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

You have made some excellent adjustments in this submission. Just one last section to perfect and you will be good to go. We look forward to seeing your final submission!

Data Exploration

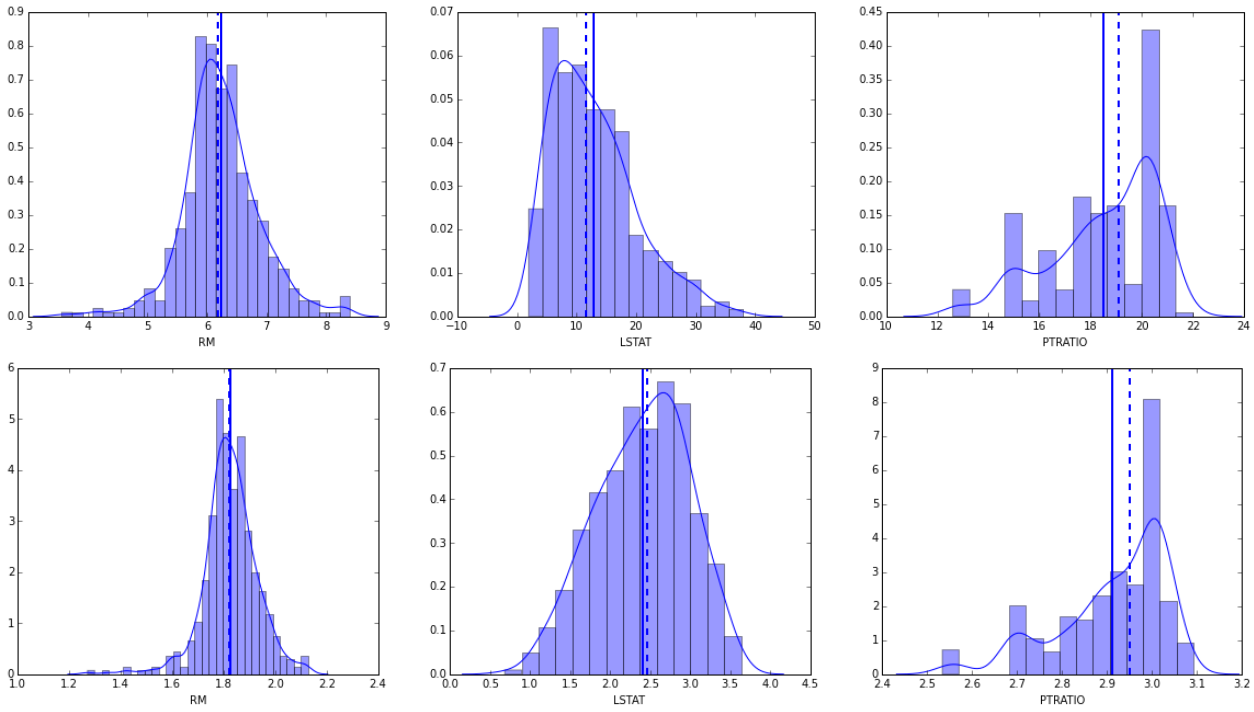
All requested statistics for the Boston Housing dataset are accurately calculated. Student correctly leverages NumPy functionality to obtain these results.

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.

Typically, in machine learning we desire to have our features to be Gaussian distributed. Therefore, could also plot histograms. Do we need any feature 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.

> "given that the R2 score is a value between 0 (unpredictable) and 1 (predictable), it seems like this has successfully captured the variation of the target variable."

Just because the R2 score is between 0 and 1 wouldn't necessarily mean that the model successfully captures the variation of the target variable. For example, an R2 score of 0.1 is vastly different than a score of 0.9. Remember that we are trying to *maximize* the R2 score.

Look into discussing how this model's R2 score compares to the 'optimal' R2 score. Or how do the 'true values' and 'predictions' compare.

Remember that 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.

> "The main benefit to having a testing subset is to verify that the model hasn't been overfitted to the training data; in other words, that the model we have isn't specific to the data such that it won't generalise well to new data"

This sums it up quite well! In short, 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. And correct 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.

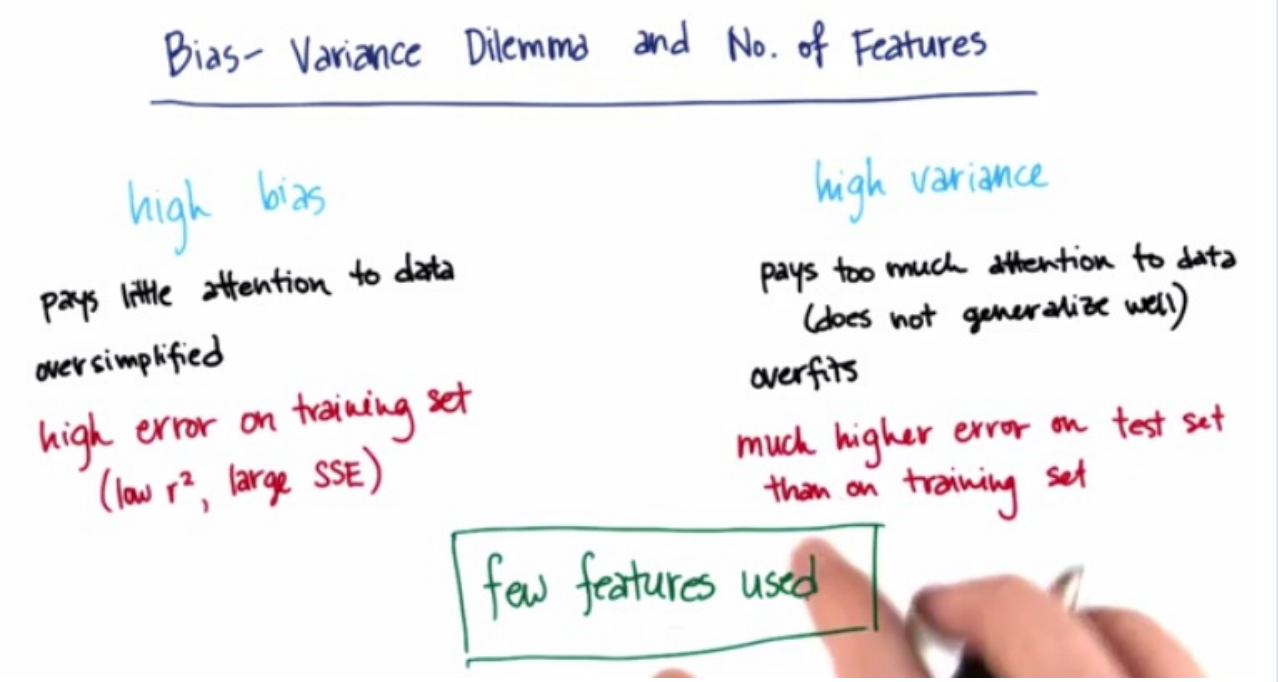
To go into a bit more detail in terms of the training and testing curves

- 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 generalizes 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.

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.

Much better and nice justification here! 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.

Much better, as I just needed to make sure that you clearly understood that we can use any evaluation metric we please in gridSearch.

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.

> "n k-fold cross-validation, the training data is broken up into n buckets; training is done n times, using n-1 buckets for training and the unused bucket for cross-validation. Each iteration, a different bucket is used for CV. Finally, the iterations are averaged to produce a model."

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

> "This mechanism allows us to detect high variance models (and the most general model) earlier while allowing us to retain the full training set for training purposes."

The word "detect" probably isn't the best to use here. But cross-validation better **estimates the volatility** by giving you the average error rate and will better represent generalization error. Using k-fold CV we perform grid search on *various validation set* so we select best parameter for generalize case.

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 testi
ng
X_train, X_test, y_train, y_test = cross_validation.train_test_split(iris.d
ata, iris.target, test_size=0.3, random_state=0)

# Build an SVC model for predicting iris classifications using training dat
a
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" value
s, 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

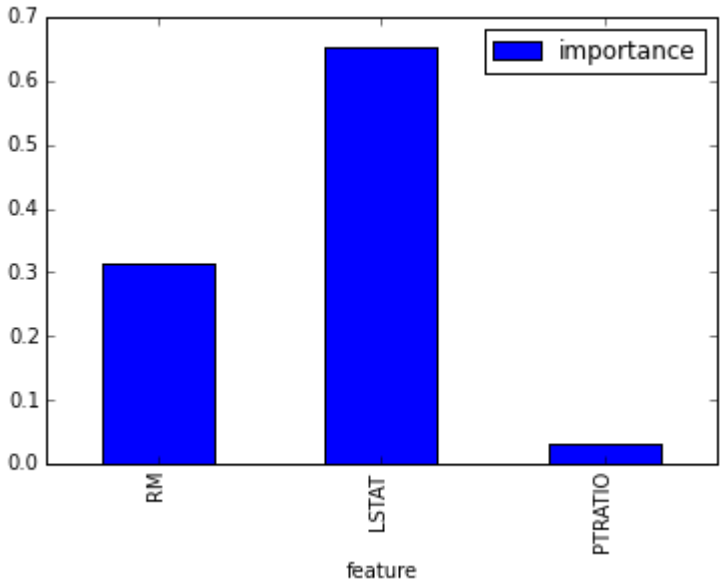
Student correctly implements the `fit_model` function in code.

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

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=['feat
ure', 'importance']).set_index('feature').plot(kind='bar')
```



You will see this more in the next project!

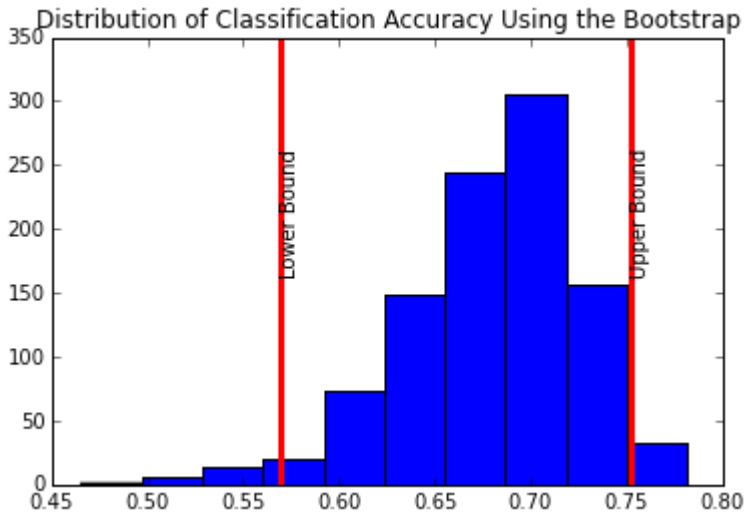
Student thoroughly discusses whether the model should or should not be used in a real-world setting.

Maybe one other idea would be to plot and 95% confidence interval to determine if the model is robust with bootstrapping

```
from sklearn.utils import resample
import matplotlib.pyplot as plt

data = pd.read_csv('housing.csv')
values = data.values
# configure bootstrap
n_iterations = 1000
n_size = int(len(data) * 0.50)

# run bootstrap
stats = []
for i in range(n_iterations):
    # prepare train and test sets
    train = resample(values, n_samples=n_size)
    test = np.array([x for x in values if x.tolist() not in train.tolist
()])
    # model
    model = DecisionTreeRegressor(random_state=100)
    model.fit(train[:,:-1], train[:,-1])
    score = performance_metric(test[:,-1], model.predict(test[:,:-1]))
    stats.append(score)
```

```
# confidence intervals
alpha = 0.95
p = ((1.0 - alpha) / 2.0) * 100
lower = max(0.0, np.percentile(stats, p))
p = (alpha + ((1.0 - alpha) / 2.0)) * 100
upper = min(1.0, np.percentile(stats, p))

# plot
plt.hist(stats)
plt.axvline(lower, color='red', lw=3)
plt.text(lower, n_iterations // 4, 'Lower Bound', rotation=90)
plt.axvline(upper, color='red', lw=3)
plt.text(upper, n_iterations // 4, 'Upper Bound', rotation=90)
plt.title('Distribution of Classification Accuracy Using the Bootstrap')
plt.show()

print('%.1f confidence interval %.1f%% and %.1f%%' % (alpha*100, lower*100,
 upper*100))
```

(https://machinelearningmastery.com/calculate-bootstrap-confidence-intervals-machine-learning-results-python/)



RESUBMIT

DOWNLOAD PROJECT

Learn the best practices for revising and resubmitting your project.

RETURN TO PATH

Rate this review

Student FAQ