boston_housing

May 30, 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 written. You will need to implement additional functionality to successfully answer all of the questions for this project. Unless it is requested, do not modify any of the code that has already been included. In this template code, there are four sections which you must complete to successfully produce a prediction with your model. Each section where you will write code is preceded by a **STEP X** header with comments describing what must be done. Please read the instructions carefully!

In addition to implementing code, there will be questions that you must answer that relate to the project and your implementation. Each section where you will answer a question is preceded by a **QUESTION X** header. Be sure that you have carefully read each question and provide thorough answers in the text boxes that begin with "**Answer:**". Your project submission will be evaluated based on your answers to each of the questions.

A description of the dataset can be found here, which is provided by the **UCI Machine Learning Repository**.

2 Getting Started

To familiarize yourself with an iPython Notebook, **try double clicking on this cell**. You will notice that the text changes so that all the formatting is removed. This allows you to make edits to the block of text you see here. This block of text (and mostly anything that's not code) is written using Markdown, which is a way to format text using headers, links, italics, and many other options! Whether you're editing a Markdown text block or a code block (like the one below), you can use the keyboard shortcut **Shift + Enter** or **Shift + Return** to execute the code or text block. In this case, it will show the formatted text.

Let's start by setting up some code we will need to get the rest of the project up and running. Use the keyboard shortcut mentioned above on the following code block to execute it. Alternatively, depending on your iPython Notebook program, you can press the **Play** button in the hotbar. You'll know the code block executes successfully if the message "Boston Housing dataset loaded successfully!" is printed.

```
In [110]: # Importing a few necessary libraries
          import numpy as np
          import matplotlib.pyplot as pl
          from sklearn import datasets
          from sklearn.tree import DecisionTreeRegressor

          # Make matplotlib show our plots inline (nicely formatted in the notebook)
          %matplotlib inline

          # Create our client's feature set for which we will be predicting a selling price
```

```
CLIENT_FEATURES = [[11.95, 0.00, 18.100, 0, 0.6590, 5.6090, 90.00, 1.385, 24, 680.0, 20.20, 3

# Load the Boston Housing dataset into the city_data variable
city_data = datasets.load_boston()

# Initialize the housing prices and housing features
housing_prices = city_data.target
housing_features = city_data.data

print "Boston Housing dataset loaded successfully!"
```

Boston Housing dataset loaded successfully!

3   Statistical Analysis and Data Exploration

In this first section of the project, you will quickly investigate a few basic statistics about the dataset you are working with. In addition, you'll look at the client's feature set in CLIENT_FEATURES and see how this particular sample relates to the features of the dataset. Familiarizing yourself with the data through an explorative process is a fundamental practice to help you better understand your results.

3.1   Step 1

In the code block below, use the imported numpy library to calculate the requested statistics. You will need to replace each None you find with the appropriate numpy coding for the proper statistic to be printed. Be sure to execute the code block each time to test if your implementation is working successfully. The print statements will show the statistics you calculate!

```
In [111]: # Number of houses in the dataset
          total_houses = city_data.data.shape[0]

          # Number of features in the dataset
          total_features = city_data.data.shape[1]

          # Minimum housing value in the dataset
          minimum_price = np.min(housing_prices)

          # Maximum housing value in the dataset
          maximum_price = np.max(housing_prices)

          # Mean house value of the dataset
          mean_price = np.mean(housing_prices)

          # Median house value of the dataset
          median_price = np.median(housing_prices)

          # Standard deviation of housing values of the dataset
          std_dev = np.std(housing_prices)

          # Show the calculated statistics
          print "Boston Housing dataset statistics (in $1000's):\n"
          print "Total number of houses:", total_houses
          print "Total number of features:", total_features
          print "Minimum house price:", minimum_price
          print "Maximum house price:", maximum_price
```

```
        print "Mean house price: {0:.3f}".format(mean_price)
        print "Median house price:", median_price
        print "Standard deviation of house price: {0:.3f}".format(std_dev)
```

```
Boston Housing dataset statistics (in $1000's):

Total number of houses: 506
Total number of features: 13
Minimum house price: 5.0
Maximum house price: 50.0
Mean house price: 22.533
Median house price: 21.2
Standard deviation of house price: 9.188
```

3.2 Question 1

As a reminder, you can view a description of the Boston Housing dataset here, where you can find the different features under **Attribute Information**. The MEDV attribute relates to the values stored in our housing_prices variable, so we do not consider that a feature of the data.

Of the features available for each data point, choose three that you feel are significant and give a brief description for each of what they measure.

Remember, you can **double click the text box below** to add your answer!

Answer:

- RM: average number of rooms per dwelling. **Reason**: This parameter measures the average rooms per dwelling, and I think the price is highly correlated with the number of rooms.

- AGE: proportion of owner-occupied units built prior to 1940. **Reason**: This parameter measures the proportion of the owner-occupied units which are built prior to 1940, and I think if the house is too old, the price may not be too high.

- DIS: weighted distances to five Boston employment centres. **Reason**: This parameter measures the weighted distances to five Boston employment centres, I think if the weighted distances are small, may more people want to buy the house, and the price may be high.

3.3 Question 2

Using your client's feature set CLIENT_FEATURES, which values correspond with the features you've chosen above?

Hint: Run the code block below to see the client's data.

```
In [112]: print CLIENT_FEATURES

[[11.95, 0.0, 18.1, 0, 0.659, 5.609, 90.0, 1.385, 24, 680.0, 20.2, 332.09, 12.13]]
```

Answer: - RM:5.609. The average number of rooms is 5.609. - AGE:90.0. The proportion of houses built prior to 1940 is 90.0%. - DIS:1.385. The weighted distances to five Boston employment centres is 1.385.

4 Evaluating Model Performance

In this second section of the project, you will begin to develop the tools necessary for a model to make a prediction. Being able to accurately evaluate each model's performance through the use of these tools helps to greatly reinforce the confidence in your predictions.

4.1 Step 2

In the code block below, you will need to implement code so that the `shuffle_split_data` function does the following: - Randomly shuffle the input data `X` and target labels (housing values) `y`. - Split the data into training and testing subsets, holding 30% of the data for testing.

If you use any functions not already acessible from the imported libraries above, remember to include your import statement below as well!

Ensure that you have executed the code block once you are done. You'll know the `shuffle_split_data` function is working if the statement "Successfully shuffled and split the data!" is printed.

```python
In [113]: # Put any import statements you need for this code block here
          from sklearn.utils import shuffle

          def shuffle_split_data(X, y, proportion):
              """ Shuffles and splits data into 70% training and 30% testing subsets,
                  then returns the training and testing subsets. """

              # Shuffle and split the data

              X, y = shuffle(X, y, random_state = 0)
              size = X.shape[0]

              X_train = X[:int (proportion * size)]
              y_train = y[:int (proportion * size)]
              X_test = X[int (proportion * size):]
              y_test = y[int (proportion * size):]

              # Return the training and testing data subsets
              return X_train, y_train, X_test, y_test


          # Test shuffle_split_data
          try:
              X_train, y_train, X_test, y_test = shuffle_split_data(housing_features, housing_prices, 0
              print "Successfully shuffled and split the data!"
          except:
              print "Something went wrong with shuffling and splitting the data."

Successfully shuffled and split the data!
```

4.2 Question 3

Why do we split the data into training and testing subsets for our model?

Answer:

- Training data used for model training, test data used for test the performance of model, so we need two datasets.
- The data should be splited, because training data and test data cannot share the same data, in order to preventing the model remember the data when training, and get too high performance in measurement.
- The split percentage should be suitable, if the training data size is too small, may cause overfitting; if the training data size is too big, the test data size is too small, it cannot measure the performance very well.

4.3 Step 3

In the code block below, you will need to implement code so that the `performance_metric` function does the following: - Perform a total error calculation between the true values of the `y` labels `y_true` and the predicted values of the `y` labels `y_predict`.

You will need to first choose an appropriate performance metric for this problem. See the sklearn metrics documentation to view a list of available metric functions. **Hint:** Look at the question below to see a list of the metrics that were covered in the supporting course for this project.

Once you have determined which metric you will use, remember to include the necessary import statement as well!

Ensure that you have executed the code block once you are done. You'll know the `performance_metric` function is working if the statement "Successfully performed a metric calculation!" is printed.

```
In [114]: # Put any import statements you need for this code block here
          from sklearn.metrics import mean_squared_error

          def performance_metric(y_true, y_predict):
              """ Calculates and returns the total error between true and predicted values
                  based on a performance metric chosen by the student. """

              error = mean_squared_error(y_true, y_predict)
              return error


          # Test performance_metric
          try:
              total_error = performance_metric(y_train, y_train)
              print "Successfully performed a metric calculation!"
          except:
              print "Something went wrong with performing a metric calculation."
```

Successfully performed a metric calculation!

4.4 Question 4

Which performance metric below did you find was most appropriate for predicting housing prices and analyzing the total error. Why? - Accuracy - Precision - Recall - F1 Score - Mean Squared Error (MSE) - Mean Absolute Error (MAE)

Answer: The best performance metric is **Mean Squared Error (MSE)**.

1. Accuracy, Precision, Recall, F1 Score are used for classification metrics, so they don't appropriate.
2. MSE compared to MAE is more appropriated, because the value will be large if predict value is far from the true value.

4.5 Step 4 (Final Step)

In the code block below, you will need to implement code so that the `fit_model` function does the following: - Create a scoring function using the same performance metric as in **Step 3**. See the sklearn `make_scorer` documentation. - Build a GridSearchCV object using `regressor`, `parameters`, and `scoring_function`. See the sklearn documentation on GridSearchCV.

When building the scoring function and GridSearchCV object, be sure that you read the parameters documentation thoroughly. It is not always the case that a default parameter for a function is the appropriate setting for the problem you are working on.

Since you are using `sklearn` functions, remember to include the necessary import statements below as well!

Ensure that you have executed the code block once you are done. You'll know the `fit_model` function is working if the statement "Successfully fit a model to the data!" is printed.

```python
In [115]: # Put any import statements you need for this code block
          from sklearn.metrics import make_scorer
          from sklearn.grid_search import GridSearchCV
          from sklearn.tree import DecisionTreeRegressor
          from sklearn.metrics import mean_squared_error

          def fit_model(X, y):
              """ Tunes a decision tree regressor model using GridSearchCV on the input data X
                  and target labels y and returns this optimal model. """

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

              # Set up the parameters we wish to tune
              parameters = {'max_depth':(1,2,3,4,5,6,7,8,9,10)}

              # Make an appropriate scoring function
              scoring_function = make_scorer(performance_metric,greater_is_better = False)

              # Make the GridSearchCV object
              reg = GridSearchCV(regressor,parameters,scoring_function)

              # Fit the learner to the data to obtain the optimal model with tuned parameters
              reg.fit(X, y)

              # Return the optimal model
              return reg.best_estimator_


          # Test fit_model on entire dataset
          try:
              reg = fit_model(housing_features, housing_prices)
              print "Successfully fit a model!"
          except:
              print "Something went wrong with fitting a model."

Successfully fit a model!
```

4.6 Question 5

What is the grid search algorithm and when is it applicable?

Answer: The grid search algorithm is a way of systematically working through multiple combinations of parameter tunes. It will compute all the possible parameters and choose the best one by the cross validation tests. When the number of parameters is small, it's ok and can find the global optmization; But when the number is large, the computing size is exponential growth. So when the number of parameters is small, it's applicable.

4.7 Question 6

What is cross-validation, and how is it performed on a model? Why would cross-validation be helpful when using grid search?

Answer: 1. Cross-validation is make the data into some(K) parts, one for testing and the other parts(K-1) for training, and do this step for K times, finally average these K error scores and choose the best one for the model. 2. All observations are used for both training and validation, and each observation is used for validation exactly once. So the performance will be very good. 3. Grid search is trying all the parameters and get one score by using cross-validation and choose the best one for the model, so it will be helpful.

5 Checkpoint!

You have now successfully completed your last code implementation section. Pat yourself on the back! All of your functions written above will be executed in the remaining sections below, and questions will be asked about various results for you to analyze. To prepare the **Analysis** and **Prediction** sections, you will need to intialize the two functions below. Remember, there's no need to implement any more code, so sit back and execute the code blocks! Some code comments are provided if you find yourself interested in the functionality.

```
In [123]: def learning_curves(X_train, y_train, X_test, y_test):
              """ Calculates the performance of several models with varying sizes of training data.
                  The learning and testing error rates for each model are then plotted. """

              print "Creating learning curve graphs for max_depths of 1, 3, 6, and 10. . ."

              # Create the figure window
              fig = pl.figure(figsize=(10,8))

              # We will vary the training set size so that we have 50 different sizes
              sizes = np.rint(np.linspace(1, len(X_train), 50)).astype(int)
              train_err = np.zeros(len(sizes))
              test_err = np.zeros(len(sizes))

              # Create four different models based on max_depth
              for k, depth in enumerate([1,3,6,10]):

                  for i, s in enumerate(sizes):

                      # Setup a decision tree regressor so that it learns a tree with max_depth = depth
                      regressor = DecisionTreeRegressor(max_depth = depth)

                      # Fit the learner to the training data
                      regressor.fit(X_train[:s], y_train[:s])

                      # Find the performance on the training set
                      train_err[i] = performance_metric(y_train[:s], regressor.predict(X_train[:s]))

                      # Find the performance on the testing set
                      test_err[i] = performance_metric(y_test, regressor.predict(X_test))

                  # Subplot the learning curve graph
                  ax = fig.add_subplot(2, 2, k+1)
                  ax.plot(sizes, test_err, lw = 2, label = 'Testing Error')
                  ax.plot(sizes, train_err, lw = 2, label = 'Training Error')
                  ax.legend()
                  ax.set_title('max_depth = %s'%(depth))
                  ax.set_xlabel('Number of Data Points in Training Set')
```

```
              ax.set_ylabel('Total Error')
              ax.set_xlim([0, len(X_train)])

          # Visual aesthetics
          fig.suptitle('Decision Tree Regressor Learning Performances', fontsize=18, y=1.03)
          fig.tight_layout()
          fig.show()

In [124]: def model_complexity(X_train, y_train, X_test, y_test):
              """ Calculates the performance of the model as model complexity increases.
                  The learning and testing errors rates are then plotted. """

              print "Creating a model complexity graph. . . "

              # We will vary the max_depth of a decision tree model from 1 to 14
              max_depth = np.arange(1, 14)
              train_err = np.zeros(len(max_depth))
              test_err = np.zeros(len(max_depth))

              for i, d in enumerate(max_depth):
                  # Setup a Decision Tree Regressor so that it learns a tree with depth d
                  regressor = DecisionTreeRegressor(max_depth = d)

                  # Fit the learner to the training data
                  regressor.fit(X_train, y_train)

                  # Find the performance on the training set
                  train_err[i] = performance_metric(y_train, regressor.predict(X_train))

                  # Find the performance on the testing set
                  test_err[i] = performance_metric(y_test, regressor.predict(X_test))

              # Plot the model complexity graph
              pl.figure(figsize=(7, 5))
              pl.title('Decision Tree Regressor Complexity Performance')
              pl.plot(max_depth, test_err, lw=2, label = 'Testing Error')
              pl.plot(max_depth, train_err, lw=2, label = 'Training Error')
              pl.legend()
              pl.xlabel('Maximum Depth')
              pl.ylabel('Total Error')
              pl.show()
```

# 6 Analyzing Model Performance

In this third section of the project, you'll take a look at several models' learning and testing error rates 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 learning and testing errors. 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.

```
In [125]: learning_curves(X_train, y_train, X_test, y_test)

Creating learning curve graphs for max_depths of 1, 3, 6, and 10. . .
```

Decision Tree Regressor Learning Performances
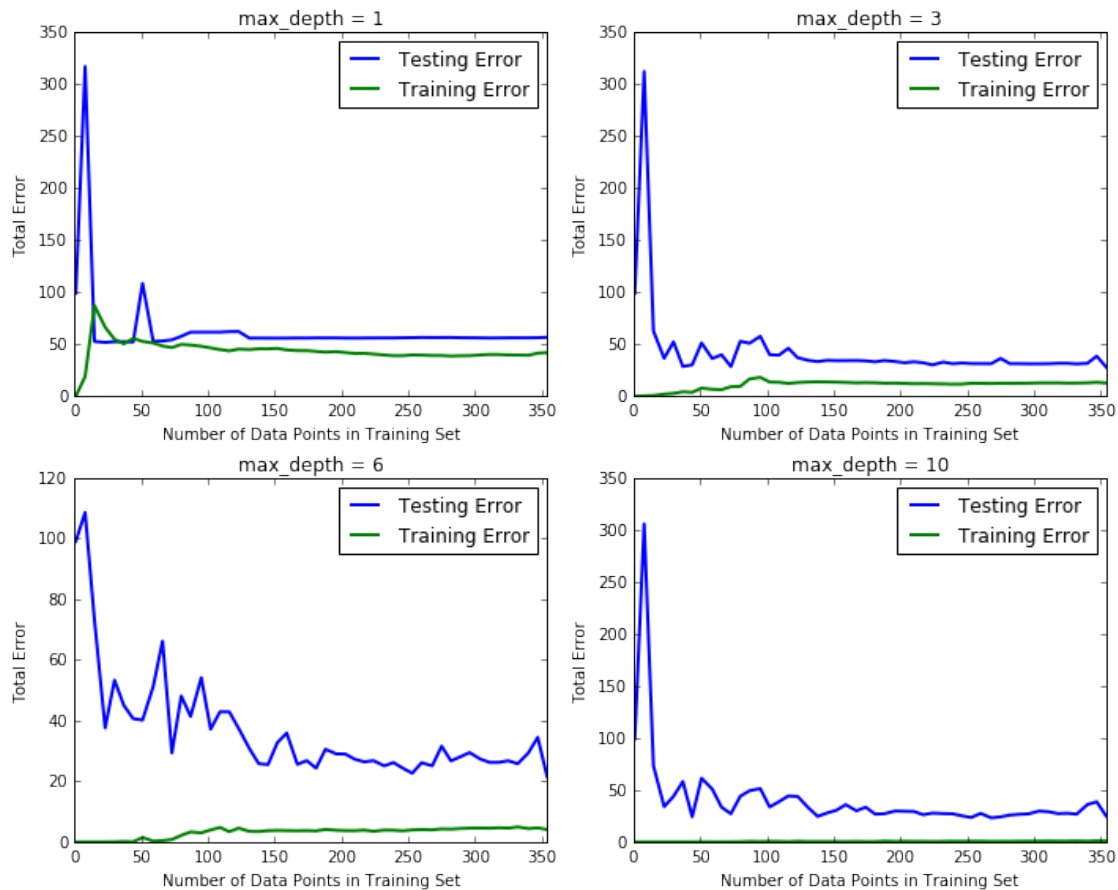
6.1 Question 7

Choose one of the learning curve graphs that are created above. What is the max depth for the chosen model? As the size of the training set increases, what happens to the training error? What happens to the testing error?

Answer: I choose the last learning curve graph. The max depth for the chosen model is 10. And because of the model is complexity, when the training set increases, the training error remaining about 0, and the testing error slowly reduces but still not very small, this because of the overfitting.

6.2 Question 8

Look at the learning curve graphs for the model with a max depth of 1 and a max depth of 10. When the model is using the full training set, does it suffer from high bias or high variance when the max depth is 1? What about when the max depth is 10?

Answer: 1. Yes, it suffer from high bias when the max depth is 1, because the curve show that, the training error and the testing error are both high and with more traning data it cannot reduce the error. 2. Yes, it suffer from high variance when the max depth is 10, because the curve of testing error is higher than the max depth of 5, the data is overfitting.

```
In [119]: model_complexity(X_train, y_train, X_test, y_test)

Creating a model complexity graph. . .
```

Decision Tree Regressor Complexity Performance

6.3 Question 9

From the model complexity graph above, describe the training and testing errors as the max depth increases. Based on your interpretation of the graph, which max depth results in a model that best generalizes the dataset? Why?

Answer: According to the figure, I think the best max depth is 5. 1. When the max depth smaller than 5, the total error is high, because of high bias. 2. When the max depth larger than 5, the total error is also growth, because of the high variance by overfitting.

7 Model Prediction

In this final section of the project, you will make a prediction on the client's feature set using an optimized model from `fit_model`. When applying grid search along with cross-validation to optimize your model, it would typically be performed and validated on a training set and subsequently evaluated on a **dedicated test set**. In this project, the optimization below is performed on the entire dataset (as opposed to the training set you made above) due to the many outliers in the data. Using the entire dataset for training provides for a less volatile prediction at the expense of not testing your model's performance.

To answer the following questions, it is recommended that you run the code blocks several times and use the median or mean value of the results.

7.1 Question 10

Using grid search on the entire dataset, what is the optimal `max_depth` parameter for your model? How does this result compare to your intial intuition?

Hint: Run the code block below to see the max depth produced by your optimized model.

```
In [122]: print "Final model has an optimal max_depth parameter of", reg.get_params()['max_depth']

Final model has an optimal max_depth parameter of 5
```

Answer: The result of max_depth is 5. And the result is the same with my analysis above.

7.2 Question 11

With your parameter-tuned model, what is the best selling price for your client's home? How does this selling price compare to the basic statistics you calculated on the dataset?

Hint: Run the code block below to have your parameter-tuned model make a prediction on the client's home.

```
In [127]: sale_price = reg.predict(CLIENT_FEATURES)
          print "Predicted value of client's home: {0:.3f}".format(sale_price[0])

Predicted value of client's home: 20.968
```

Answer: My predict value of client's home is 20.968. And compared to the basic statistics, it between the mean and median of the house price.

7.3 Question 12 (Final Question):

In a few sentences, discuss whether you would use this model or not to predict the selling price of future clients' homes in the Greater Boston area.

Answer: This model use the decision tree model with max depth of 5 which is the best depth by the model complexity performance curve. This model avoid high bias and high variance. And use the grid search alogrithm and cross validation to fitting the data. So I think this model can be used to predict the selling price.