Machine Learning Engineer Nanodegree

Model Evaluation & Validation ¶ (//htmlpreview.github.io/?https://github.com/robertodias/machine_learning/blob/master/py 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 `'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 [5]:
```
# 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['MEDV']
features = data.drop('MEDV', axis = 1)

# Success
print "Boston housing dataset has {} data points with {} variables eac
h.".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 [6]:
```
# 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?

Answer: The use of Data visualization tools (dataset distribution) helped me to understand the relation between the features on our model.

-**RM**: considering we are using the same LSTAT and PTRATIO I could verify that by increasing the RM (number of rooms) it also increased the MEDV (home price).

-**LSTAT**: considering we are using the same RM and PTRATION I could verify that LSTAT have an inverse relation with the MEDV, which means, by increasing the LSTAT (lower class neighborhood) it decreased the MED (home price).

-**PTRATIO**: PTRATIO (students per teacher) behaves almost like the LSTAT (lower class neighborhood), but it is not strong as the LSTAT. I noticed that its distribution was way more spreaded and that sometimes by increasing the PTRATIO it also increased the MEDV. But, the overall tendency showed to be an inverse relation with MEDV, too.

Based on how our features are behaving in our model, I would say that a home with RM 6 will probably worth more than a home with RM 7, when the LSTAT or the PTRATIO of the first home is lower compared with the second home. And also, that a home with RM 6 will probably worth less than a home with RM 7 when both have the same (or close) values for the other features (LSTAT and PTRATION).

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 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 [7]:
```
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_pr
edict'
    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 [8]:
```
# 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}.".form
at(score)
```

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

Answer: It is visible that the given model is following the target variations. It is not accurate, but it getting close. We can see it represented by the coefficient of determination that is very close to 1, it is in fact 0.92.

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 [9]:
```
from sklearn.cross_validation import train_test_split

# TODO: Shuffle and split the data into training and testing subsets
X_train, X_test, y_train, y_test = train_test_split(features, prices,
test_size=0.2, random_state=47)

# 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: Split the dataset provide us a feedback for comparison. With it we can check if the training data and the testing data are following the same approach on determination and that we are not **overfitting** the model. It works like a double check, in order to make sure that our model was well defined and will be good to provide consistent predictions. By not splitting the dataset there is risk of never validate it with different data, that way becomes hard to guarantee the effectiveness of our model, as we may be assuming data patterns (on our first round of data entry) that are not real patterns for the next set of data that might be applied to our model in the future.

Analyzing Model Performance

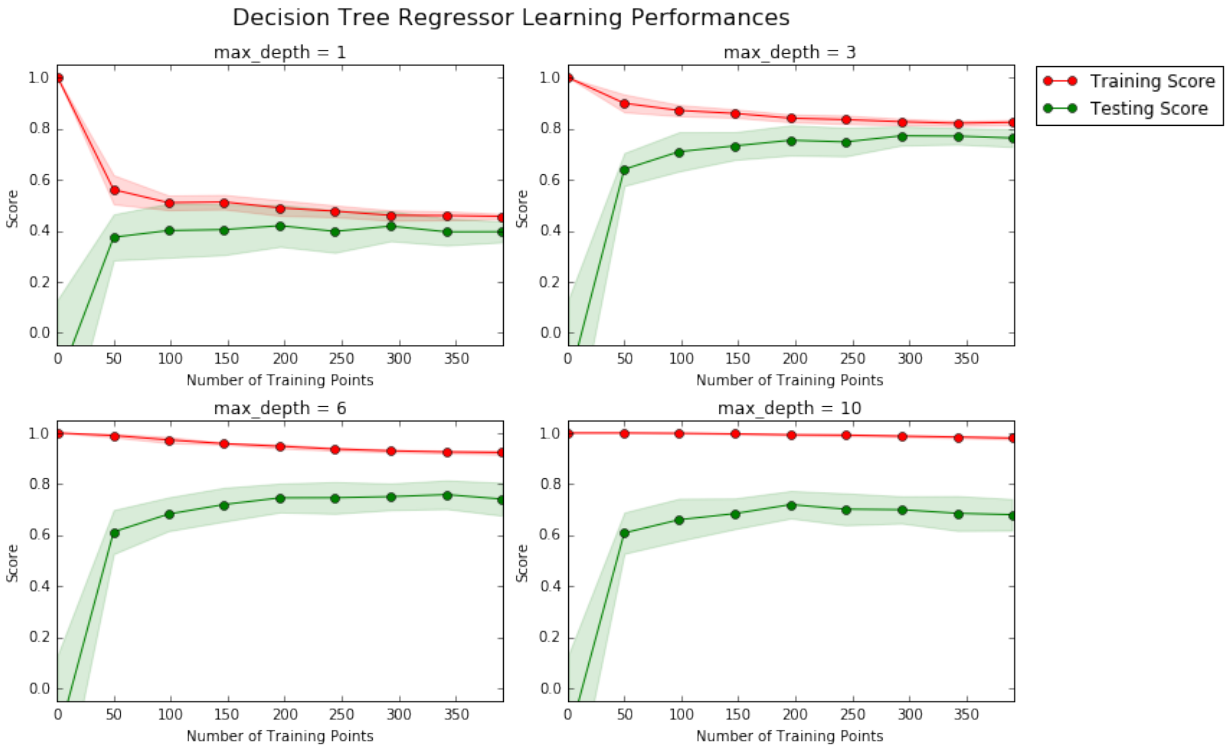
In this third section of the project, you'll take a look at several models' learning and testing performances on various subsets of training data. Additionally, you'll investigate one particular algorithm with an increasing `'max_depth'` parameter on the full training set to observe how model complexity 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 [10]:
```
# 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: After analysing the graphs, I decided to use maximum depth equals to 3, for this model, as both curves are converging in to a reasonable score.

Curves analysis when max_depth=3:

- The training curve starts with score 1 and when we added more training points this curve falls close to 0.9. Furthermore, as more training points are added the curve stabilizes around score 0.8.
- The testing curve starts with score 0 and then, after the first 50 training points, it raises to score 0.6. As more training points are added the curve stabilizes around score 0.8, as the training curve.

After more than 350 training points both curves seemed to be converging indicating that we found a good generalization model.

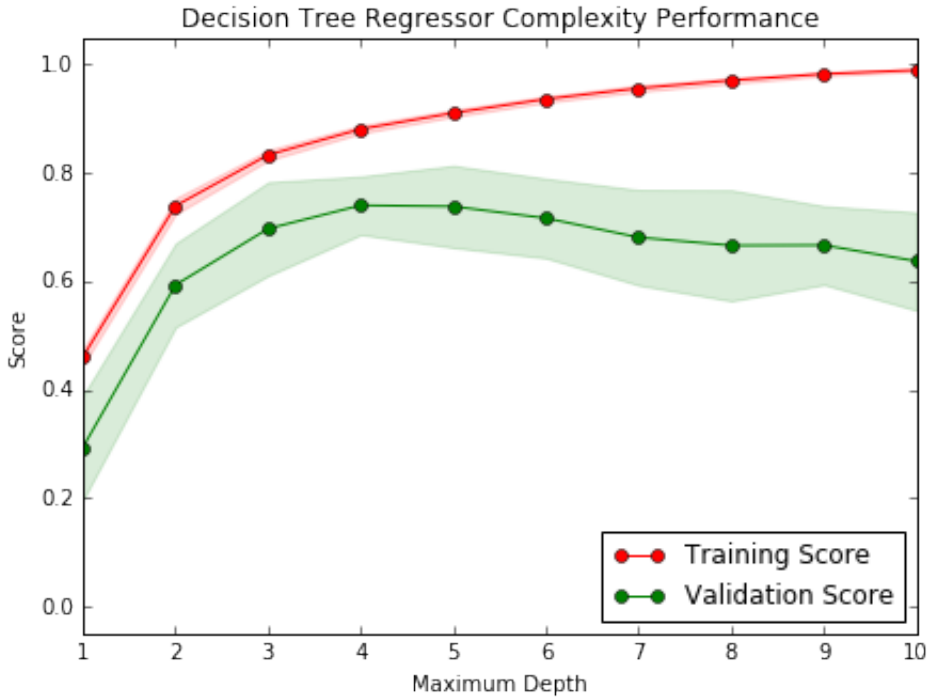
I don't think that more training points will benefit this model, considering the graph with depth equals 3, because it is possible to perceive that the testing curve have already stabilized close to the training curve and that both are close to a reasonable score.

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 [11]:
```
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:

- With **depth equals 1** we have a scenario of high bias (overfimplified). Both curves are presenting a very poor score, under 0.5. This demonstrate that a model trained like this would have not enought accurancy to predict usefull data.
- With **depth equals 10** we are paying too much attention to the data (overfit) this means we have a high variance what is not good, too, as it will not generalize well. The visual cues of that are that at around depth 4 the curves are behaving in a predictive way and after that, after we increase the depth the curves diverge and we have a much higher error on validation curve than on traninig curve, what is not what we are looking for.

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: I would say something about 3 or 4 as depth, my intuition leads me to choose 4, as I saw depth 3 decision tree regressor performance scores about 0.8, as the complexity curve remains stable for depth 4, maybe we can achieve scores highers than 0.8 using 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: Grid Search is a generalization of the traditional search technique that seeks for an optimal hyperparameter that when applied to a given model produces the best predictions (avoiding, as possible, data overfitting).

The big deal about Grid Search is that it can be applied for tuning the models that interact with with more than one hyperparameter, creating complex tuples that when are all combined we give them the name of "grids".

This techinque will loop thru around all set of hyperparameters combinations comparing the effectiveness of the model, by the end it will find the set of parameters that produces the best prediction model (avoiding, as possible, overfitting the data).

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: K-fold cross-validation is an evolution of the simple cross-validation that splits data in testing and training datasets. On **K-fold** we divide the dataset into k subsets and we perform cross-validation using k subset as testing and the rest of the k-1 subsets becomes the training data. We then run this **k-times** to calculate the averages. Using k-fold we can use the entire dataset, but it costs more if we take in count the processing time.

The main benefit of K-fold is that its cross-validation method is really good to avoid data overfit, because we can validate the model effectiveness using different subsets of training and testing data **k-times**. Adding this technique, that **reduces the overfitting**, as part of the **Grid Search** technique help us to get the best set of hyperparameters that will really generalizes the model, avoiding the model **overtuning** (that is when we the set of parameters we found does not generalize well the entire model, as it was just hooked by the validation set).

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) 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) 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 [12]:
```
# TODO: Import 'make_scorer', 'DecisionTreeRegressor', and 'GridSearch
CV'
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_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 r
ange from 1 to 10
    params = {'max_depth': range(1,11)}

    # TODO: Transform 'performance_metric' into a scoring function usi
ng '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 mo
del
    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 [13]:
```
# 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 5 for the optimal model.
```

Answer: The max_depth is 5, I said on question 6 that my intuition was 4 as it was apparently better than 3 and 6 per our learning curves. But now, using the fit and other techniques it showed to be 5.

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 [28]:
```
# 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}".for
mat(i+1, price)
```

```
Predicted selling price for Client 1's home: $431,025.00
Predicted selling price for Client 2's home: $166,350.00
Predicted selling price for Client 3's home: $879,900.00
```

Answer: Considering:
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
ps.:Considering that the Mean and Median are very close, I'm seeing this as an almost normal distribution.

Home Features
RM (#room): expected to increase the price
LSTAT (poverty level): expected to decrease the price
PTRATIO (Student-Teacher ratio): expected to decrease the price

Client 1's home: $431,025.00 **(recommended)** This home price seems to be very reasonable because there are very good schools. Furthermore the price is compatible with the market and the 5 rooms are fair for this price, not forgetting that the neighborhood is not bad at all.

Client 2's home: $166,350.00 **(recommended)** This home price seems, at first, very low. But it is still far from the Minimum price of our dataset. This price is between the second and the first standard deviation, below the median, what makes this price still acceptable. By checking its features we can see that the level of poverty is high and that Student-teacher ration is also bad. Considering this I would, also, trust in the algorithm prediction.

Client 3's home: $879,900.00 **(recommended)** This home price seems, at first, very high. But when we saw that number of rooms (that is rare these days), this very nice neighborhood and those very selective and good schools, I think the home owner should try to sell it with this price.

:)

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 [31]:
```
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 believe that the constructed model is very interesting, but still need to be better validated. We could see that our model is still too sensitive by presenting price variations at a range of $73,357.29 on predictions.

It is also important to notice that some society values changed since 1978, as the economy, the industry and the immigration, for instance. Just considering the RM, LSTAT and PTRATION as variables to our model definetly is something to revalidate. I'm not saying that we need to add more variables, but that we need to be sure about the ones we are using on our model. I believe that home price predictions should take in count also proximity to business centers/jobs, hospitals and malls, for example.

Understanding how humans associate a price to an asset, is more complex than just think in superficial items like rooms or poverty. And I also believe that our model, as-is, would not work for rural cities, by the same reason.

Based on that I think this mode should not be used in a real-world.