UDACITY

PROJECT

Predicting Boston Housing Prices

A part of the Machine Learning Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Requires Changes

1 SPECIFICATION REQUIRES CHANGES

This is a very impressive submission. Just need one adjustment and you will be golden, apologies for having to back on the previous review in one section here, but hey, more opportunity to learn!

Please also check out Question 10 and Question 4, as these also still have some issues. One tip here would be that some of these topics are extremely important as you embark on your journey throughout your Machine Learning career and it will be well worth your time to get a great grasp on these topics before you dive deeper in. Keep up the hard work!!

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.

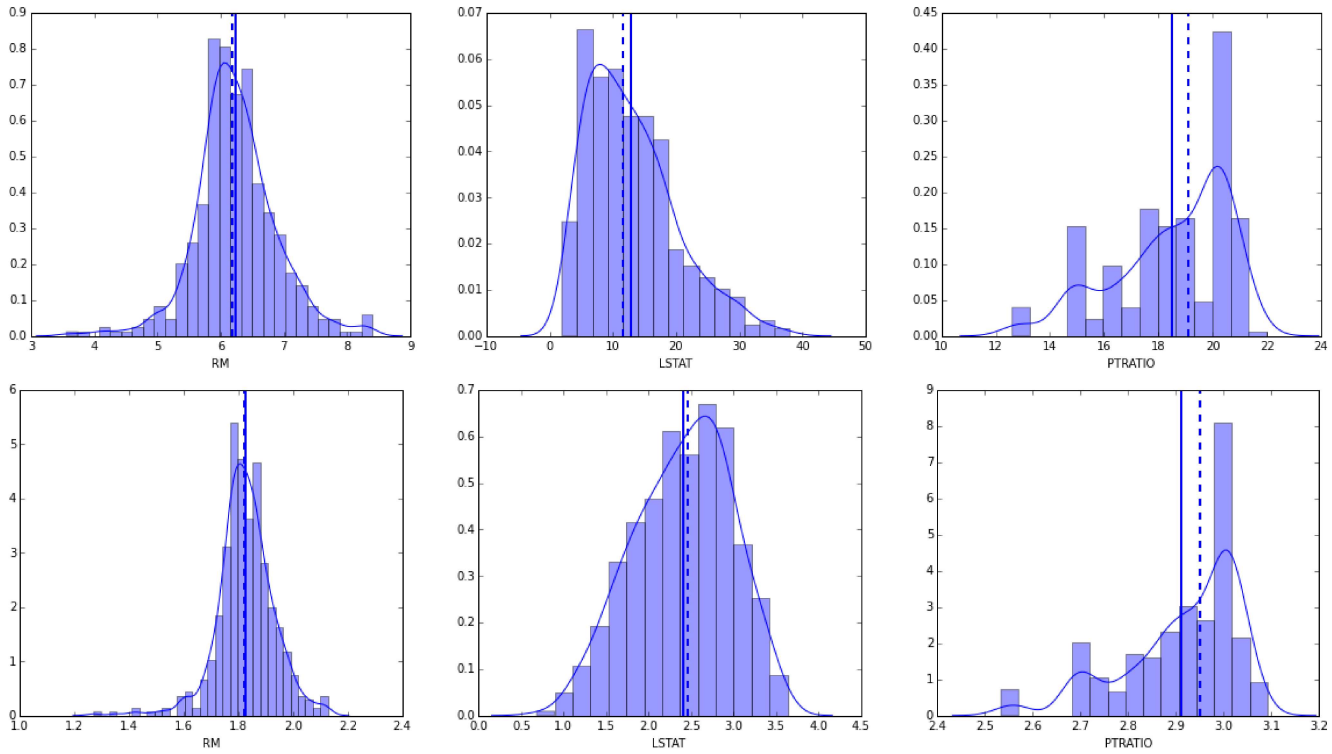
Nice observations for the features in this dataset. Visuals are nice!

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.

Nice ideas here, as this score is quite high. Could also think about if more data points would allow us to be more confident in this model? Maybe even look into hand computing this with

```
y_true_mean = np.mean(y_true)
SSres = sum(np.square(np.subtract(y_true, y_predict)))
SStot = sum(np.square(np.subtract(y_true, y_true_mean)))
score = 1.0 - SSres/SStot
```

R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression. The definition of R-squared is fairly straight-forward; it is the percentage of the response variable variation that is explained by a linear model. Or:

- R-squared = Explained variation / Total variation

R-squared is always between 0 and 100%:

- 0% indicates that the model explains none of the variability of the response data around its mean.
- 100% indicates that the model explains all the variability of the response data around its mean.

In general, the higher the R-squared, the better the model fits your data. So with a high value of 92.3% (0.923) we can clearly see that we have strong correlation between the true values and predictions.

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 reasons! We need a way to determine how well our model is doing! As we can get a good estimate of our generalization accuracy on this testing dataset. Since our main goal is to accurately predict on new unseen data.

Also note that we can try and protect against overfitting with this independent dataset.

If you would like to learn some more ideas in why we need to split our data and what to avoid, such as data leakage, check out these lectures

- https://classroom.udacity.com/courses/ud730/lessons/6370362152/concepts/63798118300923
- https://classroom.udacity.com/courses/ud730/lessons/6370362152/concepts/63798118310923

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.

> "if there was high bias present more data would help"

This comment is incorrect. High bias means that we don't have enough complexity in the model, thus no matter how many training data points we give the model, we would continue to see the testing curve to remain flat.

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.

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.

I apologize for having to mark this as *Requires Changes*, but it seems that the previous reviewer missed this. You are correct that a max_depth of 1 suffers from high bias and a max_depth of 10 suffers from high variance. And good visual justification for high variance(large gap between the training and validation scores). But please also provide some specific visual justification for a max depth of 1 and high bias. Therefore make sure you mention what the training and validation scores are. High? Low? etc... The reason for this is so you can quickly identify these instances in the future.

Links

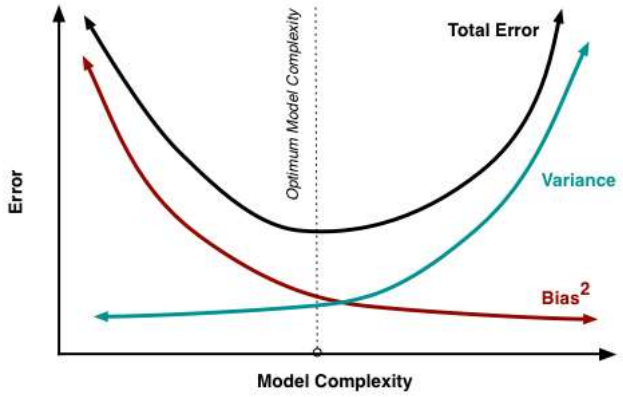
- http://scott.fortmann-roe.com/docs/BiasVariance.html
- https://insidebigdata.com/2014/10/22/ask-data-scientist-bias-vs-variance-tradeoff/
- http://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning/

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

Good intuition. Either 3 or 4 are acceptable

- As a max depth of 4 might have a higher validation score(which is what gridSearch searches for)
- But correct that a max depth of 3 has a better bias / variance tradeoff(with closer training and validation scores), also a simpler model, which is what is recommend based on Occam's razor

Check out this visual, it refers to error, but same can be applied to accuracy(just flipped)



Evaluating Model Performance

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

Excellent! As you can see that we are using a decision tree and max depth in this project. Can also note that since this trains on "*all the possible hyperparamaters*", one limitation of GridSearch is that it can be very computationally expensive when dealing with a large number of different hyperparameters and much bigger datasets. Therefore there are two other techniques that we could explore to validate our hyperparameters

- RandomizedSearchCV which can sample a given number of candidates from a parameter space with a specified distribution. Which performs surprisingly well!
- Or a train / validation / test split, and we can validate our model on the validation set. Often used with much bigger datasets

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.

> "Vanilla grid search needs a cross validation set to calculate the models score w.r.t to the paramaters. This might become a problem as we are selecting all our hyper-paramaters to optimise the score w.r.t. a consistant CV set, i.e. we could be overfitting unknowingly. K-folds allows us to consider multiple cross validation sets and hence lessening the chance of selecting a hyper-paramater that optimises for a single CV set."

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.

Nice implementation! Could also set a `random_state` in your DecisionTreeRegressor for reproducible results.

```
regressor = DecisionTreeRegressor(random_state = "any number")
```

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

Can note that GridSearch searches for the highest validation score on the different data splits in this ShuffleSplit. So 4 was a bit higher in these splits.

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.

> "The resultant predictions correspond to the relationships I intuitively determined in question 1."

I won't mark this as *Requires Changes*, but it seems that the previous reviewer missed this. For this section, you are required to give some specific justification based on the **features** for the Clients.

DO THESE PRICES SEEM REASONABLE GIVEN THE VALUES FOR THE RESPECTIVE **FEATURES**?

As a good idea would be to go client by client and give an idea for each feature of `RM`, `LSTAT`, and `PTRATIO`

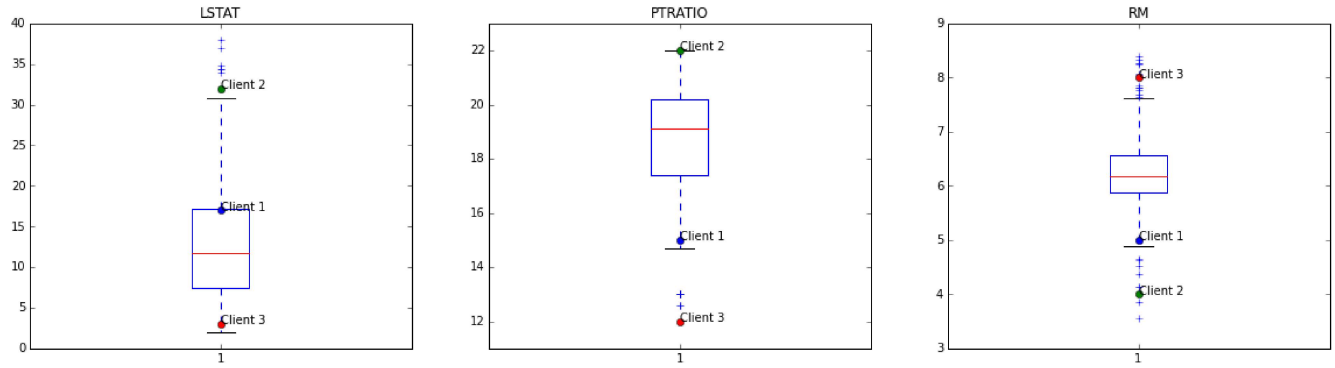
A more advanced and great idea would be to compare these to the descriptive stats of the features. We can compute the five number summary of the descriptive stats of the features with

```
features.describe()
```

For example -- Client 3 RM value is in the top 25 percentile in the data, while being near the minimum value for LSTAT, and since an increase in RM would lead to an increase in MEDV. An increase in LSTAT would lead to a decrease in MEDV. The predicted price near the maximum value in the dataset makes sense.

Maybe also plot some box plots and see how the Client's features compare to the interquartile range, median, whiskers

```
import matplotlib.pyplot as plt
plt.figure(figsize=(20, 5))
y_ax = [[3,9],[0,40],[11,23]]
for i, col in enumerate(features.columns):
    plt.subplot(1, 3, i+1)
    plt.boxplot(data[col])
    plt.title(col)
    for j in range(3):
        plt.plot(1, client_data[j][i], marker="o")
        plt.annotate('Client '+str(j+1), xy=(1,client_data[j][i]))
        plt.ylim(y_ax[i])
```



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

This dataset is quite old, probably doesn't capture enough about housing features, and the range in predictions are quite large. Therefore this model would not be considered robust!

If you would like to learn more about how to analyze the uncertainty in the output of a mathematical model, can check out these ideas (https://en.wikipedia.org/wiki/Sensitivity_analysis)

RESUBMIT

DOWNLOAD PROJECT

Learn the best practices for revising and resubmitting your project.

RETURN TO PATH

Rate this review

Student FAQ