form of cross-validation (see the `'cv_sets'` variable). While it is not the K-Fold cross-validation technique you describe in **Question 8**, this type of cross-validation technique is just as useful!. The `ShuffleSplit()` implementation below will create 10 (`'n_splits'`) shuffled sets, and for each shuffle, 20% (`'test_size'`) of the data will be used as the *validation set*. While you're working on your implementation, think about the contrasts and similarities it has to the K-fold cross-validation technique.

```
In [35]: # TODO: Import 'make_scorer', 'DecisionTreeRegressor', and 'GridSearchCV'
         from sklearn.tree import DecisionTreeRegressor
         from sklearn.metrics import make_scorer
         from sklearn.grid_search import GridSearchCV

         def fit_model(X, y):
             """ Performs grid search over the 'max_depth' parameter for a
                 decision tree regressor trained on the input data [X, y]. """

             # Create cross-validation sets from the training data
             # sklearn version 0.18: ShuffleSplit(n_splits=10, test_size=0.1, train_size=None, r
             # sklearn versiin 0.17: ShuffleSplit(n, n_iter=10, test_size=0.1, train_size=None,
             cv_sets = ShuffleSplit(X.shape[0], n_iter = 10, test_size = 0.20, random_state = 0)

             # TODO: Create a decision tree regressor object
             regressor = DecisionTreeRegressor()

             # TODO: Create a dictionary for the parameter 'max_depth' with a range from 1 to 10
             params = {'max_depth':  list(range(1,11))}

             # TODO: Transform 'performance_metric' into a scoring function using 'make_scorer'
             scoring_fnc = make_scorer(performance_metric)

             # TODO: Create the grid search cv object --> GridSearchCV()
             # Make sure to include the right parameters in the object:
```

```
    # (estimator, param_grid, scoring, cv) which have values 'regressor', 'params', 'sc
    grid = GridSearchCV(regressor, params, scoring = scoring_fnc, cv = cv_sets)

    # Fit the grid search object to the data to compute the optimal model
    grid = grid.fit(X, y)

    # Return the optimal model after fitting the data
    return grid.best_estimator_
```

0.6.4 Making Predictions

Once a model has been trained on a given set of data, it can now be used to make predictions on new sets of input data. In the case of a *decision tree regressor*, the model has learned *what the best questions to ask about the input data are*, and can respond with a prediction for the **target variable**. You can use these predictions to gain information about data where the value of the target variable is unknown — such as data the model was not trained on.

0.6.5 Question 9 - Optimal Model

- What maximum depth does the optimal model have? How does this result compare to your guess in **Question 6**?

```
In [36]: # Fit the training data to the model using grid search
         reg = fit_model(X_train, y_train)

         # Produce the value for 'max_depth'
         print("Parameter 'max_depth' is {} for the optimal model.".format(reg.get_params()['max
```

```
Parameter 'max_depth' is 5 for the optimal model.
```

Answer:
The maximum depth of the optimal model is 5, and in the answer of Question 6 we found that max depth = 4 which very close and accurate to 5.

0.6.6 Question 10 - Predicting Selling Prices

Imagine that you were a real estate agent in the Boston area looking to use this model to help price homes owned by your clients that they wish to sell. You have collected the following information from three of your clients:

Feature	Client 1	Client 2	Client 3
Total number of rooms in home	5 rooms	4 rooms	8 rooms
Neighborhood poverty level (as %)	17%	32%	3%
Student-teacher ratio of nearby schools	15-to-1	22-to-1	12-to-1

- What price would you recommend each client sell his/her home at?
- Do these prices seem reasonable given the values for the respective features?

```
In [37]: # Produce a matrix for client data
         client_data = [[5, 17, 15], # Client 1
                        [4, 32, 22], # Client 2
                        [8, 3, 12]]  # Client 3

         # Show predictions
         for i, price in enumerate(reg.predict(client_data)):
             print("Predicted selling price for Client {}'s home: ${:,.2f}".format(i+1, price))
```

```
Predicted selling price for Client 1's home: $419,700.00
Predicted selling price for Client 2's home: $287,100.00
Predicted selling price for Client 3's home: $927,500.00
```

Answer:
Statistics for Boston housing dataset:
Minimum price: 105,000.0 Maximum price: 1,024,800.0 Mean price: 454,342.9447852761 Median price: 438,900.0 Standard deviation of prices: 165,171.13154429474
Client 1: - This home is within one std of the mean at approx. dollar 454k. With 5 rooms and a relatively low value of LSTAT and student-teacher ratio of 15:1 , the predicted price would be a good selling price.
Client 2: - The minimum price in the area is just over dollar 100k. However, their 4 rooms are helping raise the price of their home, since both teacher-student ratio and LSTAT are pretty high, this may have negative impact on the selling price.
Client 3: - With maximun no.of rooms i.e. 8 rooms , lowest LSTAT value and 12:1 student-teacher ratio, the prediction of 927,500 seems accurate as statistics for boston housing dataset indicate maximum prices go as high as 1,024,800.0
However the prices and factors may have changed now as of 2018 and the given data set is old too.

0.6.7 Sensitivity

An optimal model is not necessarily a robust model. Sometimes, a model is either too complex or too simple to sufficiently generalize to new data. Sometimes, a model could use a learning algorithm that is not appropriate for the structure of the data given. Other times, the data itself could be too noisy or contain too few samples to allow a model to adequately capture the target variable — i.e., the model is underfitted.

```
In [38]: vs.PredictTrials(features, prices, fit_model, client_data)
```

```
Trial 1: $391,183.33
Trial 2: $419,700.00
Trial 3: $415,800.00
Trial 4: $420,622.22
Trial 5: $413,334.78
Trial 6: $411,931.58
Trial 7: $399,663.16
Trial 8: $407,232.00
Trial 9: $351,577.61
Trial 10: $413,700.00

Range in prices: $69,044.61
```

0.6.8 Question 11 - Applicability

- In a few sentences, discuss whether the constructed model should or should not be used in a real-world setting.

Hint: Take a look at the range in prices as calculated in the code snippet above. Some questions to answering: - How relevant today is data that was collected from 1978? How important is inflation? - Are the features present in the data sufficient to describe a home? Do you think factors like quality of apppliances in the home, square feet of the plot area, presence of pool or not etc should factor in? - Is the model robust enough to make consistent predictions? - Would data collected in an urban city like Boston be applicable in a rural city? - Is it fair to judge the price of an individual home based on the characteristics of the entire neighborhood?

Answer:

1. The data collected is old, the prices today will be very different because of urbanisation in many areas.
2. It is missing features that could affect the selling price in today's housing market such as size of a backyard , interior of the house , exterior of the house, accessibility in terms of transportation , age of the property and nearbuy hospitals,schools,mall and etc.
3. The current model appears to be not well generalized as running it multiple times for a specific client provides a wide variance in pricing.
4. NO, the data collected in an urban city like Boston will not be applicable in a rural city , it can be used only in similar urban cities.
5. NO,it is not fair to judge the price of an individual home only based on the characteristics of the neighborhood. It is though one of the important parameter for considering the overall decision.

   Note: Once you have completed all of the code implementations and successfully answered each question above, you may finalize your work by exporting the iPython Notebook as an HTML document. You can do this by using the menu above and navigating to
   File -> Download as -> HTML (.html). Include the finished document along with this notebook as your submission.