

# Machine Learning Engineer Nanodegree

## Model Evaluation & Validation

### Project 1: 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.

## 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 (<https://archive.ics.uci.edu/ml/datasets/Housing>). 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 'MDEV' 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 'MDEV' are essential. The remaining **non-relevant features** have been excluded.
- The feature 'MDEV' 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 [3]: # Import libraries necessary for this project
import numpy as np
import pandas as pd
import visuals as vs # Supplementary code
from sklearn.cross_validation import ShuffleSplit

# Pretty display for notebooks
%matplotlib inline

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

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

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

## 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**, 'MDEV', will be the variable we seek to predict. These are stored in features and prices, respectively.

## 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 'MDEV', which is stored in prices.
  - Store each calculation in their respective variable.

```
In [4]: # TODO: Minimum price of the data
minimum_price = np.amin(prices)
# TODO: Maximum price of the data
maximum_price = np.amax(prices)
# TODO: Mean price of the data
mean_price = np.mean(prices)
# TODO: Median price of the data
median_price = np.median(prices)
# TODO: Standard deviation of prices of the data
std_price = np.std(prices)
# Show the calculated statistics
print "Statistics for Boston housing dataset:\n"
print "Minimum price: ${:,.2f}".format(minimum_price)
print "Maximum price: ${:,.2f}".format(maximum_price)
print "Mean price: ${:,.2f}".format(mean_price)
print "Median price ${:,.2f}".format(median_price)
print "Standard deviation of prices: ${:,.2f}".format(std_price)
```

Statistics for Boston housing dataset:

Minimum price: \$105,000.00  
 Maximum price: \$1,024,800.00  
 Mean price: \$454,342.94  
 Median price \$438,900.00  
 Standard deviation of prices: \$165,171.13

## 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 all Boston homeowners who have a greater net worth than homeowners in the neighborhood.
- '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 'MDEV' or a **decrease** in the value of 'MDEV'? Justify your answer for each.*

**Hint:** Would you expect a home that has an 'RM' value of 6 be worth more or less than a home that has an 'RM' value of 7?

**Answer:** price should increase with RM and decrease with PTRATIO and LSTAT Higher the number of rooms, higher the price. That's because people seek more rooms for luxury. Also, if the LSAT value is low then it means that the neighborhood has richer people.

People seek and good neighborhood, so it has an inverse relationship with price. If PTRATIO is low that means the schools have more teachers available for students which indicates better care to each student. So same inverse relation here.

---

## 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.

### 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* ([http://stattrek.com/statistics/dictionary.aspx?definition=coefficient\\_of\\_determination](http://stattrek.com/statistics/dictionary.aspx?definition=coefficient_of_determination)),  $R^2$ , 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  $R^2$  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  $R^2$  of 0 always fails to predict the target variable, whereas a model with an  $R^2$  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  $R^2$  as well, which indicates that the model is no better than one that naively predicts the mean of the target variable.*

For the `performance_metric` function in the code cell below, you will need to implement the following:

- Use `r2_score` from `sklearn.metrics` to perform a performance calculation between `y_true` and `y_predict`.
- Assign the performance score to the `score` variable.

```
In [5]: # 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
```

## 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

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

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

```
In [6]: # 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<sup>2</sup>, of 0.923.

**Answer: 0.923** I believe that this model captures the variance of the data well. Because the model explains 92.3% of the variance in the target variable which is a good number. closer the R<sup>2</sup> score better the performance.

## 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.

For the code cell below, you will need to implement the following:

- Use `train_test_split` from `sklearn.cross_validation` to shuffle and split the features and prices data into training and testing sets.
  - Split the data into 80% training and 20% testing.
  - Set the `random_state` for `train_test_split` to a value of your choice. This ensures results are consistent.
- Assign the train and testing splits to `X_train`, `X_test`, `y_train`, and `y_test`.

```
In [7]: # 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

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

Training and testing split was successful.

### 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?*

**Hint:** What could go wrong with not having a way to test your model?

**Answer:** evaluate performance on different sample sets, prevent overfitting. When we train the model on a training dataset there is a chance of overfitting. We won't come to know about overfitting until and unless we test the model on an entirely different dataset. That's why it's wise to separate the testing set prior to train the model.

---

## 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.

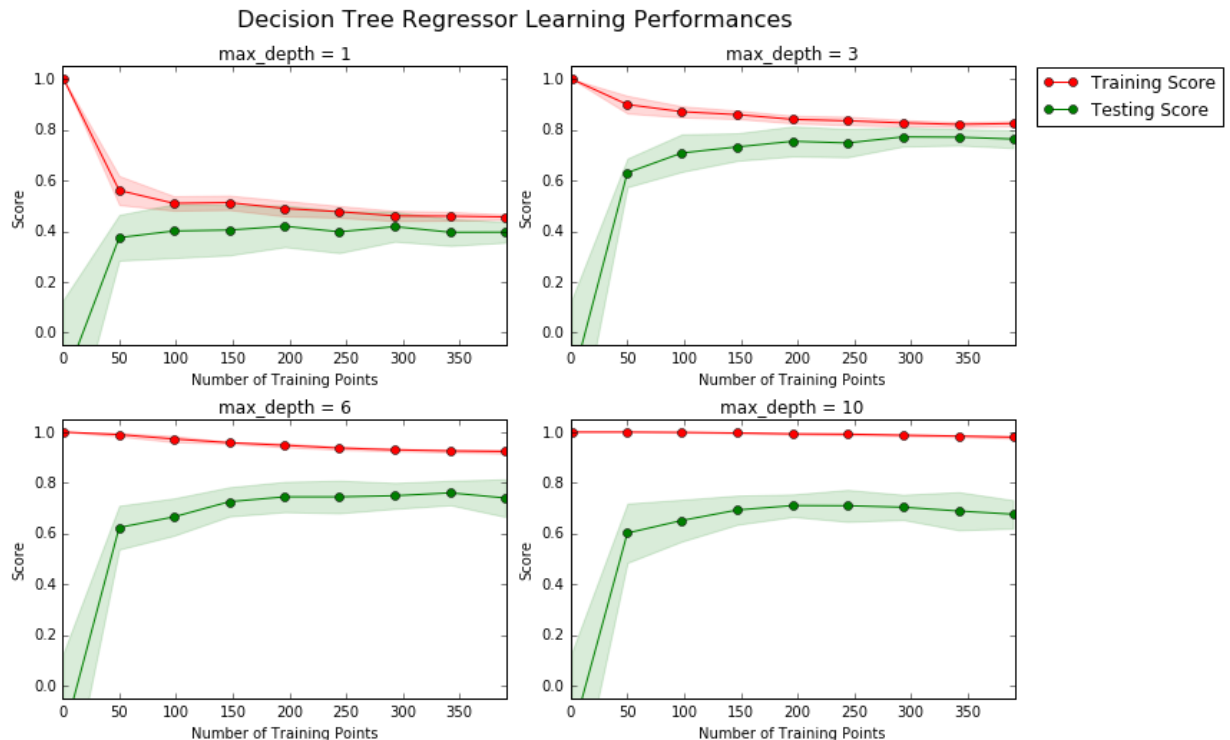
### Learning Curves

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  $R^2$ , the coefficient of determination.

Run the code cell below and use these graphs to answer the following question.

```
In [8]: # Produce Learning curves for varying training set sizes and maximum depths
vs.ModelLearning(features, prices)
```



## 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?

**Hint:** Are the learning curves converging to particular scores?

**\*\*Answer:** I choose max\_depth=3. The training score is very high for a few training points but the testing score was very low. This is because overfitting as we do not have enough data to justify the variance. As and when I add more training points, training score starts to drop slowly but the testing score starts to improve drastically. After n=150 both the curves freeze. Increasing the training points won't help here as both the curves freeze very early. Although the performance is already very good. So I don't think we need to think of finding a solution to improve the model any more.

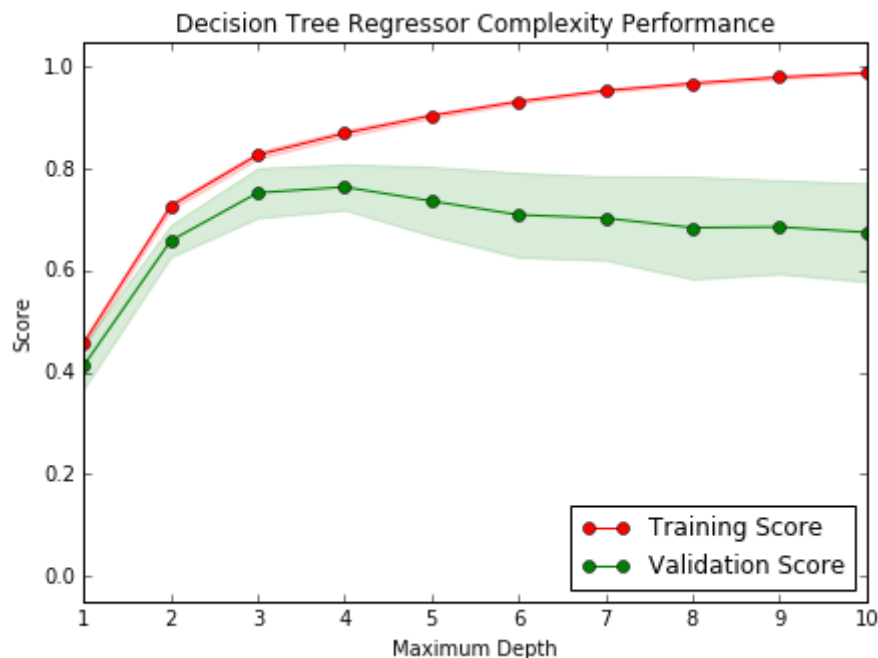
**\*\***

## 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.

Run the code cell below and use this graph to answer the following two questions.

```
In [9]: vs.ModelComplexity(X_train, y_train)
```



### 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?*

**Hint:** How do you know when a model is suffering from high bias or high variance?

**Answer:** At maximum depth of 1, the model suffers from high bias. The training and the testing curves converge at a low score.

At maximum depth of 10, the model suffers from high variance. There is a gap between the training curve and the testing curve. The training performance does not increase with sample size. Also the testing performance is consistently poor. That is a clear sign of overfitting

### 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:** According to me the ideal max-depth should be 3. The training and the testing curves nicely converge at 0.8 which is a high score.

---

## 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`.

### Question 7 - Grid Search

*What is the grid search technique and how it can be applied to optimize a learning algorithm?*

**Answer:** Grid search is the way to find the optimal value for model parameter. Many a time it's very difficult to set an optimal value for some model parameters like significance level, decision tree depth etc. We use a grid of different possible parameter values and check the scores. Grid search performs an exhaustive search of optimal model for every combination of model parameters. We don't have to depend on human intuition for the key values of the parameters.

### 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?*

**Hint:** Much like the reasoning behind having a testing set, what could go wrong with using grid search without a cross-validated set?

**Answer:** In k-fold cross-validation technique we split the data in k folds. We iterate over the folds and use those as testing sets and rest k-1 folds for training. Thus we can use the same dataset k times without repeating the exact sample. We can take the average of those k results to get a better generalization. Grid search requires a large number of dataset. Instead of using the same datasets again and again or using small samples we can use k-fold cross-validation

## 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*.

For the `fit_model` function in the code cell below, you will need to implement the following:

- Use `DecisionTreeRegressor` (<http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html>) from `sklearn.tree` to create a decision tree regressor object.
  - Assign this object to the 'regressor' variable.
- Create a dictionary for 'max\_depth' with the values from 1 to 10, and assign this to the 'params' variable.
- Use `make_scorer` ([http://scikit-learn.org/stable/modules/generated/sklearn.metrics.make\\_scorer.html](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html)) from `sklearn.metrics` to create a scoring function object.
  - Pass the `performance_metric` function as a parameter to the object.
  - Assign this scoring function to the 'scoring\_fnc' variable.
- Use `GridSearchCV` ([http://scikit-learn.org/stable/modules/generated/sklearn.grid\\_search.GridSearchCV.html](http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html)) from `sklearn.grid_search` to create a grid search object.
  - Pass the variables 'regressor', 'params', 'scoring\_fnc', and 'cv\_sets' as parameters to the object.
  - Assign the `GridSearchCV` object to the 'grid' variable.

```
In [10]: # TODO: Import 'make_scorer', 'DecisionTreeRegressor', and 'GridSearchCV'
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import make_scorer
from sklearn.grid_search import GridSearchCV
from sklearn import cross_validation

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
    cv_sets = cross_validation.ShuffleSplit(X.shape[0], n_iter = 10, test_size =

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

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

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

    # TODO: Create the grid search object
    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_
```

## 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.

## Question 9 - Optimal Model

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

Run the code block below to fit the decision tree regressor to the training data and produce an optimal model.

```
In [11]: # 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(
Parameter 'max_depth' is 4 for the optimal model.
```

**Answer: Parameter 'max\_depth' is 4 for the optimal model. I guessed 3 in question6. I wasn't exactly right but I was close.**

## 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
Household net worth (income)	Top 34th percent	Bottom 45th percent	Top 7th percent
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?*

**Hint:** Use the statistics you calculated in the **Data Exploration** section to help justify your response.

Run the code block below to have your optimized model make predictions for each client's home.

```
In [12]: # Produce a matrix for client data
client_data = [[5, 34, 15], # Client 1
               [4, 55, 22], # Client 2
               [8, 7, 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: $344,400.00
Predicted selling price for Client 2's home: $237,478.72
Predicted selling price for Client 3's home: $931,636.36
```

**\*\*Answer:** Predicted selling price for Client 1's home:

344,400.00 Predicted selling price for Client 2's home: 241,981.25 Predicted selling price for Client 3's home: \$946,800.00

Client1: the predicted value lies within one standard deviation from the mean and median. Client1 has the features which is in between the other two clients. So having the estimated prices in between of their's is justified.

Client2: the predicted value lies between one standard deviation and two standard deviations below the mean/median. Client2 has lowest RM and highest PTRATIO and LSTAT among these three. That justifies the lowest estimated price.

Client3: the predicted value lies more than three standard deviations above the mean/median. Client3 has highest RM and lowest PTRATIO and LSTAT among these three. That justifies the highest estimated price.

But client1 and client3 have a little difference in the estimated prices where as their features vary a lot. This could have been better estimated.

**\*\***

## 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. Run the code cell below to run the `fit_model` function ten times with different training and testing sets to see how the prediction for a specific client changes with the data it's trained on.

```
In [13]: vs.PredictTrials(features, prices, fit_model, client_data)
```

```
Trial 1: $324,240.00  
Trial 2: $324,450.00  
Trial 3: $346,500.00  
Trial 4: $420,622.22  
Trial 5: $413,334.78  
Trial 6: $411,931.58  
Trial 7: $344,750.00  
Trial 8: $407,232.00  
Trial 9: $352,315.38  
Trial 10: $316,890.00
```

Range in prices: \$103,732.22

## Question 11 - Applicability

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

**Hint:** Some questions to answering:

- *How relevant today is data that was collected from 1978?*
- *Are the features present in the data sufficient to describe a home?*
- *Is the model robust enough to make consistent predictions?*
- *Would data collected in an urban city like Boston be applicable in a rural city?*

**\*\*Answer:**

1. Though we can get a very faded idea about the house pricing but the data is not dependable. There are many other features that have come into picture. Also the scenarios have changed. Like the PTRATIO has definitely changed.
2. Those are not sufficient features to describe a home. Features like area, crime rate, distance from malls etc. should make an effect.
3. The above sensitivity analysis clearly show that the predictions vary a lot. That raise a question about the robustness of the model.
4. If we try to apply this model for rural area then it should not work. We can not train our model in one biased sample and test it on a different sample. Samples from urban area and rural area differ a lot.

**\*\***

## A brief appraisal of the model:

This was a simple model to start working with. As it has only three features to use for prediction, we are not able to capture the variations more accurately. Our optimal model didn't suffer from lack of training datapoints. Although, with more features, we can use more training points.

