boston_housing

October 25, 2017

1 Machine Learning Engineer Nanodegree

1.1 Model Evaluation & Validation

1.2 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.

1.3 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. 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
        # Pretty display for notebooks
        %matplotlib inline
        import matplotlib
        import numpy as np
        import pandas as pd

        # Import supplementary visualizations code visuals.py
        import visuals as vs

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

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

```
C:\Users\Aswin\Anaconda3\lib\site-packages\sklearn\cross_validation.py:44: DeprecationWarning:
  "This module will be removed in 0.20.", DeprecationWarning)
C:\Users\Aswin\Anaconda3\lib\site-packages\sklearn\learning_curve.py:23: DeprecationWarning: Th
  DeprecationWarning)


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

1.4 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.

1.4.1 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 per-

form 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 [2]:
```
# 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
```

1.4.2 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?

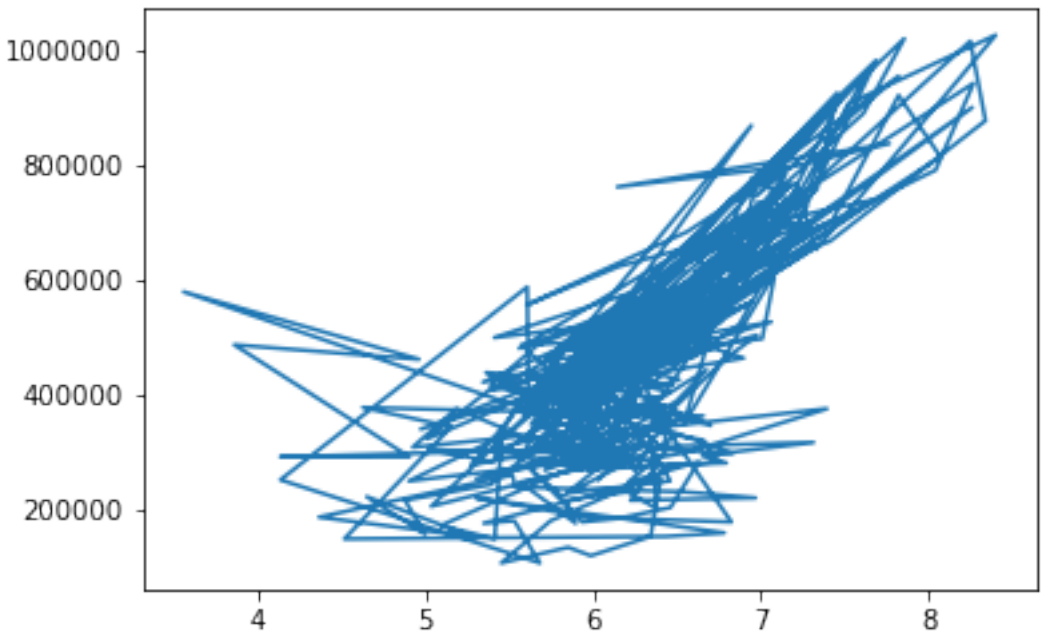
Answer: #### Increasing RM value increases the facilities of the housing which improves the usage of the house, increasing the price value

Increasing LSTAT value will affect the buying behaviour as the net income of the households decrease. This decreases the demand for higher valued houses. So the pricing for the houses decrease.

Increasing the supply for houses due to an increase in PTRATIO would not be an ideal solution because the primary and secondary schools last only for 10/11 years which keep on shifting for another rental property holder or another household. It is observable that the number of teachers would not increase unless the school's financial situation is affected by a decreasing rate. Since the demand for housing gets satisfied due to an increase in the households caused due to the increase in students, the value of the houses should increase to satisfy the demand without any change in the supply.

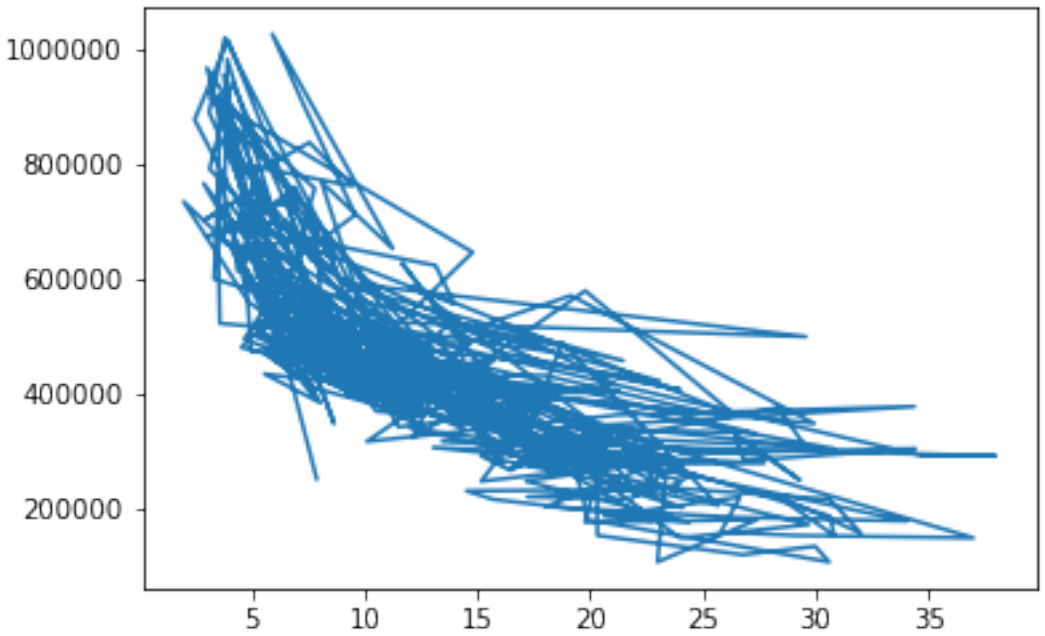
```
In [3]: import matplotlib.pyplot as plt1
        import matplotlib.pyplot as plt2
        import matplotlib.pyplot as plt3

        # Plotting the room numbers
        plt1.plot(data['RM'].values, prices.values)
```

```
Out[3]: [<matplotlib.lines.Line2D at 0x15711ba57f0>]
```



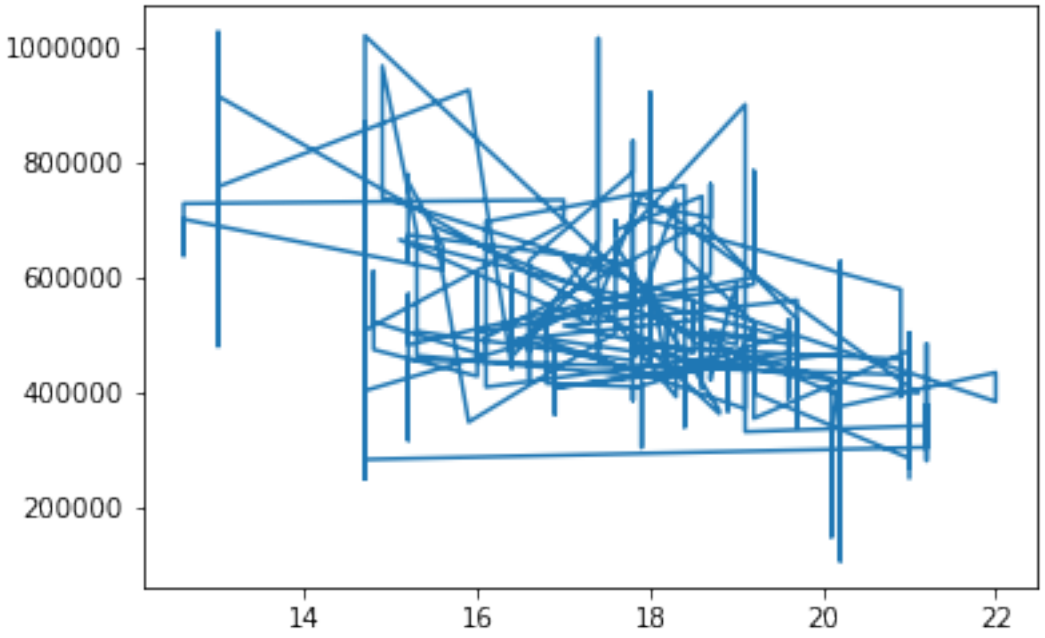
In [4]:
```
# Plotting the LSTAT
plt2.plot(data['LSTAT'].values, prices.values)
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x15711c74e10>]
```



In [5]:
```
# # Plotting the PTRATIO
plt3.plot(data['PTRATIO'].values, prices.values)
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x15712e23898>]
```



1.5 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.

1.5.1 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*, R2, 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 R2 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 R2 of 0 is no better than a model that always predicts the *mean* of the target variable, whereas a model with an R2 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 R2 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 [6]: # 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
```

1.5.2 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 [7]: # 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: #### An R^2 score of 0.923 would imply the precision of prediction is high. Since the variability is related to the closeness of the predicted value and actual value, the model seems to have been captured the variation of the target variable. #### The simplicity of fit is not considered here as we may need to know the feature specific values for Recall and Precision

1.5.3 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 [8]: # TODO: Import 'train_test_split'
        from sklearn.cross_validation import train_test_split

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

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

```
Training and testing split was successful.
```

1.5.4 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: #### The training data that you come up with will not be overfitted due to the presence of large number of data, and randomisation would enable some of the data to be considered for exclusions #### The test data would supply an estimate of how the training data should be analysed

---

1.6 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.

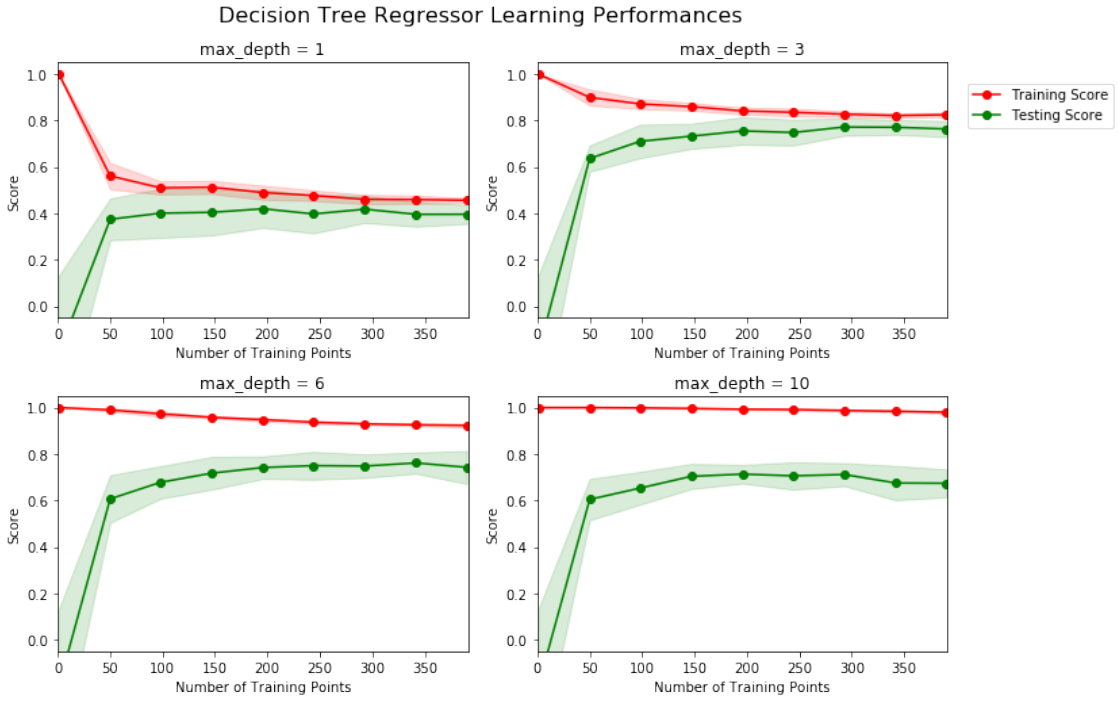
1.6.1 Learning Curves

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 R2, the coefficient of determination.

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

```
In [9]: # Produce learning curves for varying training set sizes and maximum depths
        vs.ModelLearning(features, prices)
```



1.6.2 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:

The chosen graph is the graph of the model with maximum depth of 10. The score of the training curve becomes optimal when more training points are added. The graph of the learning curve becomes a more steadily constant curve.

The score of the testing curve increases and reaches an optimal condition when more training points are added. This increases the accuracy of the model. The benefit is that the validation of the model becomes powerful.

The learning curves converge to scores of 0.99 and 0.6 for training and testing curves. Compared to the learning model presented in the other figures such as the maximum depth = 6 and maximum depth = 3, the maximum_depth = 10 model overfits the curve because the training set achieves the optimality as early as possible and seems to have memorized the results such that it produces these values at a depth of 10.

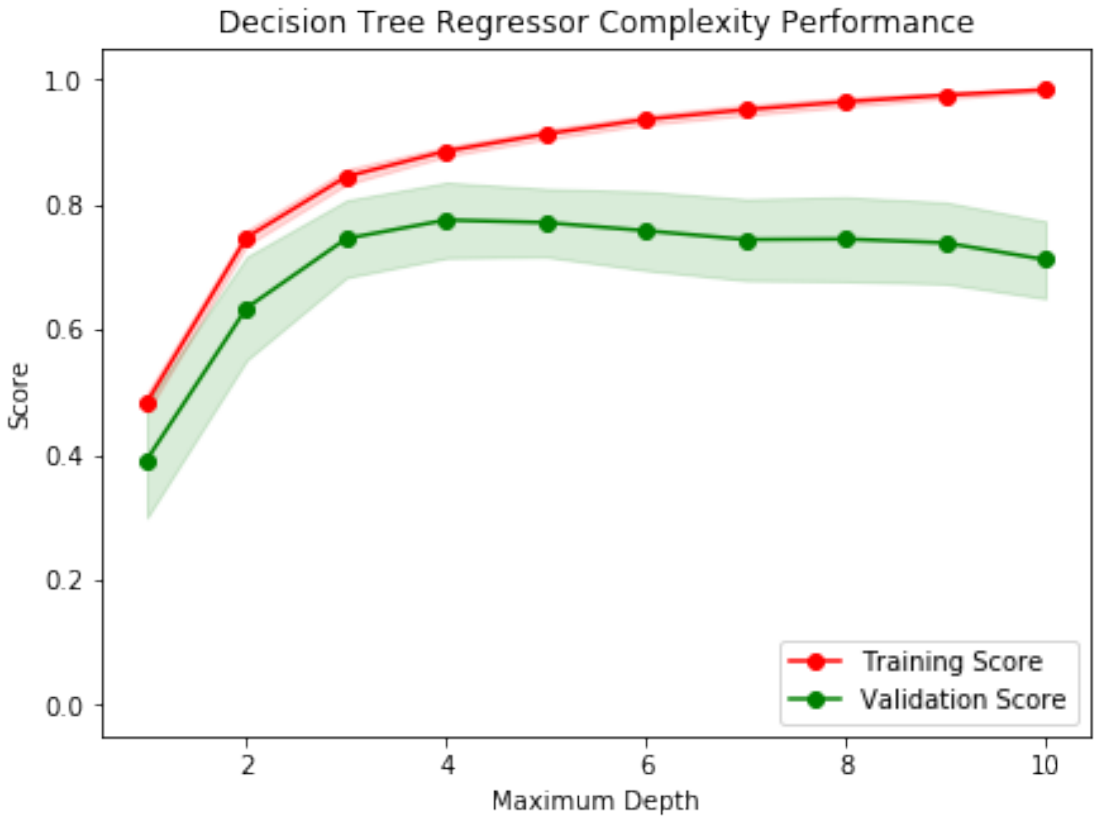
The model shows an overfitting model. The testing curve has less accuracy compared with the other models in maximum_depth = 3 and maximum_depth = 6. Thus implies there are errors when the dataset is changed. A more convenient model would be thise with depths 3 and 6 as per the Learning Curve.

1.6.3 Complexity Curves

The following code cell produces a graph for a decision tree model that has been trained and validated on the training data using different maximum depths. The graph produces two complexity curves — one for training and one for validation. Similar to the **learning curves**, the shaded regions of both the complexity curves denote the uncertainty in those curves, and the model is scored on both the training and validation sets using the `performance_metric` function.

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

```
In [10]: vs.ModelComplexity(X_train, y_train)
```

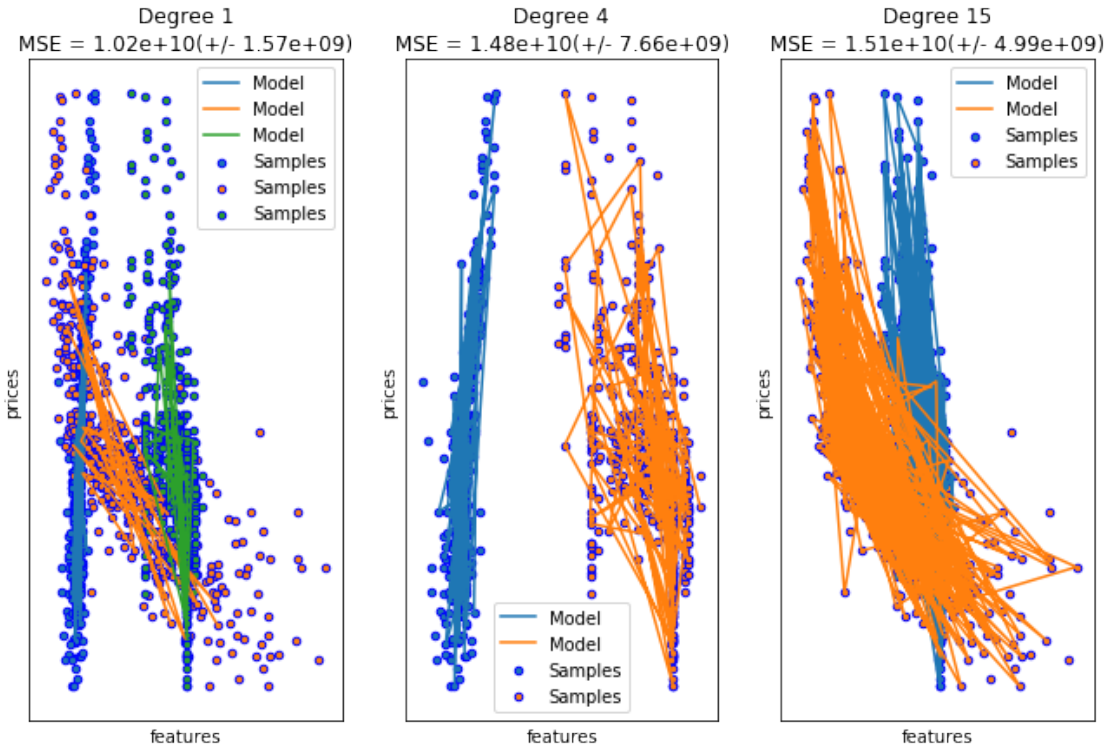


1.6.4 Question 5 - Bias-Variance Tradeoff

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?

```
In [11]: import plot_overfitting as plotting
         import matplotlib.pyplot as plt2

         plt2.figure(figsize=(11, 7))

         plotting.execute(plt2, features, prices)
```



Explanation of plots

The degree 1 plot, is a polynomial and decision tree regressor analogical to depth = 1, is a case of underfitting. The mean square error is high and is termed as high bias because corresponding to the features use there is no enough integration of prediction algorithms.

The degree 4 plot, is a polynomial and decision tree regressor analogical to greater depth, but better test train split (test_size = 0.20), is a case of moderately fitting of a curve but less features have been used hence, this is a case of high variance.

The degree 15 plot, is a polynomial and decision tree regressor analogical to higher depth, is a case of overfitting the curve (with test_size=0.75) because less features have been used and degree is reklatively high and there is high variance.

Answer: #### When the model is trained with maximum depth of 1, the model will suffer from High Bias as it is a case of Underfitting. It is underfitting because the lower the maximum depth the patterns of training set and testing set vary according to the presence of unprecise values in the training set. Thereby, showing the R2 score as low for a maximum entry data point. #### When the model is trained with maximum depth of 10, the model will suffer from High Variance as it ia case of Overfitting. It is Overfitting because the higher the maximum depth, the patterns

of training and testing set show variability and high precision and tend to match the exact data points when the curve is fit.

1.6.5 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:

Best guess optimal model is where the validation score is the highest. Here the best guess optimal model will be at: - maximum depth = 4 - score ~ 0.80

The score of training set is approximately 90% and the score of test set is approximately 80%.

1.7 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`.

1.7.1 Question 7 - Grid Search

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

Answer: #### The grid search technique considers all types of combinations for the parameters provided to the algorithm. These parameters are then mapped in a grid, which are then analysed to calculate a score to determine the best possible combination for an optimal model.

1.7.2 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:

A K-Fold cross validation technique splits the dataset into k folds. Out of the k folds, k - 1 folds are used for the training set and the remaining data is used for validation.

The test set verifies the score and the training set fits the curve.

The K-Fold technique allows shuffling of dataset before splitting into batches which ensures there is an unbiased distribution. One drawback of the K-Fold technique is if the dataset size has been changed marginally, the K-Fold technique fails to predict the outcomes because unseen data that enters the system makes the algorithm less susceptible to change and less accurate.

The integration of Grid Search Cross Validation allows the validation to be split based on varying folds of train and test data based on a CV iterable. Stratified KFold is a possible technique which can be applied using GridSearchCV.

The optimization model will be a pure model evaluated based on the accuracy score and performance score.

1.7.3 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` 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` 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` 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 'GridSearchCV'
         from sklearn.metrics import make_scorer
         from sklearn.tree import DecisionTreeRegressor
         from sklearn.model_selection import GridSearchCV
         from sklearn.model_selection import ShuffleSplit
         import numpy as np

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

             # Create cross-validation sets from the training data
             cv_sets = ShuffleSplit(n_splits=10, test_size=0.10, random_state=0)

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

             # TODO: Create a dictionary for the parameter 'max_depth' with a range from 1 to
             params = {'criterion': ['mse', 'mae'], 'splitter': ['best', 'random'], 'max_depth

             # 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(estimator=regressor, param_grid=params, cv=cv_sets, scoring=sc

    # 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_
```

1.7.4 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.

1.7.5 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()['ma
```

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

Answer:

Parameter 'max_depth' is 4 for the optimal model

The best guess model has predicted the same results compared with the GridSearchCV results.

1.7.6 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

Feature	Client 1	Client 2	Client 3
Neighborhood poverty level (as %)	17%	32%	3%
Student-teacher ratio of nearby schools	15-to-1	22-to-1	12-to-1

What price would you recommend each client sell his/her home at? Do these prices seem reasonable given the values for the respective features?
Hint: Use the statistics you calculated in the **Data Exploration** section to help justify your response.

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

```
In [14]: # Produce a matrix for client data
         client_data = [[5, 17, 15], # Client 1
                        [4, 32, 22], # Client 2
                        [8, 3, 12]]  # Client 3

         # Show predictions
         for i, price in enumerate(reg.predict(client_data)):
             print("Predicted selling price for Client {}'s home: ${:,.2f}".format(i+1, price))

         features.describe()
```

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

```
Out[14]:                RM       LSTAT     PTRATIO
         count  489.000000  489.000000  489.000000
         mean     6.240288   12.939632   18.516564
         std      0.643650    7.081990    2.111268
         min      3.561000    1.980000   12.600000
         25%      5.880000    7.370000   17.400000
         50%      6.185000   11.690000   19.100000
         75%      6.575000   17.120000   20.200000
         max      8.398000   37.970000   22.000000
```

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

The first data row has an RM value that is lesser than the mean value implying the price would be moderate, but LSTAT is within 1 Standard Deviation, and the price value would still be moderate. The PT Ratio is lesser, hence the price increases. This reflects on the selling price of $391,183.33 which is moderately high value.

The second data row has an RM value that is lesser than the first data row, the lower class working poor LSTAT ratio has increased, implying the prices would still be lower. Considering the primary and secondar school students to teacher ratio, the value seems to be high implying the overall price is relatively low which is reflected ion the model.

The thrid data row is undoubtedly high priced because of high RM value, within the 1st quartile LSTAT value and within the 1st quartile PTRATIO value. This is reflected on the predicted pricing. _____

1.7.7 Sensitivity

An optimal model is not necessarily a robust model. Sometimes, a model is either too complex or too simple to sufficiently generalize to new data. Sometimes, a model could use a learning algorithm that is not appropriate for the structure of the data given. Other times, the data itself could be too noisy or contain too few samples to allow a model to adequately capture the target variable — i.e., the model is underfitted. Run the code cell below to run the `fit_model` function ten times with different training and testing sets to see how the prediction for a specific client changes with the data it's trained on.

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

```
Trial 1: $391,183.33
Trial 2: $424,935.00
Trial 3: $419,300.00
Trial 4: $275,100.00
Trial 5: $382,500.00
Trial 6: $411,931.58
Trial 7: $394,800.00
Trial 8: $407,232.00
Trial 9: $405,300.00
Trial 10: $413,700.00

Range in prices: $149,835.00
```

1.7.8 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: - In 1978, the data containing the PTRATIO must have changed a lot due to the jobs creation and no. of schools and students. The data containing no. of rooms may have only a slight change, so the change in pricing is expected as a gradual increase. The value for LSTAT must show irrelevance during the period from 1978 to 2017.

- The three features represented in this example are not enough as there are other datasets which take around 5-8 features to obtain a price value. A price value can be dependent on the types of houses such as those with different architectural styles and location of the house

- The model shows robustness because of the considerations of optimal results using Cross Validation and Best Guess models and the model must be used in a real world setting.
- An urban city and rural city will differ from land value and thereby the price value. The data collected in an urban city will be applicable and the model will be applicable if the split is done based on an individual city.

  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.