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

This is a very impressive submission. Probably in the top 10 😄 Check out some of the other ideas presented in this review and 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.

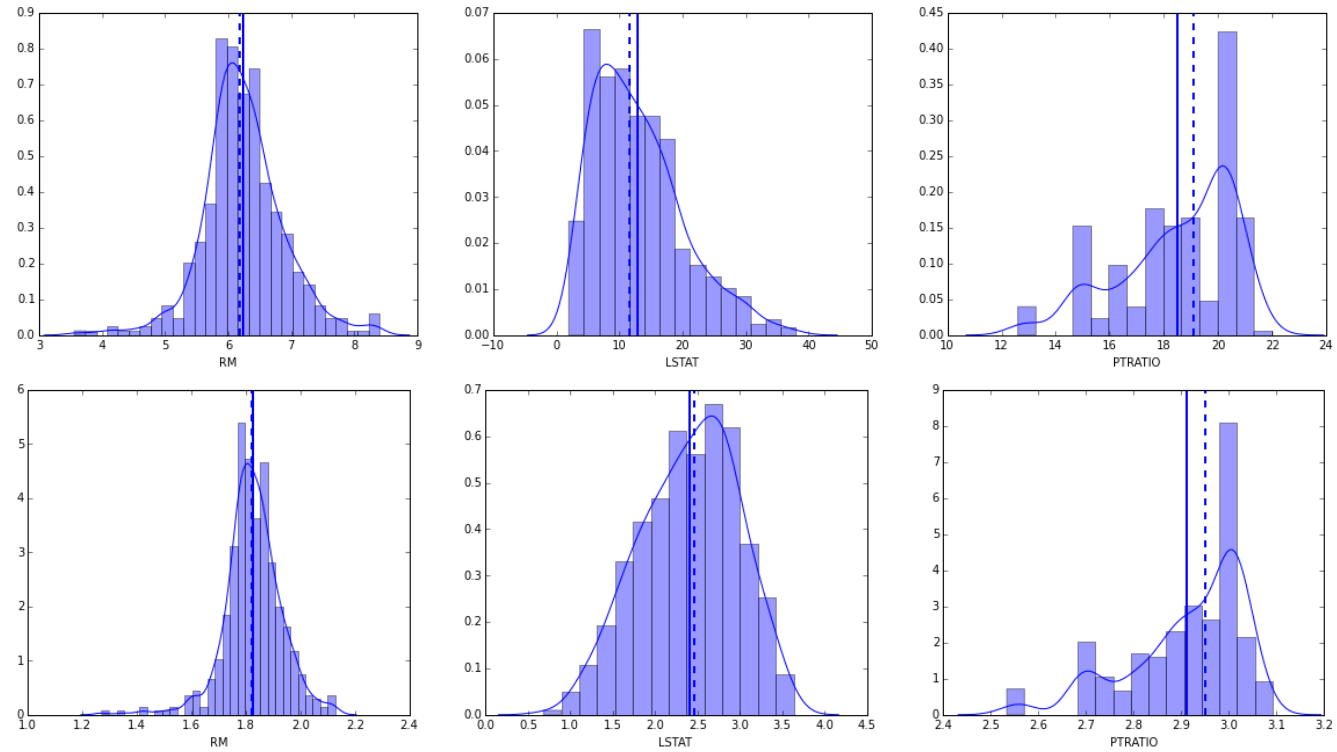
Wow! Love all the extra analysis, as this level of EDA is always the first step in any machine learning project. The outlier detection and PCA analysis are quite nice.

Here might be another way to visualize the original and log data. Should we really use log transformations for all the features?

```
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.

Code Note: We can also check out the linear relationship between the 'True Values' and 'Predictions' with a Seaborn plot

```
import seaborn as sns
sample_df = pd.DataFrame([3, -0.5, 2, 7, 4.2], [2.5, 0.0, 2.1, 7.8, 5.3]).reset_index()
sample_df.columns = ['True Value', 'Prediction']
sns.regplot('True Value', 'Prediction', sample_df)
```

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.

> "Split sizes can have the following repercussions.
> If the training data is less, the parameter estimates may have greater variance.
> With less testing data, the performance statistic will have greater variance.
> We must split, such that neither of the above variances are high. The split should also facilitate an optimal variance-bias tradeoff."

Just note that, we can actually still see underfitting or overfitting if the training data is small or if the training data is large(as it depends on the type of model and the parameter values of the model). For example you can see that a decision tree overfits the training data with a small training set(shown in the next section). But a linear model like linear regression could underfit if we don't have enough data. But both both would be bad models.

But you have the right ideas. 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.

Great analysis of the training and testing curves here. As in the initial phases, the training score decreasing and testing score increasing makes sense, since with little amounts of the data we simply memorize the training data(no generalization), then when we receive more and more data points we can't simply memorize the training data and we start to generalize better(higher testing accuracy).

> "The excess training points over 300, seem to be redundant, adding no value. This is as a result of the convergence. I would assume 300 training points would be idea"

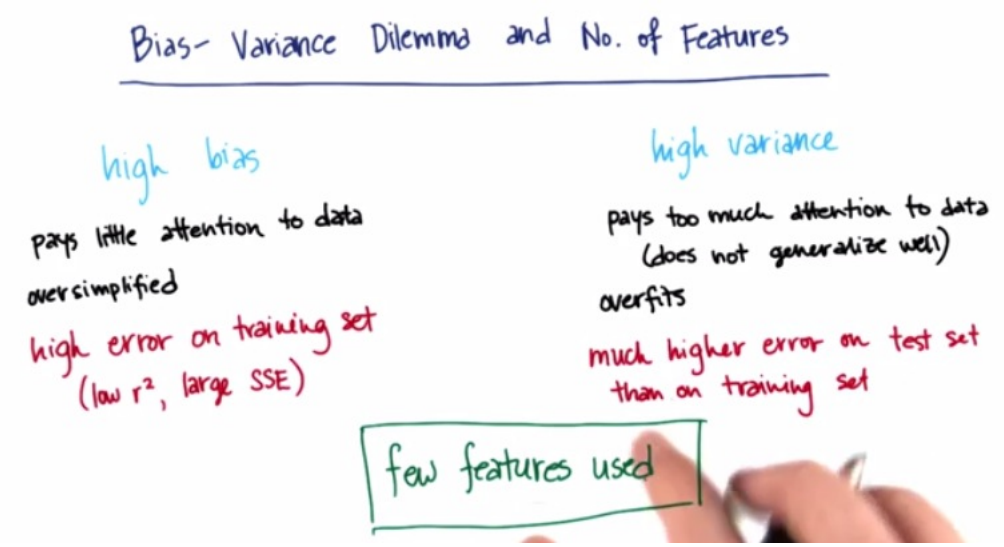
Correct! As in the beginning it is beneficial, but at the end if we look at the testing curve here, we can clearly see that it has converged to its optimal score, so more data is not necessary.

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

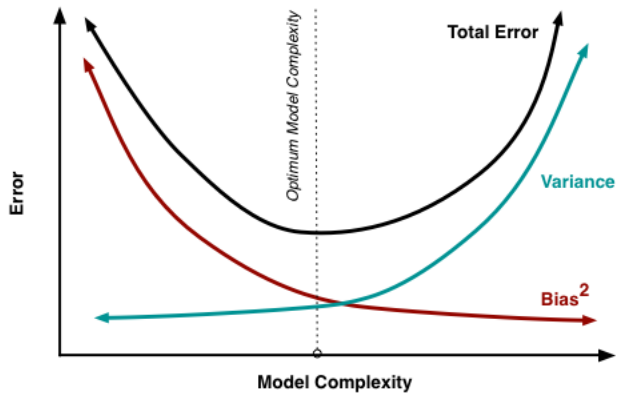


Student picks a best-guess optimal model with reasonable justification using the model complexity graph.

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! And very nice example here. 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 possible combinations of given parameters within their ranges*", 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.

> "If we do grid search on an erroraneous/broken dataset, the different permutations of our algorithim will return highly varying results compared to the average estimation."

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! Good idea to set a `random_state` in your DecisionTreeRegressor for reproducible results. This is a great habit to get into!

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

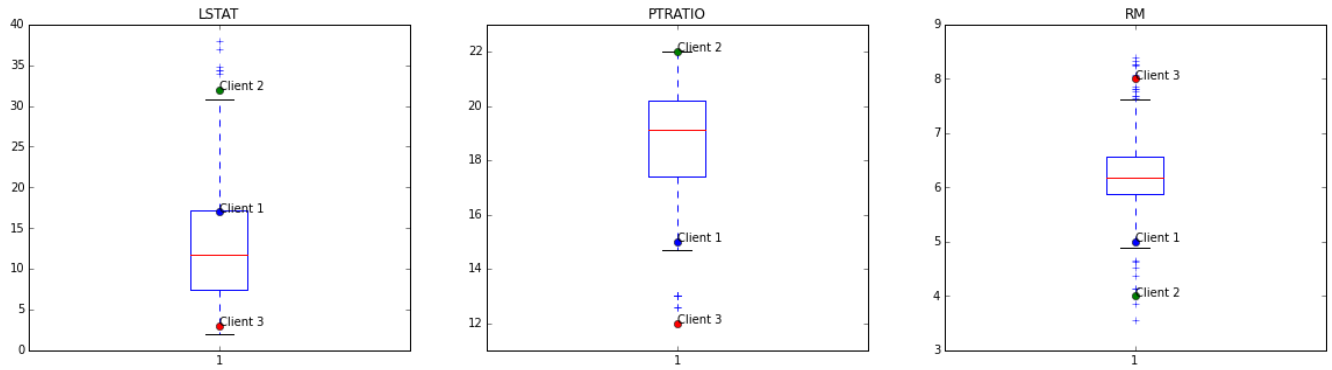
Nice plot! Can note that GridSearch searches for the highest validation score on the different data splits in this ShuffleSplit.

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.

Excellent justification for these predictions by comparing them to the features and descriptive stats. Love the ideas. This is always an important step in any machine learning project.

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])
```



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

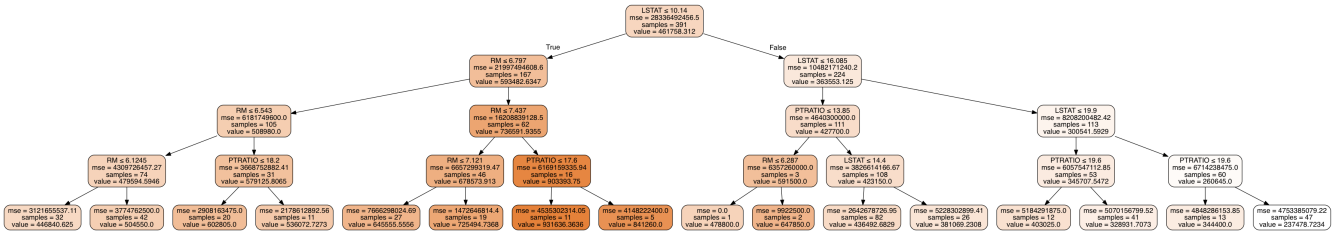
```
import matplotlib.pyplot as plt
for i,price in enumerate(reg.predict(client_data)):
    plt.hist(prices, bins = 30)
    plt.axvline(price, lw = 3)
    plt.text(price-50000, 50, 'Client '+str(i+1), rotation=90)
```

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

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

Student FAQ