boston_housing

June 20, 2016

1 Machine Learning Engineer Nanodegree

1.1 Model Evaluation & Validation

1.2 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.

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 preoprocessing steps have been made to the dataset: - 16 data points have an `'MDEV'` 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 `'MDEV'` are essential. The remaining **non-relevant features** have been excluded. - The feature 'MDEV' 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 [2]: # 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['MDEV']
        features = data.drop('MDEV', axis = 1)

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

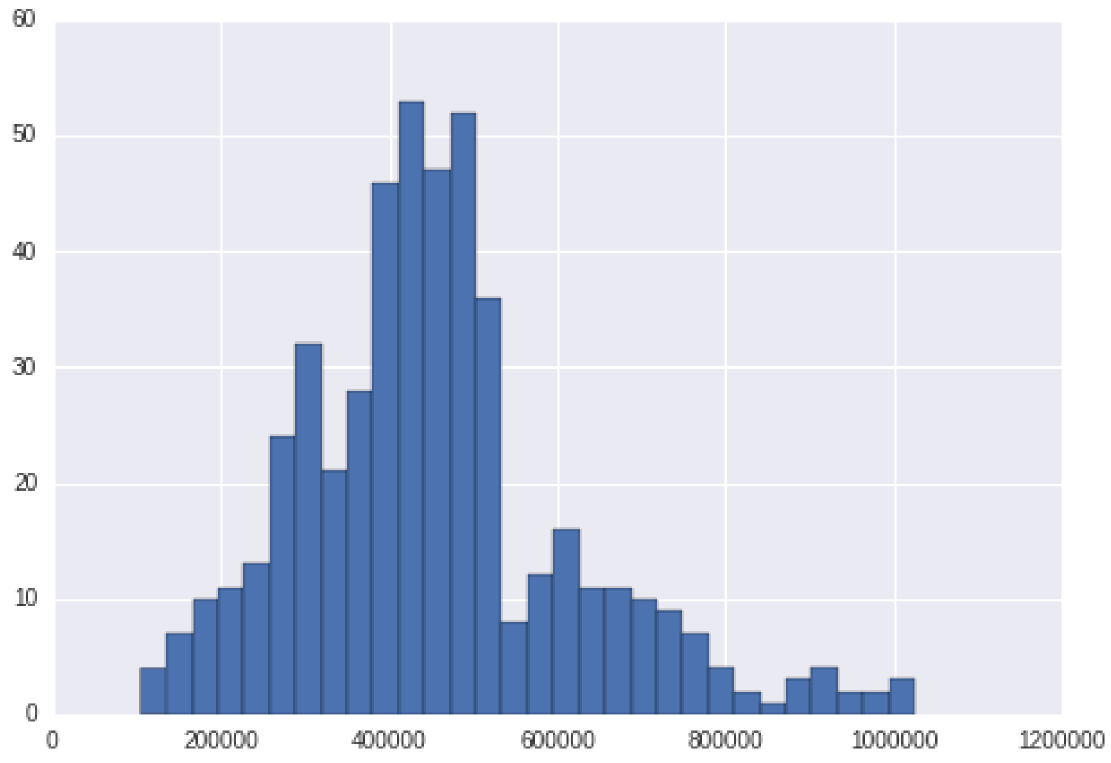
```
Boston housing dataset has 489 data points with 4 variables each.
```

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**, 'MDEV', will be the variable we seek to predict. These are stored in `features` and `prices`, respectively.

```
In [49]: # Visualizing the distribution of home prices in the dataset
         #import matplotlib.pyplot as plt
         import pylab
         a = pylab.hist(prices, bins=30)
```



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 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 `'MDEV'`, which is stored in `prices`. - Store each calculation in their respective variable.

```
In [50]: ## Here we have two good options: either we can call the
         ## numpy routines np.min(), np.max(), np.mean(), np.std()...
         ## or we can call pandas methods
         ## prices.min(), prices.max(), prices.mean(), prices.std()
         ## which are similarly efficient.

         ## I will go with the pandas methods here.

         ## There is only one caveat regarding the std() method.
         ## While the default in numpy is to assume the degrees
         ## of freedom to be zero (ddof=0), it is the one in case
         ## of pandas by default. We have to explicit about what we
         ## want exactly when calling to the method.
```

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

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

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

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

# TODO: Standard deviation of prices of the data
std_price = prices.std(ddof=0)

# 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 all Boston homeowners who have a greater net worth than homeowners in the neighborhood. - `'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 `'MDEV'` or a **decrease** in the value of `'MDEV'`? 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:

Increase in **RM**, the average number of room among homes in the neighborhood, will probably lead to **increase** in the price of the home as it is positively correlated with the overall size of the homes in the neighborhood and hence the cost it took to build the home. On the other hand it may be also indicative of **dicrease** of the price of the home as it may be indicative of low price of land in the area which may be more important than the cost of the materials that went into the physical buidling.

Increase in **LSTAT**, the percentage of all Boston homeowners who have a greater net worth than homeowners in the neighborhood, will certainly be predictive of **decrease** in the price of the home as it will be indicative of how well off the neighborhood is. The gentrified – high price – neighborhoods will have very low LSTAT while poorer neighborhoods will have higher LSTAT values.

Increase in **PTRATIO**, the ratio of students to teachers in primary and secondary schools in the neighborhood, will probably be predictive of **decrease** in the price of home in the neighborhood, as it may be telling us that overall tax revenue that results in lower number of teachers in the schools in the neighborhood (as the primary schools are typically managed by local municipality). The low tax revenue from the neighborhood is almost certainly also associated with the lower home prices.

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 always fails to predict 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 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 [51]: # 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_predic
        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:

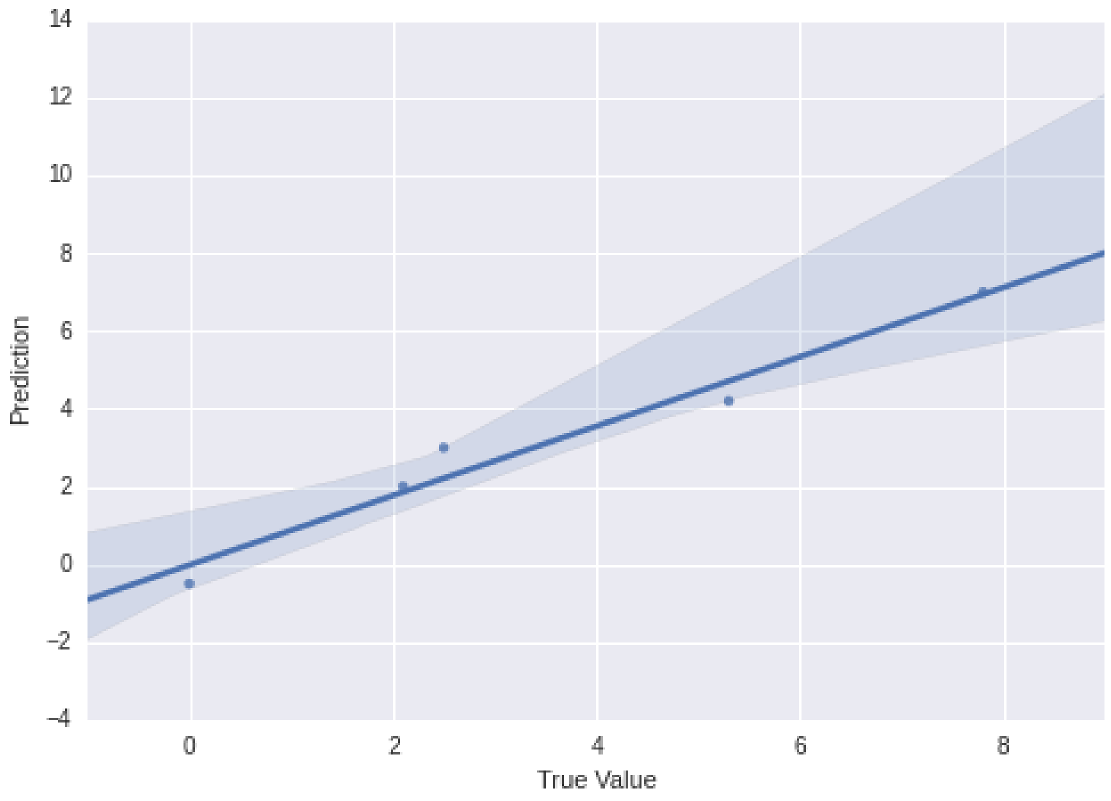
| True Value | Prediction | | :————-: | :——–: | | 3.0 | 2.5 | | -0.5 | 0.0 | | 2.0 | 2.1 | | 7.0 | 7.8 | | 4.2 | 5.3 | *Would you consider this model to have successfully captured the variation of the target variable? Why or why not?*

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

```
In [52]: # Visualise the Prediction vs. True Value
         import seaborn as sns
         sample = pd.DataFrame([3, -0.5, 2, 7, 4.2], [2.5, 0.0, 2.1, 7.8, 5.3]).res
         sample.columns = ['True Value', 'Prediction']
         sns.regplot('True Value', 'Prediction', sample)

         # 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(s
```

```
Model has a coefficient of determination, Rˆ2, of 0.923.
```



Answer: The model captures the variation in the target variable rather well as it misses less then 8% of it.

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 [53]: # 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

         # 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: Splitting a dataset into training and testing subsets preserves the possibility to control for overfitting in the model. For instance, it is always possible to perfectly fit all of the data in the training sample by adding large number of features or just simply learning all of the training data points (having a lookup table instead of a more succint model). If we keep a part of the data only for testing purposes we increase our chances to detect the problems due to the overfitting, as the overly optimized model from the training step would perform significantly worse on the data it has not seen before.

1.6 Analyzing Model Performance

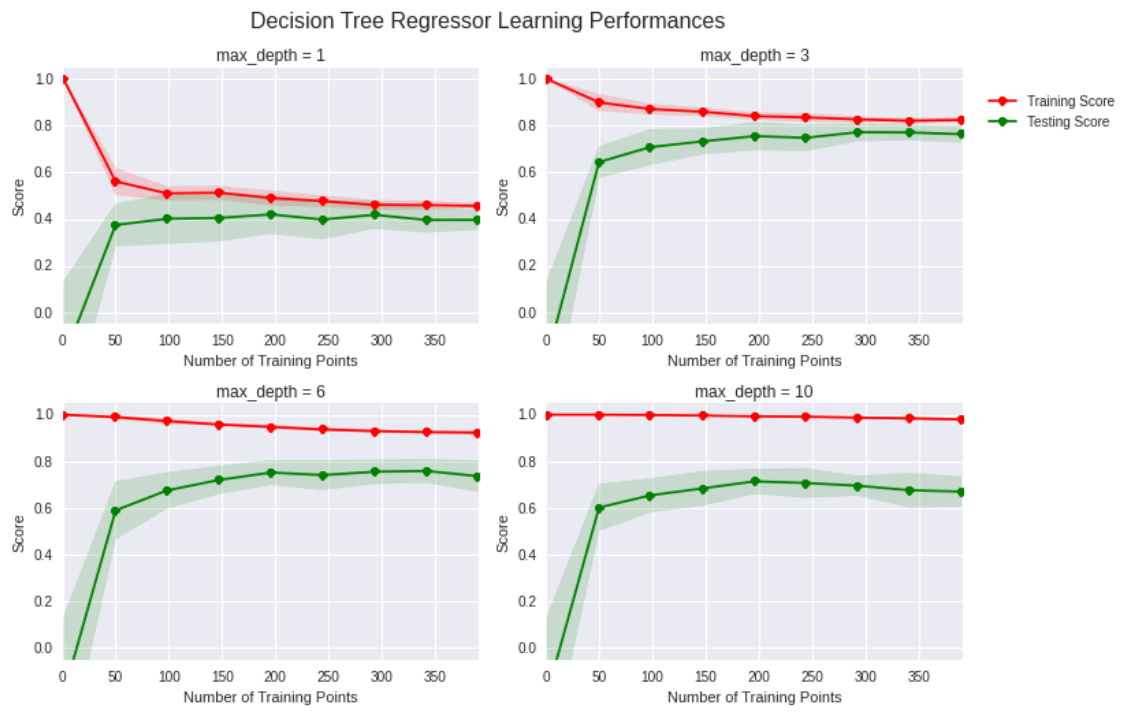
In this third section of the project, you'll take a look at several models' learning and testing performances on various subsets of training data. Additionally, you'll investigate one particular algorithm with an increasing `'max_depth'` parameter on the full training set to observe how model complexity 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 reigon 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 [54]: # Produce learning curves for varying training set sizes and maximum dept
         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 best option among the choices tested above is clearly the one for max_depth=3. It gives the best overall performance on the testing test and it also gives very good performance for the test dataset.

The training score starts very high and then slowly decreases and reaches plateau after ~200-300 training points. The testing score starts very low but then quickly increases with more training points untill it reaches plateau of values similar to those of the training curve.

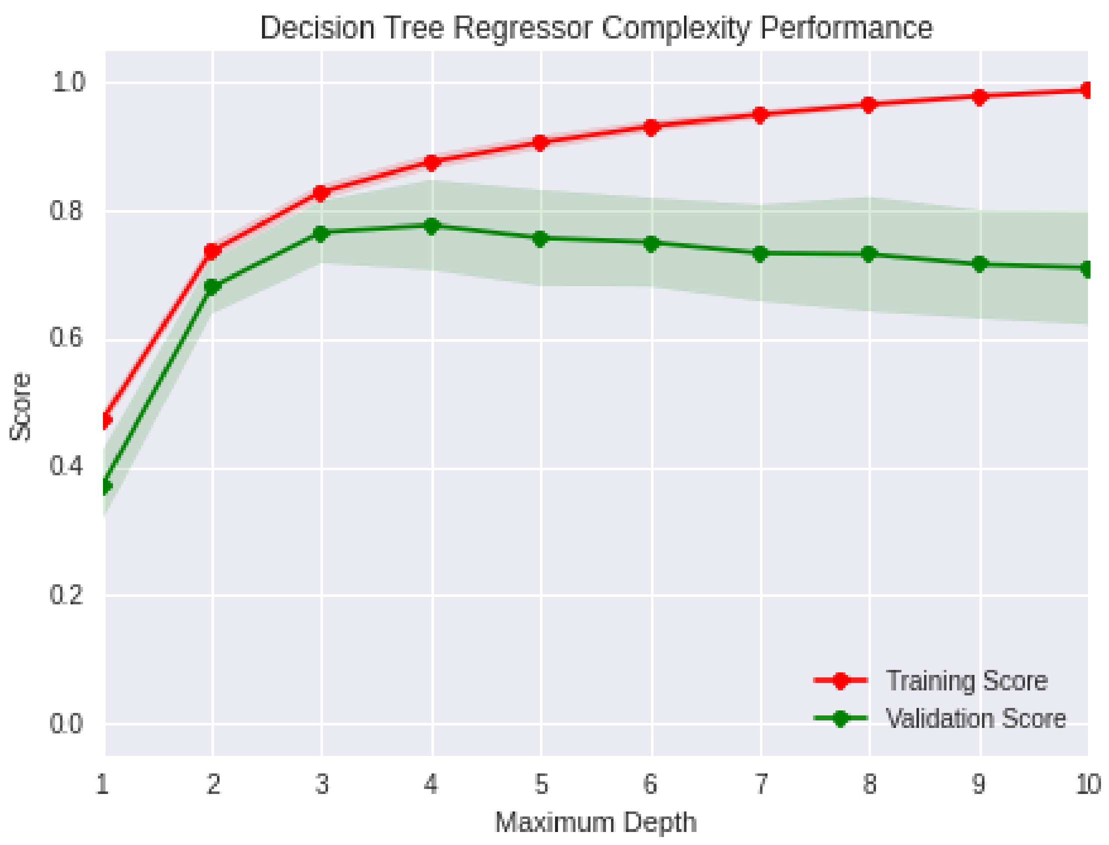
Both learning and testing curves seem to reach a plateau after ~300 examples. It is thus unlikely that adding more training points would improve the performance of the model significantly.

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 [55]: 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?

Answer:

The max_depth=1 clearly underfits the data – the initial part (max_depth=1) of the complexity curve indicates a poor performance for both the training and the test datasets. The model suffers from the high bias.

The max_depth=10 overfits the data. Judging from the complexity curve, we get an excellent performance on the training set but significantly poorer performance on the test dataset. Adding significantly more data points may be helpful in this case as it may help to alleviate the high variance problem.

Overall visual cues that I see in the graph: close and not very good scores for both train and test datasets for max_depth=1,2 – indication of high bias. Diverging curves of training and validation scores for max_depth>4 – indication of overfitting.

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: The max_depth that results in the best score on the test dataset should generalize best to unseen data. In this particular case it seems to be the max_depth=3. The elbow in the validation score curve between max_depth=3 and max_depth=4 could be a cue for selecting the optimum complexity of the model.

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: Grid search is a method for searching for optimum model parameters. In this technique we go over all combinations of different model porameters within pre-defined range or enumerated list and evaluate performance of the model at each parameter hyperspace point.

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: K-fold cross-validation is a model selection/evaluation technique.

It uses of all of the data for training and validation of the model, while controlling for overfitting.

In the k-fold cross-validation we go over the data k times. We first split the dataset into k folds, then we go over data keeping each time one of the folds as the test subset while training the model on the rest of the data.

At each step we determine the error associated with the model.

At the end of the process we attribute an average of the errors from the individual steps as the overall performance metrics of the model.

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*.

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 [56]: # TODO: Import 'make_scorer', 'DecisionTreeRegressor', and 'GridSearchCV'
         from sklearn.tree import DecisionTreeRegressor
         from sklearn.grid_search import GridSearchCV
         from sklearn.metrics import make_scorer
         def fit_model(X, y):
             """ Performs grid search over the 'max_depth' parameter for a
                 decision tree regressor trained on the input data [X, y]. """

             # Create cross-validation sets from the training data
             cv_sets = ShuffleSplit(X.shape[0], n_iter = 10, test_size = 0.20, ranc

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

             # TODO: Create a dictionary for the parameter 'max_depth' with a range

             params = {'max_depth':range(1,11)}

             # TODO: Transform 'performance_metric' into a scoring function using
             scoring_fnc = make_scorer(performance_metric)

             # TODO: Create the grid search object
             grid = GridSearchCV(estimator=regressor, param_grid=params, 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 [57]: # 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_
```

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

Answer: The grid search determined max_depth=4 as optimum. In question 6 I have guessed the optimum to be max_depth=3. In question 6 I have judged by eye from the graph the training and validation score was as good as for max_depth=4, the test dataset score was closer to the score obtained on the train dataset and the spread of the test scores was lower for max_depth=3 then for max_depth=4.

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 | | Household net worth (income) | Top 34th percent | Bottom 45th percent | Top 7th percent | | 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.

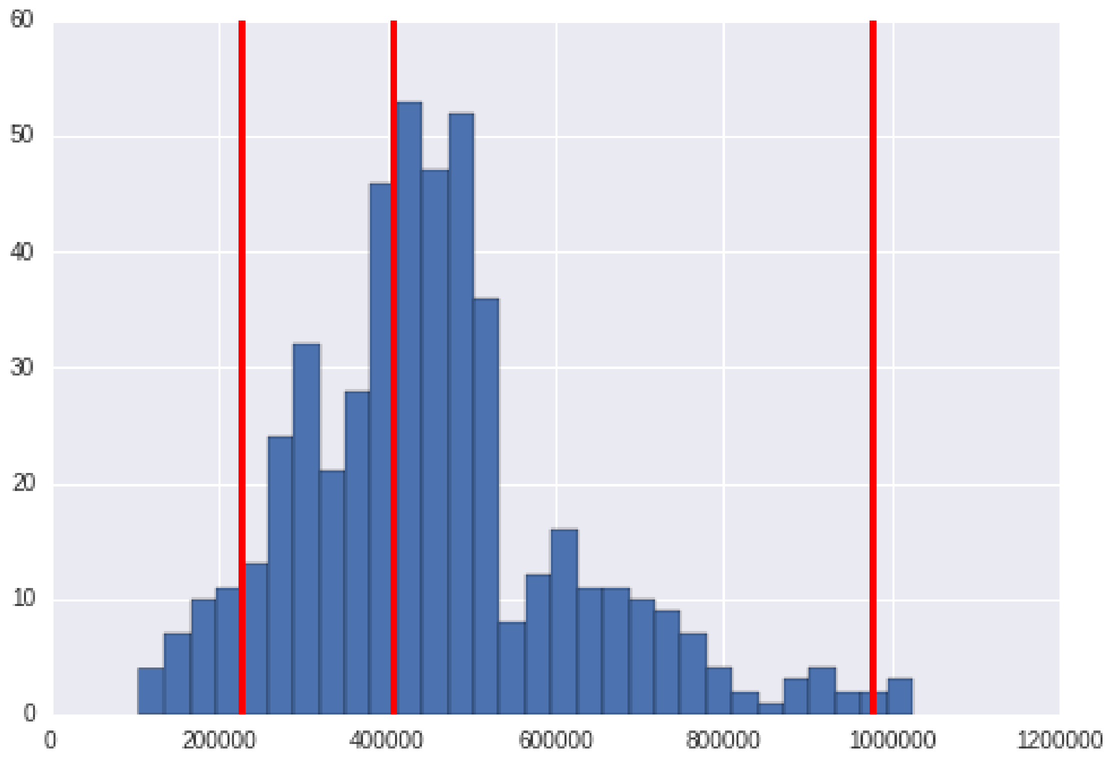
```
In [58]: # Produce a matrix for client data
         client_data = [[5, 34, 15], # Client 1
                        [4, 55, 22], # Client 2
                        [8, 7, 12]]  # Client 3

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

```
Predicted selling price for Client 1's home: $407,480.77
Predicted selling price for Client 2's home: $225,642.86
Predicted selling price for Client 3's home: $976,150.00
```

```
In [59]: import matplotlib.pyplot as plt
         plt.hist(prices, bins=30)
```

```
for price in reg.predict(client_data):
    plt.axvline(price, c='r', lw=3)
```



Answer: The recommendation of the selling prices for the three clients based on the model would be the following (with a little rounding to help with marketing):

Client 1's home: $410,000.00

Client 2's home: $225,000.00

Client 3's home: $980,000.00

The overall prediction of selling prices seems reasonable. All prices are within the range of prices in our training dataset. The positioning of the clients' selling points can be better appreciated with respect to the overall learning dataset distribution in the figure above.

The model reasonably predicts the lowest price for the smallest home in the relatively poorer neighborhood and the highest price for the largest home in the most well off neighborhood. And the mid-range price for the home in the middle class neighborhood.

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

```
Trial 1: $324,240.00
Trial 2: $324,450.00
Trial 3: $346,500.00
Trial 4: $420,622.22
Trial 5: $302,400.00
Trial 6: $411,931.58
Trial 7: $344,750.00
Trial 8: $407,232.00
Trial 9: $352,315.38
Trial 10: $316,890.00

Range in prices: $118,222.22
```

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: I think the data collected in 1978 are still relevant today, provided the prices are adjusted for inflation. Though it may not be indefinitely so. As the society changes overtime, various economic and social factors may be changing the real estate landscape. For instance the rising income inequality may lead to significant rise of prices in a few selected areas and lower prices elswhere. On the other hand more significant redistribution and social housing programs may be able to act in the opposite direction and narrow and unskew the distribution.

The features in the data are sufficient to get a ballpark figure for the price of the home. They may be sufficient to get an estimate to within several tens of thousands of dollars. More extensive set of features would probably be necessary to get higher precision (e.g. age and state of the property, more detailed hints of the location, transport connection). The three features that the model use are remarkably informative though as they tell us a lot about the home size, the average income and the social development of the neighborhood.

I think the model is quite robust as the features it is built upon are predicitive and tell different, complementary piece of information. The model e.g. correctly ranks different kind of properties in different neighborhoods, and it can give a rough estimate of the selling price (to within a few tens of thousands of dollars).

The concrete data collected in the city like Boston would not be applicable in a rural city as the segmentation of the society and the income distribution may be quite different. But if we were to built a model for any city it is not unreasonable to include the very same features as the predictors of the home prices as they are likely to preserve their predictive power.

```
In [ ]:
```

```
In [ ]:
```