boston_housing

March 27, 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 [1]: # 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, 332

# 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 [2]: housing_features.shape[1]

Out[2]: 13

In [3]: # Number of houses in the dataset
        total_houses = len(housing_prices)

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

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

        # Maximum housing value in the dataset
        maximum_price = np.amax(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:

1. **CRIM: per capita crime rate by town**

   Crime rate. As for myself, being safe is the most important thing for living somewhere.

2. **Rooms per dwelling.**

   The rooms number. I think it's significant because it affect the whole structure of the house. So
   it certainly affect the price of house.

3. **TAX: full-value property-tax rate per $10,000**

   The higer the tax imply needing more money to live there.

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 [18]: CHOSEN_FEATURES = ['CRIM', 'RM', 'TAX']
         features = city_data.feature_names.tolist()
         for feature in CHOSEN_FEATURES:
             idx = features.index(feature)
             print(feature, ' : ', CLIENT_FEATURES[0][idx])

CRIM  :  11.95
RM   :  5.609
TAX  :  680.0
```

Answer: > - CRIM(Crime rate): 11.95 - RM(average number of rooms per dwelling): 5.609 - TAX(full-
value property-tax rate per $10,000): 680.0

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.

```
In [24]:  # Put any import statements you need for this code block here
          # from sklearn.utils import shuffle
          from sklearn.cross_validation import train_test_split

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

              # Shuffle and split the data
              X_train, X_test, y_train, y_test = train_test_split(
              X,y, test_size=0.3, random_state=10)

              # 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)
              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: > Splitting the dataset into testing and training subsets makes the model not to overfit to just one dataset.

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.

4

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 [25]: # 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: > MSE

First, accuracy, precision, recall, and F1 score are for classification problem.

And I would choose MSE, since it's easier to compute the gradient.

Squaring the error makes big number even bigger and vice versa.

Last but not least, if there are some outliers in our dataset,

I might choose MSA or excluded the outliers,

since MSE is more sensitive to the outliers.

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 2**. 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.

```
In [19]: # Put any import statements you need for this code block
         from sklearn.metrics import make_scorer
         from sklearn.grid_search import GridSearchCV
         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(mean_squared_error, greater_is_better = False)

             # Make the GridSearchCV object
             reg = GridSearchCV(regressor, parameters, scoring=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.")

Something went wrong with fitting a model.
```

4.6 Question 5

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

Answer:

What is the grid search algorithm?

By the definition on the sklearn doc:

Exhaustive search over specified parameter values for an estimator.

In my words, grid search is putting all the parameters(features) onto a **"grid"**,

and **"search"** for which parameters get a better performance on our scorer function.

Finally, it would return a combination of parameters that delivers the best performance.

When is it applicable?

When there are too many features let us wrtie some tedious codes to try out which of them are important.

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:

What is cross-validation, and how is it performed on a model?

Split datasets into `k` blocks.(k is a positive integer)

And take one block as testing set,

the other `k-1` blocks as training set.

Repeat process aboved for k-1 times,

get the average errors of each trial.

Every block got picked as the testing set once,

and be part of the training set k-1 times.

Why would cross-validation be helpful when using grid search?

Cross-validation can maximize both the training and testing data,

it's worth to do if our dataset's size is relatively limited.

And the training set and testing set changed in every trial,

if we try to minimize the average error between so many different training sets and testing sets

that means we can minimize variance more,

which a;sp implies we can do better toward the unseen data.

Though the grid search can return the features that deliver the best performance,

but according to the reasons aboved,

cross-validation makes grid search work better on unseen data.

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 [28]: 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 [29]: 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)
```

```python
        # 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 [30]: learning_curves(X_train, y_train, X_test, y_test)

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

/Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/site-packages/matplotlib/figure.py:397:
  "matplotlib is currently using a non-GUI backend, "

Decision Tree Regressor Learning Performances
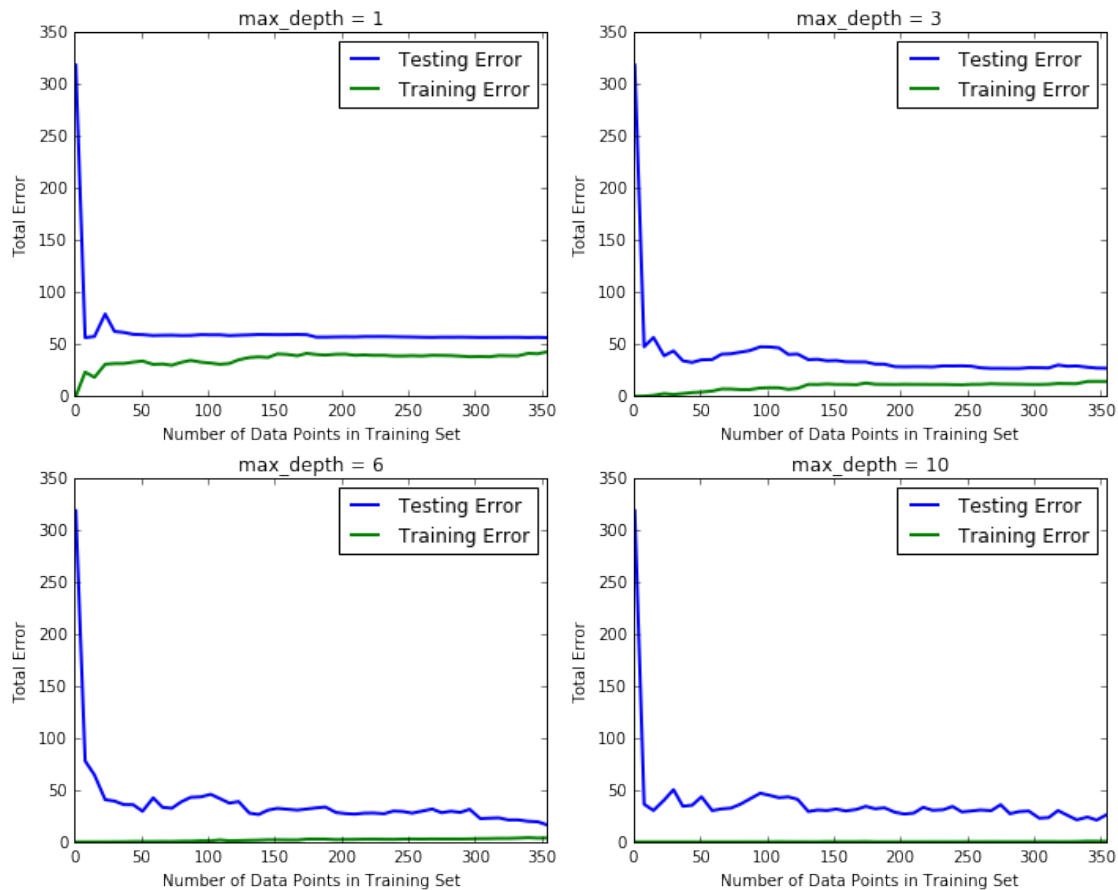
6.1 Question 7

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:

When the size of the training set increased,

the training error decrease almost to zero.

As for the testing error, it decreased fast at first,

then stick to a point and didn't decrease any more.

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:

When the max depth is 10, it may suffer from high variance,

because the model complexity might comes too high while the max depth is 10

When the max depth is 1, it may suffer from high bias and also high variance,

it may get a high training error and testing error,

because it's underfitted to both testing and training dataset by a over-simplified model.(max_depth=1)

```
In [31]: model_complexity(X_train, y_train, X_test, y_test)
```

```
Creating a model complexity graph. . .
```


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:

When the max depth start to increase,

both testing and training error decrease fast.

After `max_depth` equals to 5, the training error still goes down,

but testing error is getting bigger,

this may be a signal telling us that the model's becoming too complex to fit the unseen data.

So we may choose the point at max_depth equals to 5,

since it possibly has minimized both of underfitting and overfitting according the aboved graph.

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 <u>entire dataset</u> (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.

<u>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.</u>

7.1 Question 10

<u>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?</u>
Hint: Run the code block below to see the max depth produced by your optimized model.

```
In [32]: 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 optimal `max_depth` is 5.

My intial intuition is 8 according to the graph about complexity aboved,

I think my assumption may be overfitting.

7.2 Question 11

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

```
In [33]: 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:

Best selling price: 20.968.

It's a little bit lower than the mean.

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:

I might not use this model to predict house price.

Since it cost so much, and the dataset is relatively small,

we must have enough dataset to tune a model which have better performance.