UDACITY

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 adjustments here, check out the corresponding 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.

Good job utilizing the power of Numpy!! Always important to get a basic understanding of our dataset before diving in. As we now know that a "dumb" classifier that only predicts the mean would predict $454,342.94 for all houses.

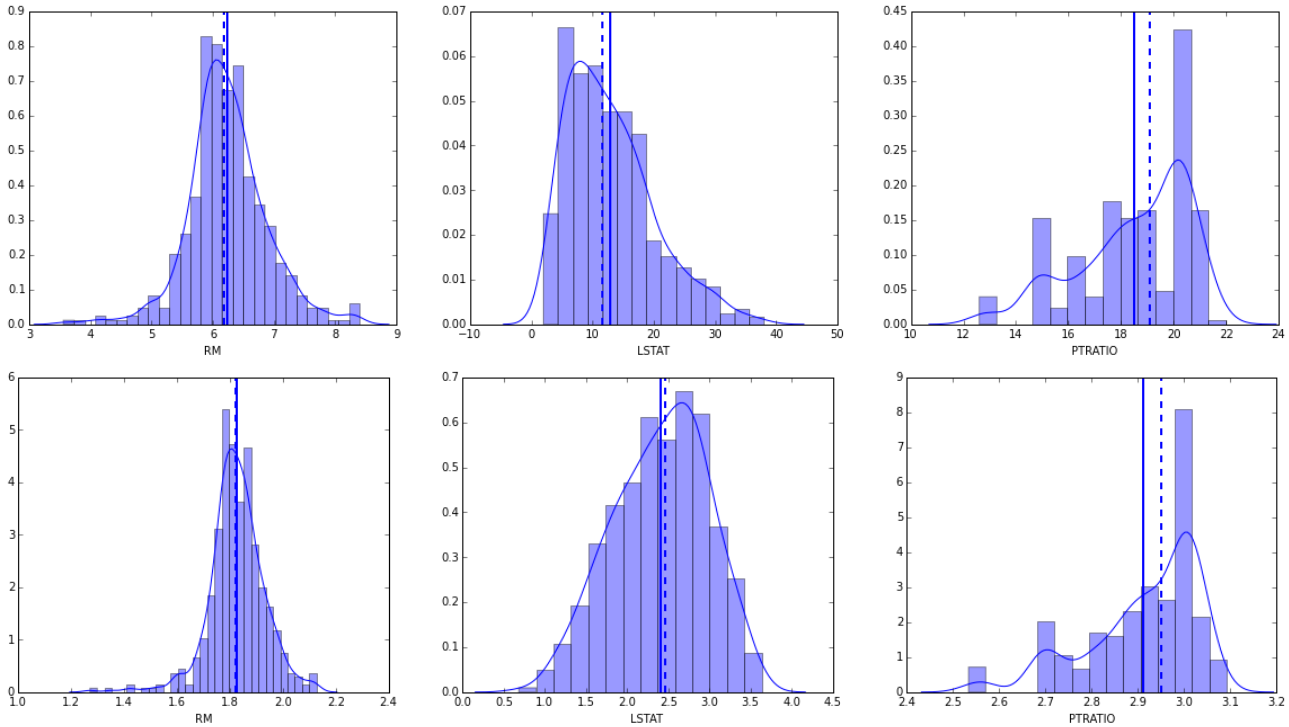
Student correctly justifies how each feature correlates with an increase or decrease in the target variable.

Could also plot histograms. Do we need any data transformations? Maybe a log transformation could be ideal?

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(20, 5))

# original data
for i, col in enumerate(features.columns):
    plt.subplot(131 + i)
    sns.distplot(data[col])
    plt.axvline(data[col].mean(), linestyle='solid', linewidth=2)
    plt.axvline(data[col].median(), linestyle='dashed', linewidth=2)

# plot the log transformed data
plt.figure(figsize=(20, 5))
for i, col in enumerate(features.columns):
    plt.subplot(131 + i)
    sns.distplot(np.log(data[col]))
    plt.axvline(np.log(data[col]).mean(), linestyle='solid', linewidth=2)
    plt.axvline(np.log(data[col]).median(), linestyle='dashed', linewidth=2)
```

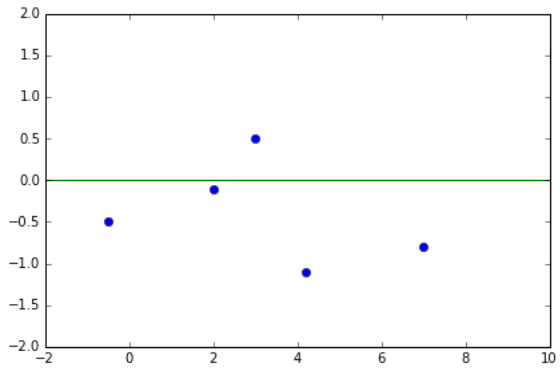


Developing a Model

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.

Great ideas. 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.

> "For all the models, testing score started to converge from 50 training points, not much improvement until almost 400 training points. So I don't think it'll benefit the model with more training points."

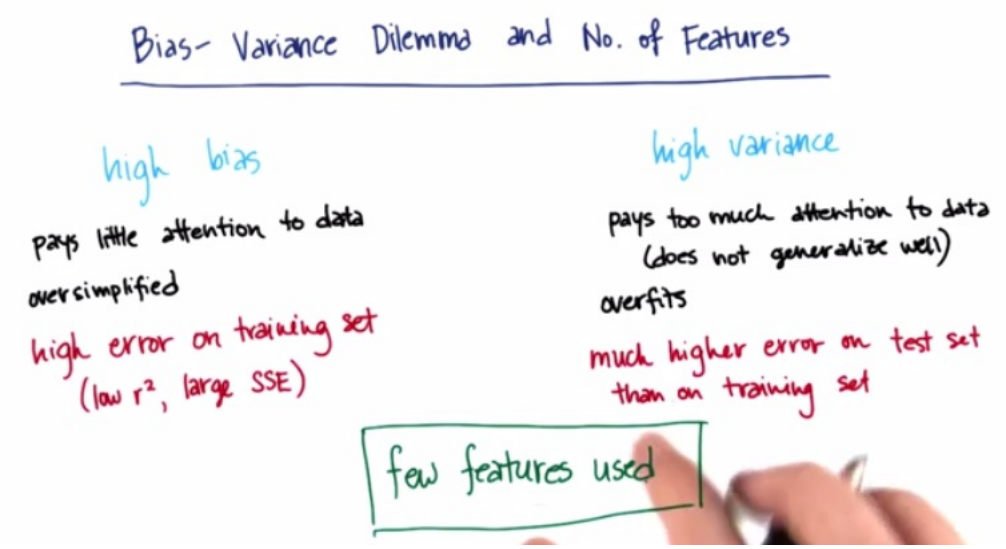
Exactly! As in the beginning it is beneficial, but at the end if we look at the testing curve here for all max depths, we can clearly see that it has converged to its optimal score, so more data is not necessary.

Also note that in practice collecting more data can often be time consuming and/or expensive, so when we can avoid having to collect more data the better. Therefore sometimes receiving very minor increases in performance is not beneficial, which is why plotting these curves can be very critical at times.

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.

Nice justification here! As the low training score is what truly depicts high bias. You clearly understand the bias/variance tradeoff.

- As a max_depth of 1 suffers from high bias, visually this is due to the low training score(also note that it has low variance since the scores are close together). As this model is not complex enough to learn the structure of the data
- And a max_depth of 10 suffers from high variance, since we see a large gap between the training and validation scores, as we are basically just memorizing our training data and will not generalize well to new unseen data



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.

> "For example, in the previous section we can also use grid search to try different max depth to find the optimized value."

To expand in how this works -- 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 out 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.

Great description of the k-fold cross-validation technique, probably the most use CV method in practice.

> "The K-Fold provides a more 'accurate' estimation of the model. If we test the model against only one validation dataset, how well the model performs really depends on how 'lucky' it is. It can be a coincidence that the model performs better than average on the specific validation dataset despite we randomly chose the data points to form the dataset."

Spot on! This is an extremely important concept in machine learning, as this allows for multiple testing datasets and is not just reliant on the particular subset of partitioned data. For example, if we use single validation set and perform grid search then it is the chance that we just select the best parameters for that specific validation set. But using k-fold we perform grid search on various validation set so we select best parameter for generalize case. Thus cross-validation better estimates the volatility by giving you the average error rate and will better represent generalization error.

If you would like a full run example, run this code based on the iris data set in your python shell or something and examine the print statements, as this is a great example

```
import numpy as np
from sklearn import cross_validation
from sklearn import datasets
from sklearn import svm

iris = datasets.load_iris()

# Split the iris data into train/test data sets with 30% reserved for testing
X_train, X_test, y_train, y_test = cross_validation.train_test_split(iris.data, iris.target, test_size=0.3, random_state
=0)

# Build an SVC model for predicting iris classifications using training data
clf = svm.SVC(kernel='linear', C=1, probability=True).fit(X_train, y_train)

# Now measure its performance with the test data with single subset
print('Testing Score', clf.score(X_test, y_test))

# We give cross_val_score a model, the entire data set and its "real" values, and the number of folds:
scores = cross_validation.cross_val_score(clf, iris.data, iris.target, cv=5)

# Print the accuracy for each fold:
print('Accuracy for individual foldd', list(scores))

# And the mean / std of all 5 folds:
print('Accuracy: %0.2f (+/- %0.2f)' % (scores.mean(), scores.std() * 2))
```

http://scikit-learn.org/stable/modules/cross_validation.html#computing-cross-validated-metrics

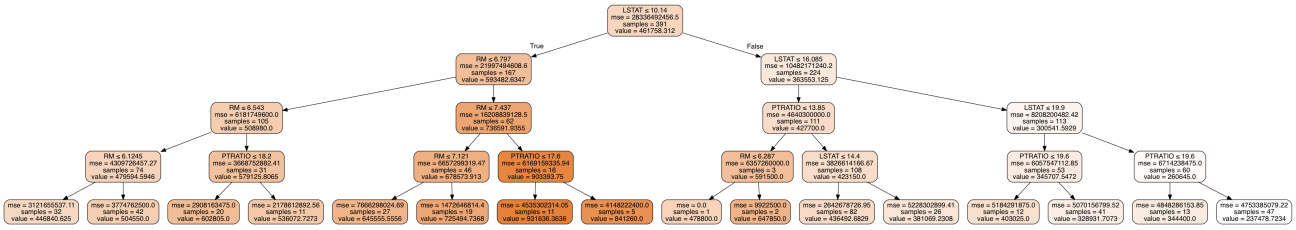
Student correctly implements the `fit_model` function in code.

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

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())
```



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.

Just remember to keep in mind the testing error here.

```
reg.score(X_test, y_test)
```

Pro Tip: We can also plot a histogram of all of the housing prices in this dataset and see where each of these predictions fall

```
import matplotlib.pyplot as plt
for i,price in enumerate(reg.predict(client_data)):
    plt.hist(prices, bins = 30)
    plt.axvline(price, lw = 3)
    plt.text(price-50000, 50, 'Client '+str(i+1), rotation=90)
```

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

> "In my previous submission, the answer was supposed to be "shouldn't", typo, I made it "should"."

I had a feeling that this might have been a typo, but just wanted to make sure. Everything looks great now 😄

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

Student FAQ