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 [4]:

```
# 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.
```

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 [9]:

```python
# 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: ${}".format(minimum_price))
print("Maximum price: ${}".format(maximum_price))
print("Mean price: ${}".format(mean_price))
print("Median price ${}".format(median_price))
print("Standard deviation of prices: ${}".format(std_price))
```

```
Statistics for Boston housing dataset:

Minimum price: $105000.0
Maximum price: $1024800.0
Mean price: $454342.944785
Median price $438900.0
Standard deviation of prices: $165171.131544
```

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: This problem can phrased using examples like below.

- Would you expect a home that has an `'RM'` value(number of rooms) of 6 be worth more or less than a home that has an `'RM'` value of 7?
- Would you expect a neighborhood that has an `'LSTAT'` value(percent of lower class workers) of 15 have home prices be worth more or less than a neighborhood that has an `'LSTAT'` value of 20?
- Would you expect a neighborhood that has an `'PTRATIO'` value(ratio of students to teachers) of 10 have home prices be worth more or less than a neighborhood that has an `'PTRATIO'` value of 15?

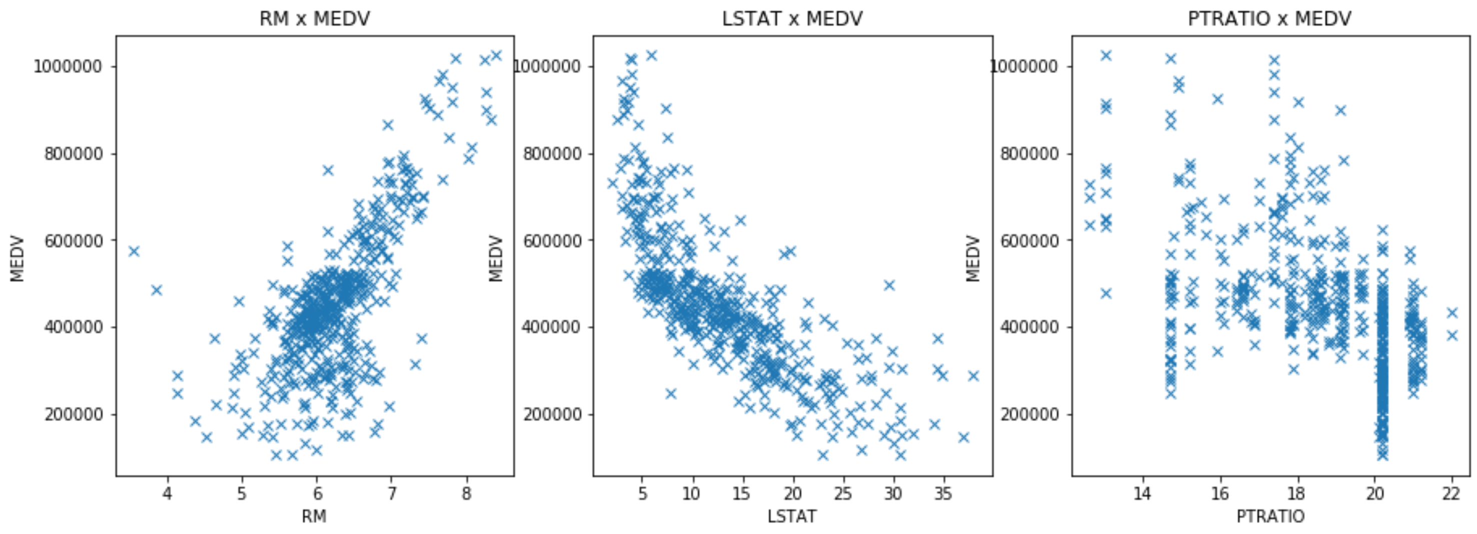
Answer:

- I would think a home with more rooms `'RM'` would have a greater value of `'MEDV'` than one with less rooms. The value of a property is often related to the size of the property itself.
- I would think a neighborhood with a greater percentage of "lower class" `'LSTAT'` homeowners would have a lower value of `'MEDV'`. The ability to buy a house is going to be related to that person's ability to replay the loan. As such I would expect lower income families will purchase properties in areas where they can afford to do so and be granted a mortgage.
- I would think a neighborhood with a greater percentage of students to teachers `'PTRATIO'` would have a lower median house price `'MEDV'`. Teachers may prefer to work in areas close to where they live. As teachers will normally earn an income larger than those considered `'LSTAT'` 'working poor' this will affect their location of work. As such the number of teachers working in areas of cheaper housing I would expect to be less and affect the `'PTRATIO'`

The charts below show the MEDV prices compared to each of the features and expectations proposed match. However; the PTRATIO does appear to have some margin for difference. There are still areas with a high MEDV value where the PTRATIO is between 13 and 19, though below this ratio the MEDV value does drop below 600,000. Similarly there are areas with a lower MEDV when the PTRATIO is low. Additionally the lowest MEDV values are where the PTRATIO is around 20, though at this number the MEDV values do vary from lowest to values significantly above mean. As such it would suggest PTRATIO may have less importance to MEDV prices when taken on its own.

In [10]:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(15, 5))
for i, col in enumerate(features.columns):
    plt.subplot(1, 3, i+1)
    plt.plot(data[col], prices, 'x')
    plt.title('%s x MEDV' % col)
    plt.xlabel(col)
    plt.ylabel('MEDV')
```



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 [11]:

```
# 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

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

```
In [12]:
```

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

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

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

Hint: The R2 score is the proportion of the variance in the dependent variable that is predictable from the independent variable. In other words:

- R2 score of 0 means that the dependent variable cannot be predicted from the independent variable.
- R2 score of 1 means the dependent variable can be predicted from the independent variable.
- R2 score between 0 and 1 indicates the extent to which the dependent variable is predictable. An
- R2 score of 0.40 means that 40 percent of the variance in Y is predictable from X.

Answer:

- The result shows a high R^2 coefficient for the Prediction and True values provided and shows a high `goodness of fit`. For this particular set of results, the independent variables used in creating the prediction account for 92.3% of the variation of the predictions.
- However; the number of predicted results is not particularly large, what confidence can we have that further predictions will be accurate? An R^2 score from measuring 5 predictions may not be sufficient to make a solid conclusion. Similarly; what the model is trying to predict will determine what score is acceptable. We may find that in some instances a higher score is required, or even a lower score is acceptable. This would be based on its real-world application and requirements.
- The R^2 result only tests against this particular test case. It may be that there are other models or features which could improve the score.

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 [13]:
```

```
# TODO: Import 'train_test_split'
from sklearn.model_selection 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=42)

# 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: Think about how overfitting or underfitting is contingent upon how splits on data is done.

Answer:

By splitting the data into a train and test set we help reduce the likelyhood of overfitting. Overfitting could occur if we trained against all the data using particular model parameters. We could have our model potentially memorising training data rather than using it to find patterns, with no independent test data available to confirm this. Keeping back some of the data to test independently of the training data can help provide validation of the model. As such, comparing results of training and test data is essential to help review the model for high bias/high variance. This is inline with the `Golden rule` being to not just test a model on the data you used to train.

We also have to be mindful that the data *may* need to be shuffled before splitting the data. This will help ensure that there is no inherent bias if using a specific block of data from a specific range.

Splitting data with 80% training 20% testing provides a good ratio for training and testing. The 80% of the data can be further trained/tested using cross-validation (such as k-fold), and still keep the final 20% for final testing.

If we use a smaller percentage of data for the training set, it could lead to our model potentially underfitting i.e. providing an over-simplistic model because there was simply not enough data to train the model. This could provide a model which may not predict accurately enough, with the larger test set not providing a benefit.

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.

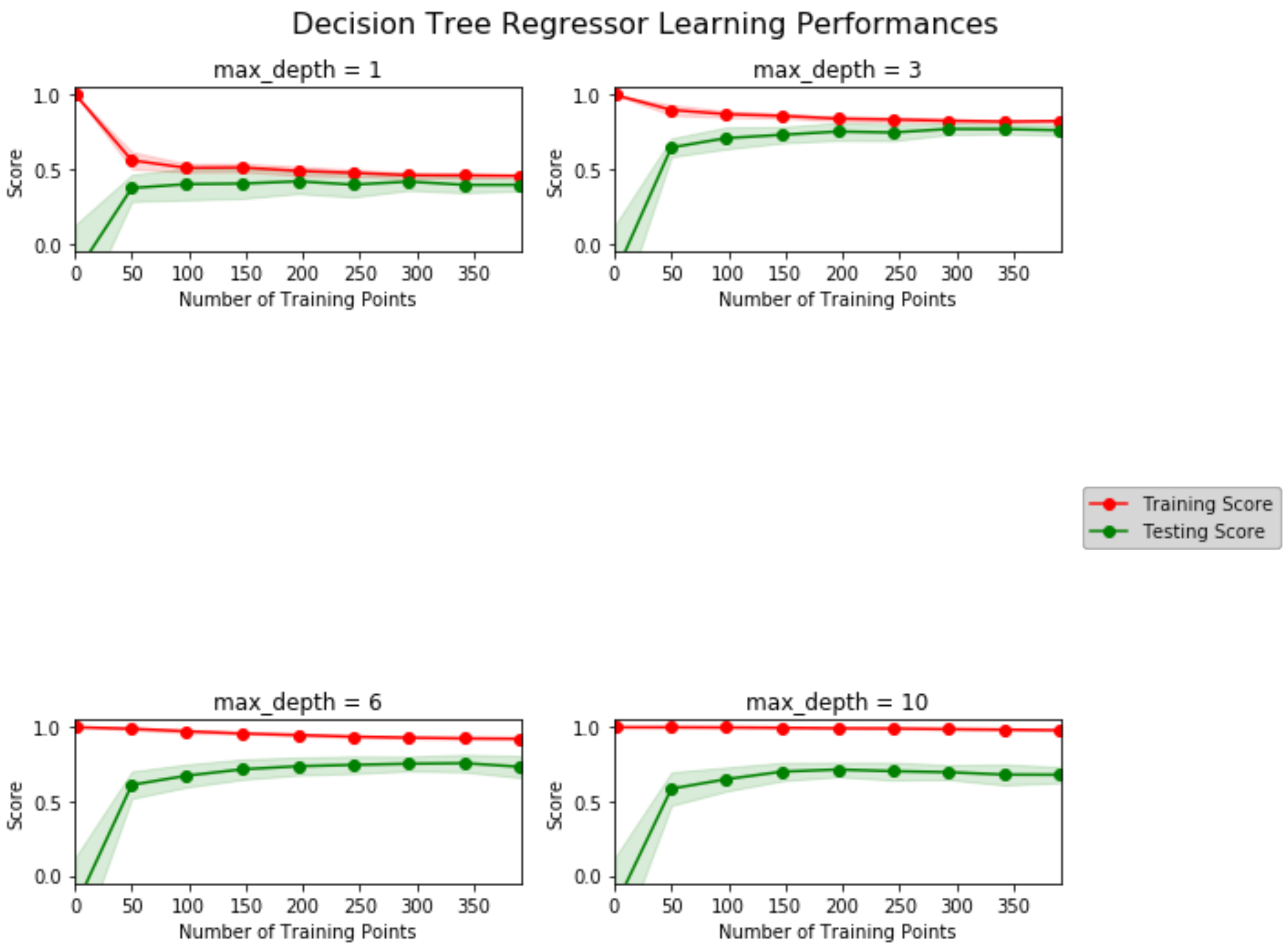
Learning Curves

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 [14]:

```
# 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? Generally speaking, the more data you have, the better. But if your training and testing curves are converging with a score above your benchmark threshold, would this be necessary? Think about the pros and cons of adding more training points based on if the training and testing curves are converging.

Answer:

The first graph uses a `'max_depth'` of 1. As such, the Descision Tree can only differentiate between one of two states.

As the number of training points increases, both the training and testing scores converge towards a value on 0.5 but do not significantly change as the number of training points increases beyond 100. When comparing this alongside the `uncercainty` margin of the scores, it shows little improvement would be gained by increasing the number of training points beyond this.

Because of the low `'max_depth'` value, training and test scores of around 0.5 show only 50% percent of the outcomes could be predicted by the model (no better than flipping a coin).

It is unlikely having even more training points beyond this would benefit this model unless there was a significant change to the data that was not seen in the previous training points. The increase in training data would increase processing time with no benefit.

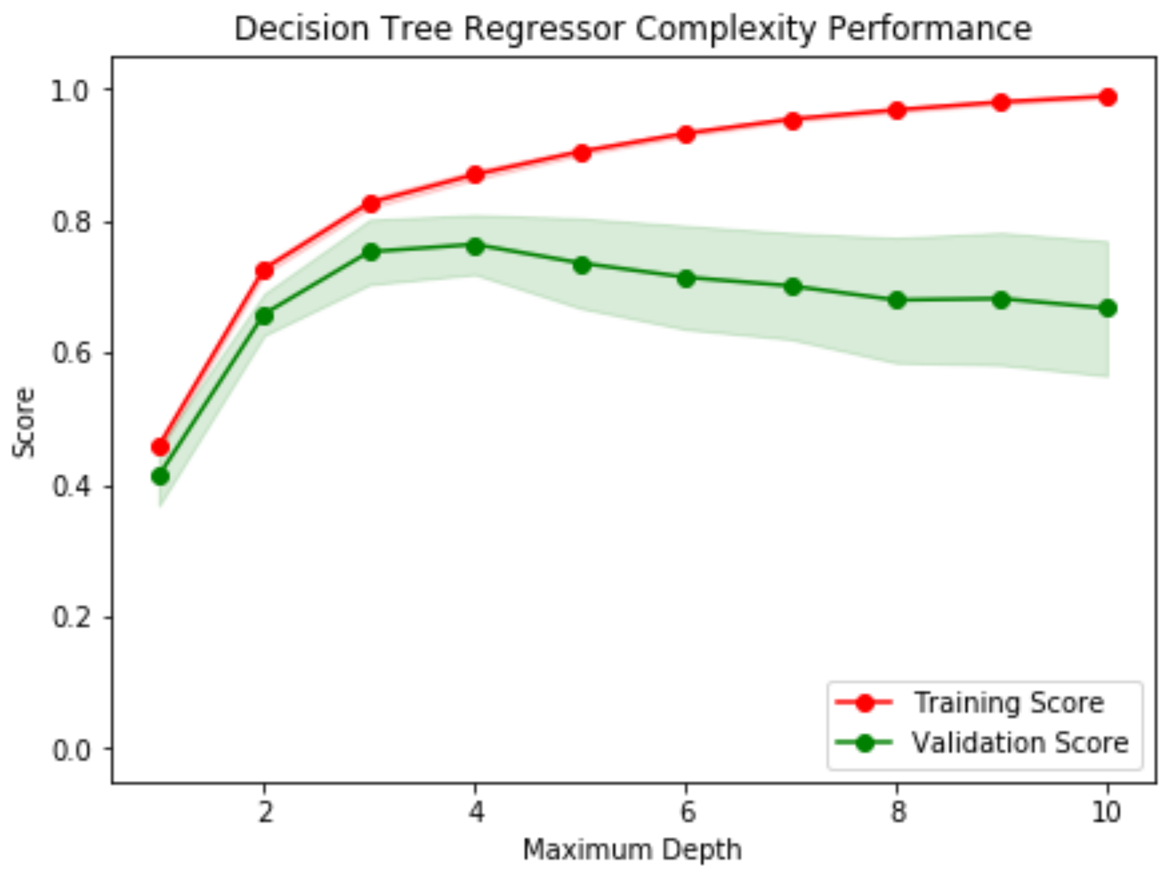
Complexity Curves

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 Q5 and Q6.

In [15]:

```
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: High bias is a sign of underfitting(model is not complex enough to pick up the nuances in the data) and high variance is a sign of overfitting(model is by-hearting the data and cannot generalize well). Think about which model(depth 1 or 10) aligns with which part of the tradeoff.

Answer:

- With a `maximum depth` of 1 the model suffers from high bias. Because it can only differentiate between two states, it is underfitting the data. Both training and validation scores are similar and poor.
- With a `maxumum depth` of 10 the model suffers from high variance. The training score increases but the validation score is reduced. The difference between the two scores increases and the model looks to be overfitting to the training data. As the depth increases the `uncertainty` of the validation score increases.

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?

Hint: Look at the graph above Question 5 and see where the validation scores lie for the various depths that have been assigned to the model. Does it get better with increased depth? At what point do we get our best validation score without overcomplicating our model? And remember, Occams Razor states "Among competing hypotheses, the one with the fewest assumptions should be selected."

Answer:

- Looking at the chart it appears that the optimal value for `maximum depth` would be 3
- This value offers a close convergence between Testing and Validation scores.
- The validation score at this depth is similar to that when using maximum depth of 4. However; a maximum depth of 4 shows that the Training data is starting to overfit. Training and Validation scores are diverging.
- The additional complexity of increasing `maximum depth` to 4 provides no benefit.

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?
- How it can be applied to optimize a learning algorithm?

Hint: When explaining the Grid Search technique, be sure to touch upon why it is used, what the 'grid' entails and what the end goal of this method is. To solidify your answer, you can also give an example of a parameter in a model that can be optimized using this approach.

Answer:

Grid Search allows you to provide various models to apply your data to in order to see which one provides the most useful insight. Similarly the individual parameters of each model can be adjusted and tested to see which parameter settings provide the optimal results. This optimal result can be measured as a score such as r^2, F1 etc. depending what you want to use as the score measure.

For example; with a descision tree classifyer, there are parameters such as max_depth, min_samples_leaf and min_samples_split available to optomise your model. If you wanted to know for a particular train,test data set which parameters produce the best outcome you could try these all manually by hand, or place them into a grid search to run through without the manual effort.

Below shows an example of the values in each parameter which would be tested over the various sequences. {'min_samples_leaf':[2,4,8,16,20], 'min_samples_split':[2,4,6,8,10],'max_depth':[2,4,8,10]}

Once the grid search is completed you can then use the best estimator grid search found, and see the parameter settings it used to reach the score it did.

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: When explaining the k-fold cross validation technique, be sure to touch upon what 'k' is, how the dataset is split into different parts for training and testing and the number of times it is run based on the 'k' value.

When thinking about how k-fold cross validation helps grid search, think about the main drawbacks of grid search which are hinged upon **using a particular subset of data for training or testing** and how k-fold cv could help alleviate that. You can refer to the docs (http://scikit-learn.org/stable/modules/cross_validation.html#cross-validation) for your answer.

Answer:

With a traditional Train/Validation/Test split you are limiting the amount of data available to each of these three groups since the data for each set is only used the once. This can vastly reduce the amount of data that can be used.

`k-fold` cross-validation training allows you to extend the use of your data so the Train and Validation data can be treated as one. Thus only needing to remove a portion of data for final testing. This is acheived by creating multiple data sets or `'folds'` of the same data. A model is trained using k-1 of the folds specified as training data and the remainder for the validation. The testing/validation is run for each of the folds. It is then possible to take an average score from across the folds to help determine the usefulness of the model.

For Example, using a data set of 12. If 4 folds are specified, the data will be split into 4 combinations. With k-1 of the folds being used for training, that provides 9 blocks of data for Training and 3 blocks for Validation. Since all data is used in each set.

'TRAIN:', array([3, 4, 5, 6, 7, 8, 9, 10, 11]), 'VALIDATE:', array([0, 1, 2])

'TRAIN:', array([0, 1, 2, 6, 7, 8, 9, 10, 11]), 'VALIDATE:', array([3, 4, 5])

'TRAIN:', array([0, 1, 2, 3, 4, 5, 9, 10, 11]), 'VALIDATE:', array([6, 7, 8])

'TRAIN:', array([0, 1, 2, 3, 4, 5, 6, 7, 8]), 'VALIDATE:', array([9, 10, 11])

Applying cross-validation alongside grid search, it is possible to ensure the results of a grid search are not just fitting to one set of training data, but that the results are consistent across multiple train/validates sets created by k-fold cross validation. If grid search is only run against one set of training data it is more difficult to ensure the model is working consistently.

One drawback to consider with k-fold is that you are now running train/validation multiple times. Additionally when combined with grid search running multiple parameters, this could affect processing time when determining the optimum model.

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) 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 [16]:

```
# TODO: Import 'make_scorer', 'DecisionTreeRegressor', and 'GridSearchCV'
from sklearn.metrics import make_scorer
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor

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

    # Create cross-validation sets from the training data
    # sklearn version 0.18: ShuffleSplit(n_splits=10, test_size=0.1, train_siz
e=None, random_state=None)
    # sklearn versiin 0.17: ShuffleSplit(n, n_iter=10, test_size=0.1, train_si
ze=None, random_state=None)
    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 cv object --> GridSearchCV()
    # Make sure to include the right parameters in the object:
    # (estimator, param_grid, scoring, cv) which have values 'regressor', 'par
ams', 'scoring_fnc', and 'cv_sets' respectively.
    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 [20]:
```

```
# 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.
```

Hint: The answer comes from the output of the code snipped above.

Answer:

Parameter 'max_depth' is 4 for the optimal model.

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. Of the three clients, client 3 has has the biggest house, in the best public school neighborhood with the lowest poverty level; while client 2 has the smallest house, in a neighborhood with a relatively high poverty rate and not the best public schools.

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

In [21]:

```
# 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: $403,025.00
Predicted selling price for Client 2's home: $237,478.72
Predicted selling price for Client 3's home: $931,636.36
```

Answer:

Predicted selling price for Client 1's home: $403,025.00

Predicted selling price for Client 2's home: $237,478.72

Predicted selling price for Client 3's home: $931,636.36

The following information was pulled from the `Data Exploration` section of the project:

Minimum price: $105000.0

Maximum price: $1024800.0

Mean price: $454342.944785

Median price $438900.0

Standard deviation of prices: $165171.131544

Additionally, we find the following values with the Maximum MEDV of $1024800.0

RM 8.398 LSTAT 5.910 PTRATIO 13.000 MEDV 1024800.000

For the Minimum MEDV of $105000.0 we find the following values

RM 5.453 LSTAT 30.590 PTRATIO 20.200 MEDV 105000.000

Looking at the features of the clients against the predicted prices, Client 1 and 3's predictions appear reasonable when compared to to the feature values given for the Highest and Lowest MEDV prices. Client 2's estimated price initially appears to be overpriced. The client 2's house has less rooms than the house with the lowest MEDV and the LSTAT and PTRATIO values are very similar. I believe the closeness in values does not necessarily justify the price being over twice the price of the cheapest house, even though the price is within one standard deviation.

Possibly the scaling of the features would need to be considered to weight some features over others. e.g. would a higher weighting for rooms be more effective estimator?

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 respect to the data it's trained on.

In [22]:

```
vs.PredictTrials(features, prices, fit_model, client_data)
```

```
Trial 1: $391,183.33
Trial 2: $419,700.00
Trial 3: $415,800.00
Trial 4: $420,622.22
Trial 5: $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: $69,044.61
```

Question 11 - Applicability

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

Hint: Take a look at the range in prices as calculated in the code snippet above. Some questions to answering:

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

Answer:

The model could provide a general overview however it would not be suitable in a current real-world setting.

- Though the data has been adjusted for inflation, it may not necessarily be enough for real-world estimates. House prices can fluctuate and exceed inflation rates if there is a particular demand for housing in specific locations or if the neighborhood itself is updated. In this respect relying purely on inflation for house price increases would not be sufficient.
- As testing shows in section 10, depending on the data provided there is a significant variance in house price predictions. This variation is a significant difference between estimates.
- In general house prices do tend to stay within a particular range for specific neighborhoods. However; this does not include exceptions or outliers. For example a swimming pool, or increased room numbers, larger plot sizes which are outside the norm for the area could have an affect on the house price.
- The data itself would be very specific to the area it was taken. The same model could not be applied to other locations. For example a rural location would have other features which would need to be taken into consideration, and others which would not be as useful. For example the PTRATIO in a rural setting may not be of use, but land/plot size may be a feature to consider.

> **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.