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). 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 [1]:
```
# 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.".form
at(*data.shape)
```

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

In [2]:
```
data.head(10)
```

Out[2]:

	RM	LSTAT	PTRATIO	MEDV
0	6.575	4.98	15.3	504000.0
1	6.421	9.14	17.8	453600.0
2	7.185	4.03	17.8	728700.0
3	6.998	2.94	18.7	701400.0
4	7.147	5.33	18.7	760200.0
5	6.430	5.21	18.7	602700.0
6	6.012	12.43	15.2	480900.0
7	6.172	19.15	15.2	569100.0
8	5.631	29.93	15.2	346500.0
9	6.004	17.10	15.2	396900.0

In [3]:
```
data.describe()
```

Out[3]:

	RM	LSTAT	PTRATIO	MEDV
count	489.000000	489.000000	489.000000	4.890000e+02
mean	6.240288	12.939632	18.516564	4.543429e+05
std	0.643650	7.081990	2.111268	1.653403e+05
min	3.561000	1.980000	12.600000	1.050000e+05
25%	5.880000	7.370000	17.400000	3.507000e+05
50%	6.185000	11.690000	19.100000	4.389000e+05
75%	6.575000	17.120000	20.200000	5.187000e+05
max	8.398000	37.970000	22.000000	1.024800e+06

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]:
```python
# # TODO: Minimum price of the data
# minimum_price = prices.min()

# # TODO: Maximum price of the data
# maximum_price = prices.max()

# # TODO: Mean price of the data
# mean_price = prices.mean()

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

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

'''
Using Numpy instead of Pandas for statistics
'''
# TODO: Minimum price of the data
minimum_price = np.min(prices)

# TODO: Maximum price of the data
maximum_price = np.max(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?

Answer:

1. Increase in 'RM' will lead to increase in 'MEDV', because larger 'RM' usually means bigger house, which will cost more.
2. Increase in 'LSTAT' will lead to decrease in 'MEDV', because poorer neighbor will affect the housing price negatively.
3. Increase in 'PTRATIO' will lead to decrease in 'MEDV', becuase more teacher usually means better education.

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), 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]:
```python
# 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.
```

Answer: My short answer is 'Yes'. Although the none of the predicted value is exactly the same the the true value, I think the prediction predict the trend well. And the R^2 score is quite high, indicating a good model to predict the data.

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_siz
e=0.2, random_state=20)

# 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: A simple answer is that if we don't split the dataset into training set and testing set, and we test the performance of our model only on the trainning, i.e. the whole dataset, the misfit error will be very small, or the score will be very high. But the small misfit error, or the high evaluation score could be caused by 'overfitting' to the dataset, rather than a good prediction model. If we split the dataset and only build our model base on the training set and latter test it on the testing set, if we have overfitting problem, we can easily tell that from a high training score but a low testing score.

Analyzing Model Performance

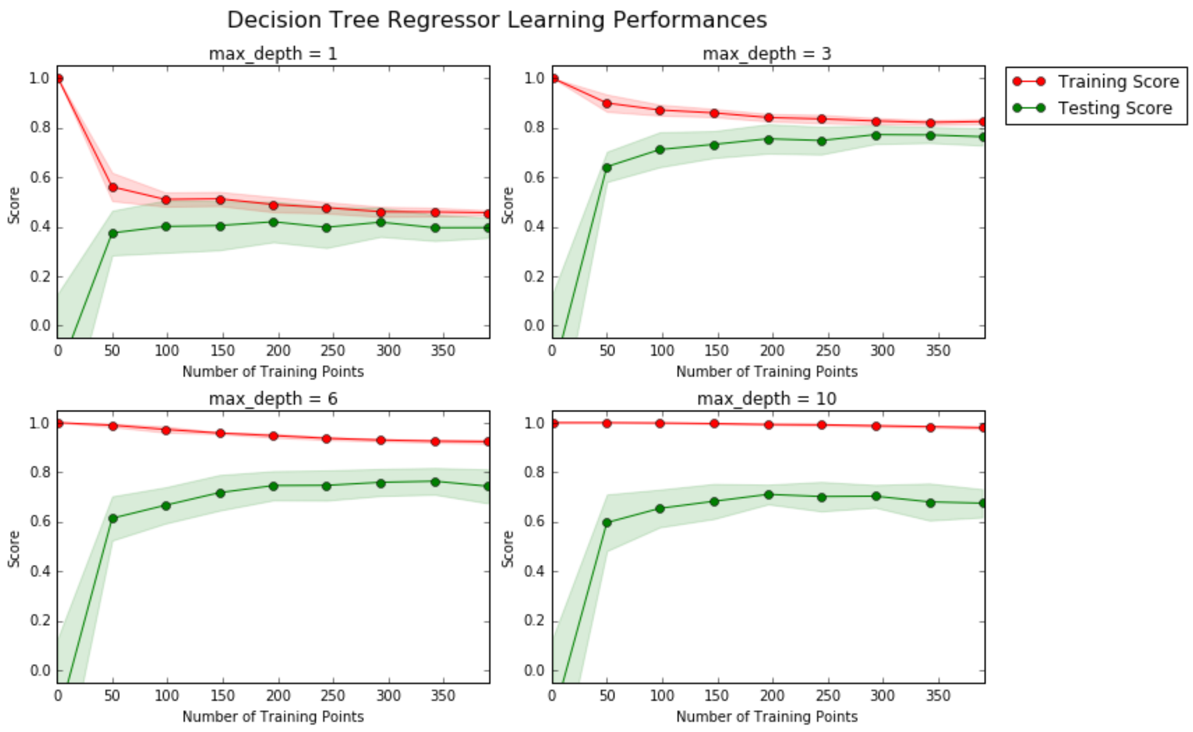
In this third section of the project, you'll take a look at several models' learning and testing performances on various subsets of training data. Additionally, you'll investigate one particular algorithm with an increasing `'max_depth'` parameter on the full training set to observe how model complexity 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: In general, with more training points, the score of training set will decrease, because it is easier to fit few points well rather than lots of points well. The score of the testing set will increase, because with more training point, the model is trained better.

Would having more training points benefit the model?

This depends on how complicated our model is. In this case, it depends on the max_depth parameter. Higher max_depth value means more complicated model. Taking two extreme cases as examples.

1. When max_depth=1, the model is too simple, thus it is suffering 'underfitting' or 'high-bias' error. What we should do is to increase the complexity of the model. Getting more data does not help for this case. From the learning curve, the training curve and testing curve quickly merge, but to a low score.
2. When max_depth=10, the model is too complicated. Usually, for models suffereing from 'overfitting', more dataset will help. However, the testing score already stops improving and starts to diverge, which means more dataset won't help even for this case.

Thanks for providing the fruitful comments! Yes, as you said, usually overfitting problem will benefit from more dataset. And that is how I thought it should be. So as long as I identify max_depth = 6 or 10 to be an overfitting problem, I thought more data will definitely help. But you are right! The train_score and testing_core already start to diverge with more training points! This is a sign that more dataset will not help!

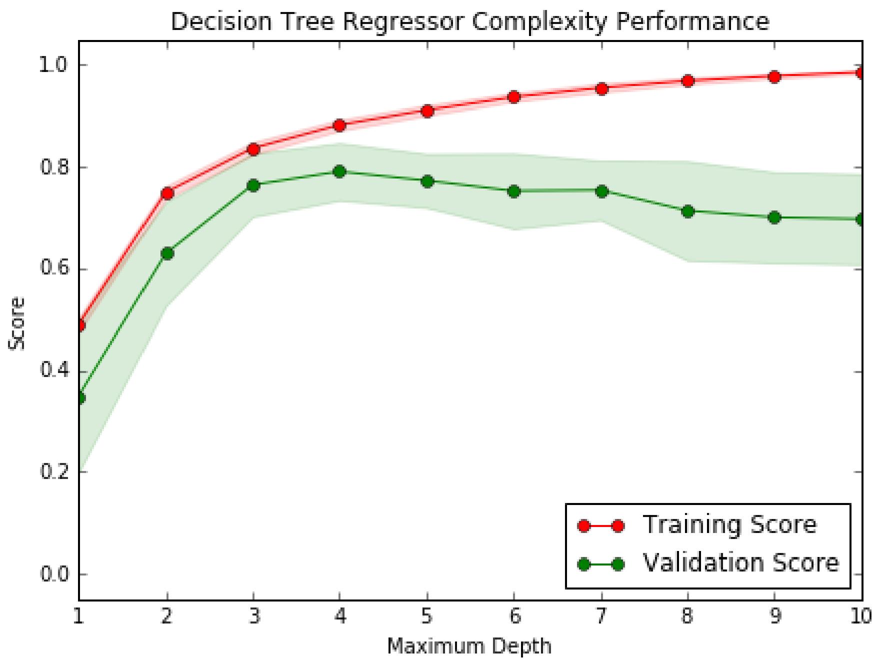
**Old Answer: ** In general, with more training points, the score of training set will decrease, because it is easier to fit few points well rather than lots of points well. The score of the testing set will increase, because with more training point, the model is trained better. *Would having more training points benefit the model?* This depends on how complicated our model is. In this case, it depends on the max_depth parameter. Higher max_depth value means more complicated model. Taking two extreme cases as examples. 1. When max_depth=1, the model is too simple, thus it is suffering 'underfitting' or 'high-bias' error. What we should do is to increase the complexity of the model. Getting more data does not help for this case. From the learning curve, the training curve and testing curve quickly merge, but to a low score. 2. When max_depth=10, the model is too complicated. More data will help for this case. The learning curves have two features. First, the two curves are not merging. Second, the traning curve is almost not decreasing.

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: The complexity curves also tell us whether model is suffering from high bias or high variance, in a different way compared with the learning curves.

1. maximum depth =1, model is high bias. Because both training score and validation score are low and similar.
2. maximum depth =10, model is high variance. Beause training score is high, meaning model fit training set very well. But the validation score is low, meaning the model fits the test set terribly.

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: Maximum depth = 4 is the best model. The complexity curves give this answer. The validation score with the testing set has highest score when maximum 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?

Answer:

In machine learning, to train or build an estimator, sometimes you need to provide extra hyper-parameters. These hyper-parameters are not directly learnt with the training data. Different choices of the hyper-parameter value, or even conbination of various hyper-parameters, will give different learning models with different capability to predict the data accurately. Grid search technique just exhaustively try all the values of the hyper-parameters you specified by changing one value at a time while fixing all the other values. Then evaluation scores were generated with the cross-validation set at each run. In the end, by looking at the highest score, we can decide what is the best value/combination of the hyper-parameters for the mechine learning estimator.

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: Below is the answer from Sklearn user guide:

> When evaluating different settings ("hyperparameters") for estimators, such as the C setting that must be manually set for an SVM, there is still a risk of overfitting on the test set because the parameters can be tweaked until the estimator performs optimally. This way, knowledge about the test set can "leak" into the model and evaluation metrics no longer report on generalization performance. To solve this problem, yet another part of the dataset can be held out as a so-called "validation set": training proceeds on the training set, after which evaluation is done on the validation set, and when the experiment seems to be successful, final evaluation can be done on the test set. However, by partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets. A solution to this problem is a procedure called cross-validation (CV for short). A test set should still be held out for final evaluation, but the validation set is no longer needed when doing CV. In the basic approach, called k-fold CV, the training set is split into k smaller sets (other approaches are described below, but generally follow the same principles). The following procedure is followed for each of the k "folds":
>
> - A model is trained using k-1 of the folds as training data;
> - the resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).
>
> The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop. This approach can be computationally expensive, but does not waste too much data (as it is the case when fixing an arbitrary test set), which is a major advantage in problem such as inverse inference where the number of samples is very small.

This is already a very clear explanation of the k-fold cross-validation and why we should use it.

One more point I want to add to this explanation in terms of 'randomness' of the way training and validation sets are splitted. We can try diffferent ways to choose chunks of the whole dataset as training set or cross-validation set. Every time we do this and calculate the score using the model trained from the training set and testing it on the cross-validation set, the score could have a large variation. Cross-validation score is obtained as the average of the score using different training set and cross-validation set, thus is more robust and stable. For more, read Stanford class note (https://lagunita.stanford.edu/c4x/HumanitiesScience/StatLearning/asset/cv_boot.pdf).

If we do grid search not using cross-validated set, then for sure the most overfitted model will have the best score, which is just a illusion.

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_iter'`) 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.

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) 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) 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

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(X.shape[0], n_iter = 10, test_size = 0.20, random_s
tate = 0)

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

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

    # 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_
```

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_para
ms()['max_depth'])
```

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

Answer: 'max_depth' = 4. This is the same to my answer for Question 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?
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, 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: $306,337.50
Predicted selling price for Client 2's home: $224,342.55
Predicted selling price for Client 3's home: $930,490.91
```

In [13]:
```
# choose the data in the dataset with RM around 5,
# then look at the distribution of the LSTAT and PTRATIO features and the pric
ing
data[(data.RM>4.8)&(data.RM<5.2)].describe()
```

Out[13]:

	RM	LSTAT	PTRATIO	MEDV
count	13.000000	13.000000	13.000000	13.000000
mean	5.004000	25.625385	18.438462	286569.230769
std	0.095342	8.142193	2.660056	86439.153789
min	4.880000	12.120000	14.700000	155400.000000
25%	4.926000	20.080000	14.700000	214200.000000
50%	5.000000	29.290000	20.200000	302400.000000
75%	5.036000	30.620000	20.200000	338100.000000
max	5.186000	34.770000	21.200000	459900.000000

In [14]:
```
# choose the data in the dataset with RM around 4,
# then look at the distribution of the LSTAT and PTRATIO features and the pric
ing
data[(data.RM>3.8)&(data.RM<4.2)].describe()
```

Out[14]:

	RM	LSTAT	PTRATIO	MEDV
count	3.000000	3.000000	3.0	3.000000
mean	4.046333	24.880000	20.2	341600.000000
std	0.158771	12.391977	0.0	125865.761826
min	3.863000	13.330000	20.2	249900.000000
25%	4.000500	18.335000	20.2	269850.000000
50%	4.138000	23.340000	20.2	289800.000000
75%	4.138000	30.655000	20.2	387450.000000
max	4.138000	37.970000	20.2	485100.000000

In [15]:
```
# choose the data in the dataset with RM around 8,
# then look at the distribution of the LSTAT and PTRATIO features and the pric
ing
data[(data.RM>7.8)&(data.RM<8.2)].describe()
```

Out[15]:

	RM	LSTAT	PTRATIO	MEDV
count	5.000000	5.000000	5.000000	5.000000e+00
mean	7.920400	3.696000	16.600000	8.988000e+05
std	0.123581	0.392912	1.662829	9.629135e+04
min	7.820000	3.130000	14.700000	7.896000e+05
25%	7.820000	3.570000	14.900000	8.127000e+05
50%	7.853000	3.760000	17.400000	9.198000e+05
75%	8.040000	3.810000	18.000000	9.534000e+05
max	8.069000	4.210000	18.000000	1.018500e+06

In [16]:
```
data.describe()
```

Out[16]:

	RM	LSTAT	PTRATIO	MEDV
count	489.000000	489.000000	489.000000	4.890000e+02
mean	6.240288	12.939632	18.516564	4.543429e+05
std	0.643650	7.081990	2.111268	1.653403e+05
min	3.561000	1.980000	12.600000	1.050000e+05
25%	5.880000	7.370000	17.400000	3.507000e+05
50%	6.185000	11.690000	19.100000	4.389000e+05
75%	6.575000	17.120000	20.200000	5.187000e+05
max	8.398000	37.970000	22.000000	1.024800e+06

Answer:

```
data[(data.RM>4.8)&(data.RM<5.2)].describe()
data[(data.RM>3.8)&(data.RM<4.2)].describe()
data[(data.RM>7.8)&(data.RM<8.2)].describe()
data.describe()
```

The codes above describes the statistics of each features and the housing price of the boston dataset we are using. And comparing with the three new clients' data, the features value are within the range of the feature value we have in the dataset. Then I only pick the data from the dataset with RM feature similar to the clients' RM feature and then look at the range of the other two features and the price. I think the predicted prices for the clients' house fell into the range of the prices of the houses with similar feature values in our dataset. That's why I think these are reasonable prices to recommend.

It is difficult to answer whether the predicted prices are perfect. Based on the low R^2 score of our best model, I am not sure our model has really good ability to predict the housing price for new dataset with new feature values. But with the statistics we have, I couldn't find a reason to say the predicted prices are unreasonable.

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 [17]: 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?*

Answer:

- I think this old dataset can still provide some values to understand how to predicting housing price today. Some fundmental rules still remains, like people want to spend more money for a bigger house, better education opportunity and better neighborhood. However, I don't think we can directly use the old data to predict the housing price in Boston now. 1. because of inflation, the price is not that **cheap** any more (Bummer! I am now living in Boston and thinking about buying a house...) 2. There could be some features that matter more in 1978 but not nowadays. And there could be new features that is playing an important role today, which did not exist in 1978. I am guessing if the house is closer to a high-tech company, like research lab of microsoft, google, or some excellent bio-tech companies, the price will be higher. Many of the high-tech and bio-tech companies probably did not even exist in 1978.
- From the complexity curve, we see that even for the best parameter value, max_depth=4, the R^2 score is 0.8. Probably we need to find more features or try other methods to build a better model to predict the housing price.
- I think it is robust enough. Based on the results from the **Sensitivity** test.
- No, I bet the price in rural city will be cheaper, for the same values in the features.

> **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.