



PROJECT

Predicting Boston Housing Prices

A part of the Machine Learning Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!  

Meets Specifications

Very nice adjustment here, check out these sections for even more insight! If the rest of your projects are on this level, this program will be a breeze. Wish you the best of luck throughout this program!

Data Exploration

All requested statistics for the Boston Housing dataset are accurately calculated. Student correctly leverages NumPy functionality to obtain these results.

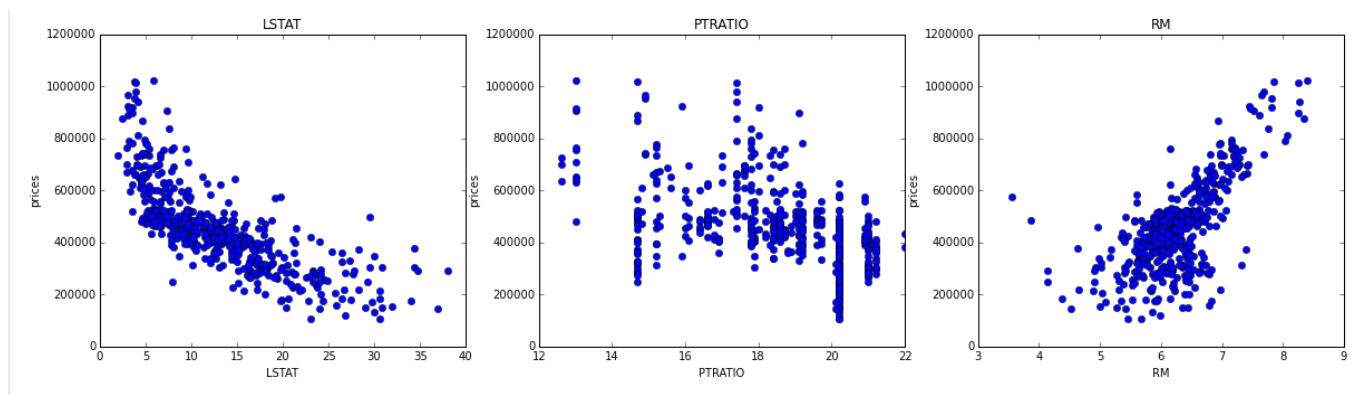
All correct! And good use of Numpy!

Student correctly justifies how each feature correlates with an increase or decrease in the target variable.

"As is usually the case, lower class workers will not be able to afford expensive houses and hence a neighborhood with higher percent of lower class workers will have a relatively lower MEDV. Therefore, MEDV will increase with the decrease in LSTAT."

Much better! Nice observations for the features in this dataset. As we can confirm these ideas by plotting each feature vs MEDV housing prices.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(20, 5))
for i, col in enumerate(features.columns):
    plt.subplot(1, 3, i)
    plt.plot(data[col], prices, 'o')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('prices')
```

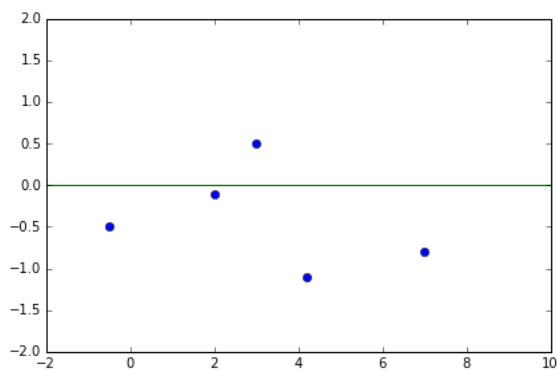


Developing a Model

Student correctly identifies whether the hypothetical model successfully captures the variation of the target variable based on the model's R^2 score. The performance metric is correctly implemented in code.

Another interesting thing to check out for regression problems would be a residual plot. A residual plot is a graph that shows the residuals on the vertical axis and the independent variable on the horizontal axis. If the points in a residual plot are randomly dispersed around the horizontal axis, a linear regression model is appropriate for the data; otherwise, a non-linear model is more appropriate.

```
import matplotlib.pyplot as plt
trueValues = [3, -0.5, 2, 7, 4.2]
predictions = [2.5, 0.0, 2.1, 7.8, 5.3]
residuals = [i - j for i, j in zip(trueValues, predictions)]
plt.plot(trueValues, residuals, 'o', [-2, 10], [0, 0], '-')
plt.ylim(-2, 2)
plt.tight_layout()
```



Student provides a valid reason for why a dataset is split into training and testing subsets for a model. Training and testing split is correctly implemented in code.

You have captured the need of a testing set for evaluation of our model. The purpose and benefit of having training and testing subsets from a dataset is the opportunity to quantify the model performance on an independent dataset and to check for overfitting. Evaluating and measuring the performance of the model over the testing subset provides estimations of the metrics that reflect how good the model predictions will be when the model is used to estimate the output using independent data (that was not used by a learning algorithm when the model was being tuned). If there is no testing subset available, there would be no way to estimate the performance of the model.

Analyzing Model Performance

Student correctly identifies the trend of both the training and testing curves from the graph as more training points are added. Discussion is made as to whether additional training points would benefit the model.

"We can see that training and testing scores are the same (around 80%) beyond 300 data points, introducing more number of data points does not seem to yield any better score."

Would recommend also describing the initial trends.

- Analysis of the training score curve: for a very small number of training points, training score is 1, or close. This happens due to model overfitting. As more training points are added, training score drops because the model cannot explain all the variance anymore. It converges to a value close to 0.8.
- Analysis of the testing score curve: for a very small number of training points, testing score is 0, or close. This happens due to model overfitting (decision tree model explains well a few points but doesn't generalize well). As more training points are added, testing score increases rapidly and converges to a value also close to 0.8. The model seems to generalize well.

"But in this case, we see that training and testing scores are converging at one point and at a decent score, hence adding more training points in this case is less likely to be beneficial"

Correct! As we gain no benefit from the testing(unseen data) if we were to receive more data. We would continue to see the testing curve to remain flat.

Student correctly identifies whether the model at a max depth of 1 and a max depth of 10 suffer from either high bias or high variance, with justification using the complexity curves graph.

If you would like to learn more about these topics, check out these links

- <http://scott.fortmann-roe.com/docs/BiasVariance.html>
- <http://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning/>
- <http://insidebigdata.com/2014/10/22/ask-data-scientist-bias-vs-variance-tradeoff/>

Student picks a best-guess optimal model with reasonable justification using the model complexity graph.

Evaluating Model Performance

Student correctly describes the grid search technique and how it can be applied to a learning algorithm.

Nice ideas! In our example we are passing 10 different values [1,2,..9,10] for 'max_depth' to grid search, meaning, we are asking to run the decision tree regression for each value of 'max_depth'. Therefore we first fit the decision tree regression model with max_depth = 1, evaluate the model based on our scoring function (r2_score in this project) based on our train/validation data(which is actually 10 sets of train/validation data produced using the ShuffleSplit method). Then we do the same for a max depth = 2 and so on. And at the end we are returned the highest scoring max depth for the validation set.

If you would like to learn about some more advanced techniques and combining gridSearch with other techniques, with the notion of [Pipelining](#), check out this blog post brought to you by Katie from lectures

- (<https://civisanalytics.com/blog/data-science/2016/01/06/workflows-python-using-pipeline-gridsearchcv-for-compact-code/>)

Student correctly describes the k-fold cross-validation technique and discusses the benefits of its application when used with grid search when optimizing a model.

To go into a bit more detail here, I would say that k-fold cross-validation with 5 splits is described as the following:

- The data is split into k folds, (cv = 5)
- We train a model on the k-1 folds, k times, and use the remaining fold as the validation set (4 for train and 1 for validation here)
- For each model we calculate the validation error (5 errors rates here)
- Then at the end, all of the error rates are averaged together. (single number)

If you would like a visual of this, this video does a great job. [Check this one out.](#)

Student correctly implements the `fit_model` function in code.

Could also check out using the `range` command

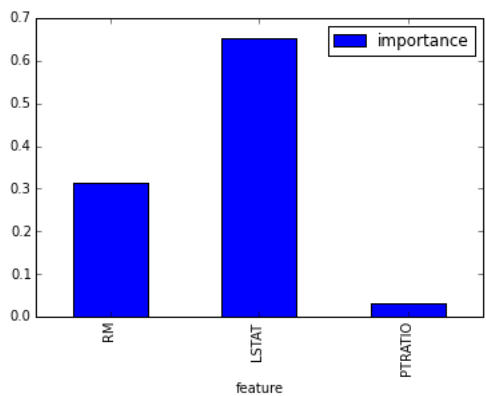
```
params = {'max_depth': range(1, 11)}
```

Student reports the optimal model and compares this model to the one they chose earlier.

Student reports the predicted selling price for the three clients listed in the provided table. Discussion is made for each of the three predictions as to whether these prices are reasonable given the data and the earlier calculated descriptive statistics.

Another cool thing with tree based method is that we can use `feature_importances` to determine the most important features for the predictions. Check this out(which one of these features contributes to the model the most?)

```
%matplotlib inline
pd.DataFrame(zip(X_train.columns, reg.feature_importances_), columns=['feature', 'importance']).set_index('feature').plot(kind='bar')
```



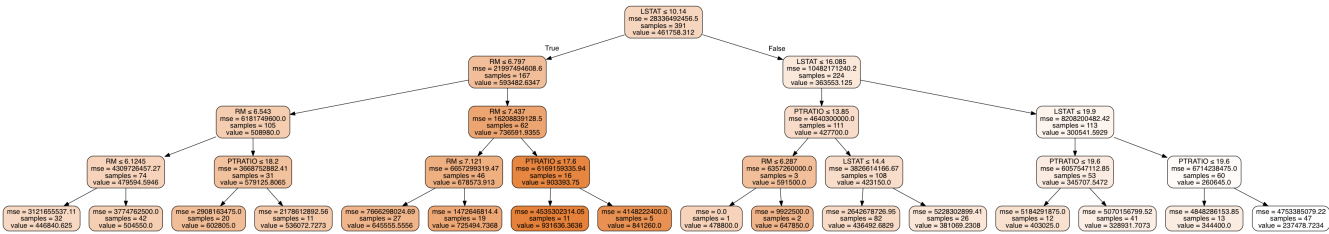
You will see this more in the next project!

Student thoroughly discusses whether the model should or should not be used in a real-world setting.

One of the biggest advantages when using a decision tree as a classifier in the interpretability of the model. Therefore we can actually visualize this exact tree with the use of `export_graphviz`

```
from IPython.display import Image
from sklearn.externals.six import StringIO
import pydot
from sklearn import tree

clf = DecisionTreeRegressor(max_depth=4)
clf = clf.fit(X_train, y_train)
dot_data = StringIO()
tree.export_graphviz(clf, out_file=dot_data,
                     feature_names=X_train.columns,
                     class_names="PRICES",
                     filled=True, rounded=True,
                     special_characters=True)
graph = pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```



[Download Project](#)

RETURN TO PATH

Rate this review



[Student FAQ](#)