

# Machine Learning Engineer Nanodegree

## Model Evaluation & Validation

### 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.

# 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) (<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 '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 [3]: # 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.shape)
```

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**, 'MEDV', 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 'MEDV', 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 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.

**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?

RM Feature: Usually the more rooms a house has, the more expensive it is because it is understood that it may be a bigger house (area wise). LSTAT Feature: given that this feature portrays the percentage of lower class population in the area, the higher it is, I would expect the house pricing to decrease since the this population may only be able to purchase houses on this price range PTRATIO: Generally, in upper class communities schools offer a lower PTRATIO so teachers can dedicate more time to each student. Having said this, the lower the PTRatio, I would expect to see a price increase since it may mean that the community is closer to an upper class community

---

## 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 is no better than a model that always predicts the *mean* of 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 **arbitrarily worse** than one that always 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,  $R^2$ , of 0.923.

The coefficient of determination shows a value of 0.923 which is very close to 1, hence, indicating that the model obtained can closely describe the relationship between the two variables

## 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 import cross_validation # for version 0.17

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

The advantage of breaking the data in train and test datasets is that we'd be able to confirm or evaluate our prediction model using new data, hence, avoiding overfitting the model

---

## Analyzing Model Performance

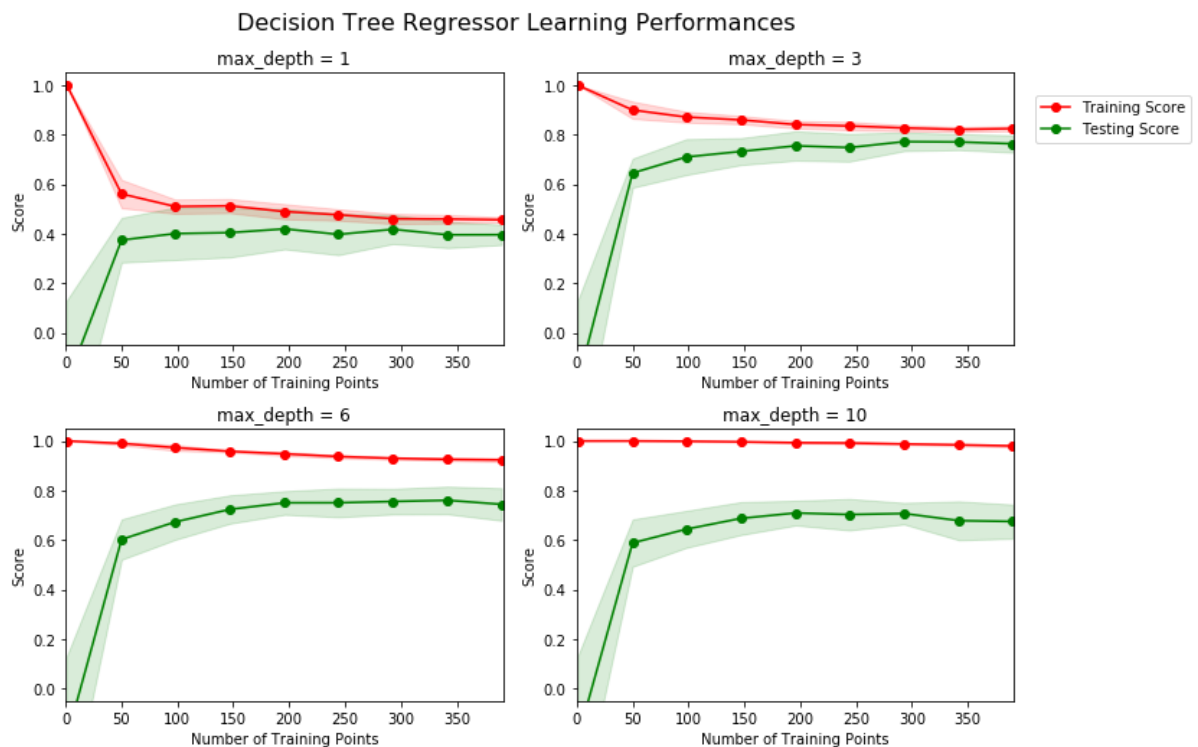
In this third section of the project, you'll take a look at several models' learning and testing performances on various subsets of training data. Additionally, you'll investigate one particular algorithm with an increasing '`max_depth`' parameter on the full training set to observe how model complexity 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?



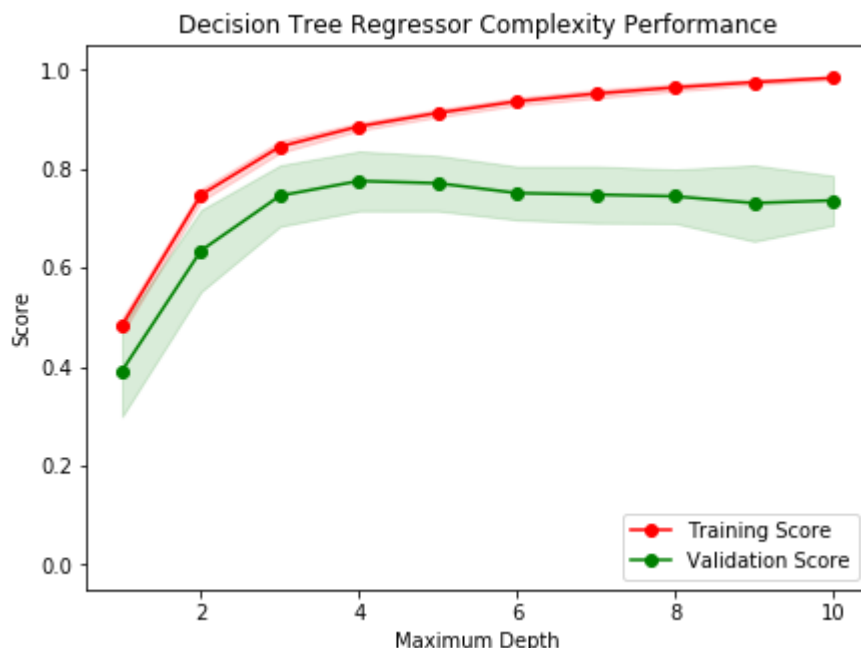
-Choose one of the graphs above and state the maximum depth for the model. I pick Max\_depth = 3 because it is a good trade off between test accuracy and model complexity. It also allows having very close performance to the training set giving more confidence in it -(UPDATE) What happens to the score of the training curve as more training points are added? The accuracy score (training score) decreases while/when training size increases. On the other hand, the validation score reaches a high accuracy after 100 or 150 samples, but after this point, the validation score has little variation; therefore selecting a training set larger than 150 samples may be unnecessary depending on the objective of the analysis and data availability. Thanks for this comment:"Which makes sense, since with little amounts of the data we simply memorize the training data(no generalization), then when we receive more and more data points we can't simply memorize the training data and we start to generalize better(higher testing accuracy)", I would have not seen it that way myself. -What about the testing curve? Testing curve rapidly finds a "stable" accuracy level after 100 Training set size. At eye level, the best model performance in the test data was at depth = 3. -Would having more training points benefit the model? Not really. for increases in training set size were helping training accuracy BUT they did not improve test set accuracy so no major advantages on further increasing training set

## 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 [10]: 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?

- When the model is trained with a maximum depth of 1, does the model suffer from high bias or from high variance? It suffers of high bias because it "consistently" misses the predictions (low performance score)
- How about when the model is trained with a maximum depth of 10? It suffers of high variance because despite the higher complexity and better training performance, it still underperforms on its predictions on the validation set, particularly if compared to depths 3 and 4.
- What visual cues in the graph justify your conclusions? The validation score did not change(or decreased) after a depth of 4 despite improving training score

In [ ]:

## 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?*

I'd use a depth of 4 since it has a good balance between complexity and prediction score. (update). "Maximum" accuracy is equivalent to "minimum" error which is depicted in the Bias-Variance Tradeoff Chart as the lowest aggregated error from both bias and variance. The more complex the model, the higher the bias of the model obtained (minimum bias error; in the other hand, variance which measures the accuracy of the test (new) data, varies from a relatively "low" variance error to a very high one when the model complexity increases. In the chart above, that minimal error (~ Max accuracy) occurred at around Depth = 4.

---

## 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?*

Gridsearch allows to fine tune model parameters (by using different combinations of them) using crossvalidation for each set of parameters so as to define or determine which set provides the best model performance. It can be used to find the specific set of parameters that will provide the best performance for any model (algorithm) being studied

## 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?

K-Fold Technique breaks a dataset into K "folds" or sub-datasets. Cross-validation selects, for each iteration, one of the K-fold subsets as testing set, while the remaining are treated as training set. This will happen K times, one for each K generated fold. On each iteration the prediction error is calculated, and at the end of the iterations, all these prediction errors are averaged to obtain one error for the entire procedure. It is important to note that by following this procedure as-is, we may end up having a too optimistic model, so a mean to decrease this chance and if data availability is not an issue, the original data could be separated in Training and Test (Validation) sets and then applying K-Fold cross-validation to the obtained training dataset. At the end of the k-fold crossvalidation we'll obtain our trained model and apply it to the new (unseen) data from the test(validation) dataset. This method would allow our model to be rather on the conservative side of the predictions. Benefit of CV to Gridsearch: CV allows to use the entire set of data for all combinations of parameters, hence, avoiding issues of underfitting or overfitting given by the specific combination of K subsets used for model training.

## 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.

Please note that `ShuffleSplit` has different parameters in scikit-learn versions 0.17 and 0.18. 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/0.17/modules/generated/sklearn.grid\\_search.GridSearchCV.html](http://scikit-learn.org/0.17/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 [11]: ## TODO: Import 'make_scorer', 'DecisionTreeRegressor', and 'GridSearchCV'

#from sklearn.cross_validation import ShuffleSplit
from sklearn.tree import DecisionTreeRegressor
#from sklearn.model_selection import ShuffleSplit
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import make_scorer

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 = ShuffleSplit(n_splits = 10, test_size = 0.20, random_state = 0)
    #cv_sets = cv_sets.split(X)
    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(random_state=0)

    # TODO: Create a dictionary for the parameter 'max_depth' with a range from 1 to 10
    params = {'max_depth':[1,2,3,4,5,6,7,8,9,10]} #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 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_

```

C:\ProgramData\Anaconda2\lib\site-packages\sklearn\grid\_search.py:43: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model\_selection module into which all the refactored classes and functions are moved. This module will be removed in 0.20.  
 DeprecationWarning)

## 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 [12]: # 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_depth'])

Parameter 'max_depth' is 4 for the optimal model.
```

The optimal value obtained by the `fit_model` function matches with the value I originally identified

## 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?*

**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 [14]: # 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)
features.describe()
```

```
Predicted selling price for Client 1's home: $391,183.33
Predicted selling price for Client 2's home: $189,123.53
Predicted selling price for Client 3's home: $942,666.67
```

Out[14]:

	RM	LSTAT	PTRATIO
<b>count</b>	489.000000	489.000000	489.000000
<b>mean</b>	6.240288	12.939632	18.516564
<b>std</b>	0.643650	7.081990	2.111268
<b>min</b>	3.561000	1.980000	12.600000
<b>25%</b>	5.880000	7.370000	17.400000
<b>50%</b>	6.185000	11.690000	19.100000
<b>75%</b>	6.575000	17.120000	20.200000
<b>max</b>	8.398000	37.970000	22.000000

I would recommend the following prices: Predicted selling price for Client 1's home: 391,183.33 Predicted selling price for Client 2's home: 189,123.53 Predicted selling price for Client 3's home: \$942,666.67

The obtained prices seem reasonable to me. 1) Bigger houses (more rooms) are expected to cost more (case for Client-3). On top of this, the house is also located on a non-poverty area (only 3% poverty level) which further supports the higher price of this house against others) 2) Poverty level seems to follow the pattern of cost, this is, Client two is looking at an area where the poverty level is the highest along with the highest Student to Teacher ratio. The characteristics/features of this house would make it the lowest price of all which actually is the prediction of our model. 3) Client one is looking for a house a bit larger (by one room) than the one from Client 2, but it is located in a less "poor" neighborhood (15% vs 32% poverty level) with better PTRATIO ratio.

After looking at the results of features.describe() it re-confirms the observations above: 1) RM Avg = 6. This leaves Client 3 far beyond the RM average, followed by Client 1 and 2. In particular, Client 2 is just slightly above the minimum RM sizes in Boston. 2) LSTAT: In this case, Client 3 is well below the LSTAT average confirming its high price while Client 1 and 2 are above the average, hence, identifying their locations on working poor neighborhoods 3) PTRATIO: Client 1 is close to the AVG PTRATIO while Client 3 is just slightly above of the Minimum PTRATIO of the dataset giving an idea of how this Client features' combination produce ends up defining a much higher price for its house

## 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 [15]: vs.PredictTrials(features, prices, fit_model, client_data)
```

```
Trial 1: $391,183.33  
Trial 2: $424,935.00  
Trial 3: $415,800.00  
Trial 4: $420,622.22  
Trial 5: $418,377.27  
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: $73,357.39
```

## 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?*



The data used to train the model was collected in 1978, which makes this model less reliable considering potential changes in the real state market such as neighborhoods changes (for good and bad), poverty levels, etc. #0 years is a long time even if the model was corrected for inflation. For those of us that have been in the market for a house, certainly while these features are relevant, there are other features that depending on the family could be as or more important than these, i.e. proximity to public transportation, city taxes, school quality levels, crime ratings (poor does not mean dangerous) among others. By looking at the PredictTrials function I realized the predictions are made for Client # 1 which had an initial prediction of 391K. By changing the random selection of samples for training and test datasets we encountered a variation of 73K with a trial average of 405K. This yields a maximum 18% variation from the average (assuming all variation occurs from the average point). Having said this we could say that we can get a 82% or more accuracy on our prediction which is a very reasonable prediction level in my opinion, however, it often times depends on the exact accuracy needs of the problem to solve to make that decision. Certainly, I would only recommend to use this model in Boston since other cities (large or small) will not necessarily have the same feature relationship with the prices. In rural areas these difference may be more marked.

**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.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: