UDACITY Logout

PROJECT

Predicting Boston Housing Prices

A part of the Machine Learning Engineer Nanodegree Program

PROJECT REVIEW NOTES

Requires Changes

SHARE YOUR ACCOMPLISHMENT

1 SPECIFICATION REQUIRES CHANGES

This is a very impressive submission. Just need one adjustment and you will be golden, but also check out some of the other ideas presented in this review. 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

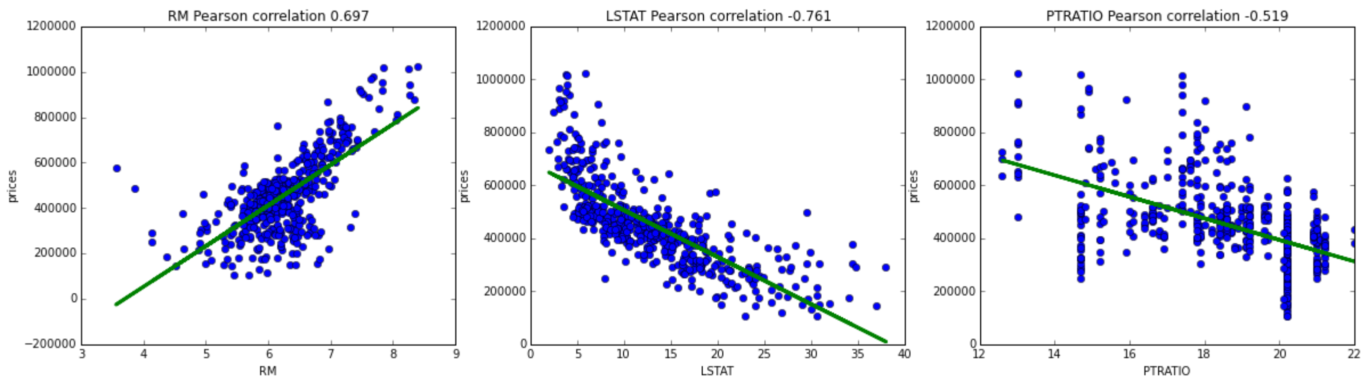
All requested statistics for the Boston Housing dataset are accurately calculated. Student correctly leverages NumPy functionality to obtain these results.

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. 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+1)
    plt.plot(data[col], prices, 'o')
    fit = np.polyfit(data[col], prices, 1)
    plt.plot(data[col], data[col] * fit[0] + fit[1], lw=3)
    plt.title(col + ' Pearson correlation ' + str(np.round(np.corrcoef(data[col], prices)[1][0], 3)))
    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.

Would recommend expanding a bit more. How does this result compare to the optimal score? How do the true values and predictions compare? etc... Could also think about if more data points would allow us to be more confident in this model?

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.

> "If almost all the dataset is used to train the model, the model will obtain very good results (low error rate) on training set, but poor performance on the training set. This is overfitting: the model modeled the training dataset too well, it learned the details in the training dataset to the point that it can't generalize to new data.
> On the other hand, if the training dataset is too small/not enough to the point that the model does not fit the training data, missing the trends in the data, this means the model underfitted. An underfitted model will have also poor performance on the training dataset."

The actual data split doesn't really have an effect on overfitting or underfitting, as the actual machine learning model controls these aspects. Therefore these comments aren't necessarily true. 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.

Therefore just focus on *why* we actually need a testing set. What can we try and 'protect against' or 'detect' being able to test it with an independent dataset? etc...

Links

- Lecture
- (http://info.salford-systems.com/blog/bid/337783/Why-Data-Scientists-Split-Data-into-Train-and-Test)

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 learning curves suggest that is unnecessary to have more training points after 300, where training and testing curves begin to run in parallel."

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.

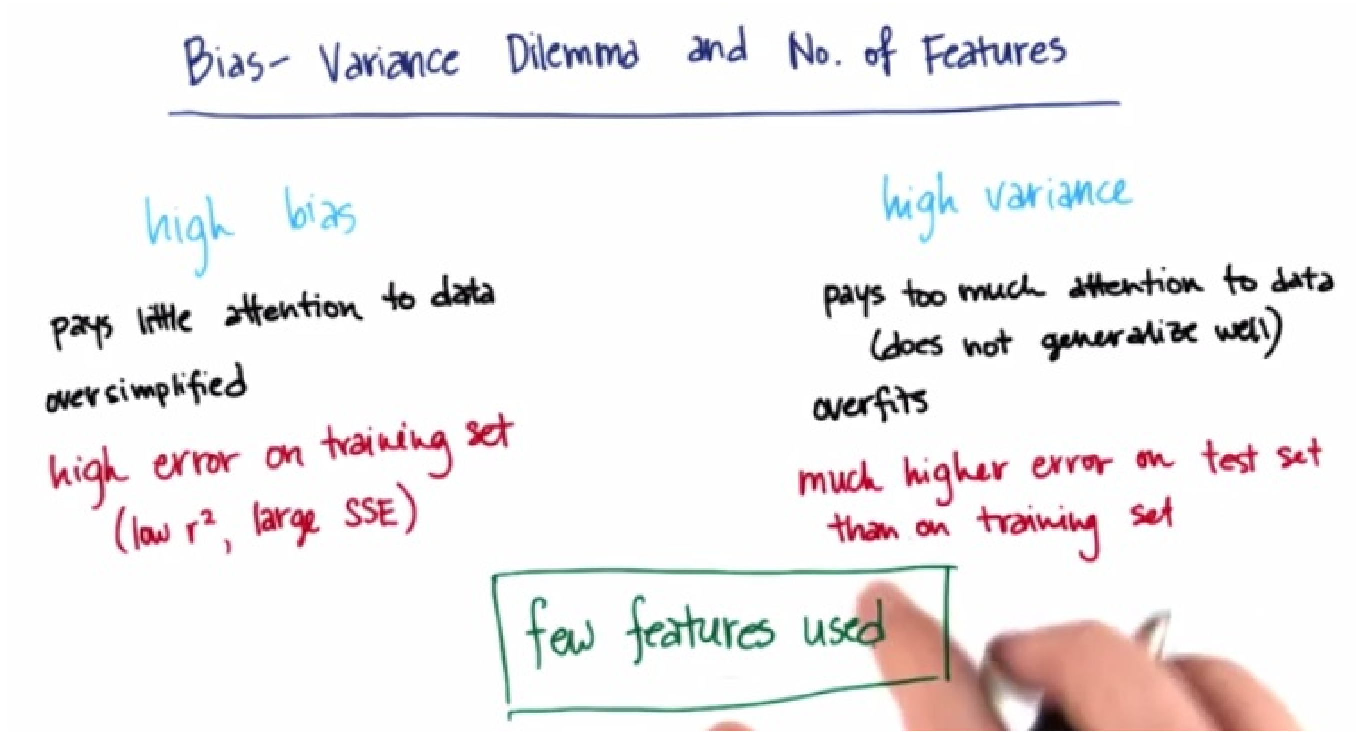
Also note that in practice collecting more data can often be time consuming and/or expensive, so when we can avoid having to collect more data the better. Therefore sometimes receiving very minor increases in performance is not beneficial, which is why plotting these curves can be very critical at times.

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

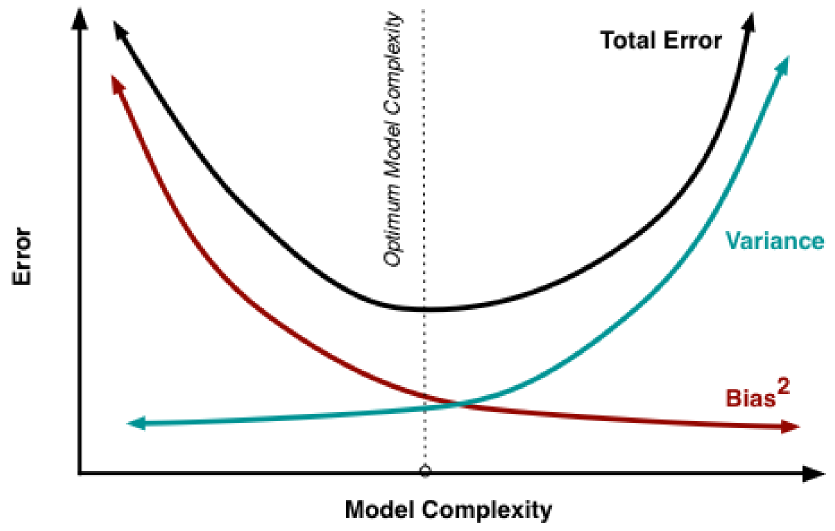

Bias-Variance Dilemma and No. of Features
high bias
pays little attention to data
oversimplified
high error on training set
(low r², large SSE)
high variance
pays too much attention to data
(does not generalize well)
overfits
much higher error on test set than on training set
few features used

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 the possibilities of parameter values*", 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.

> "Applying only Grid Search in order to find the optimal hyper parameter values for a model can lead to a model tuned/learnt from only to a specific subset - overfitting the model (because the training and testing set are always the same)."

This is an extremely important concept in machine learning, as this allows for multiple validation 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 there 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 sets so we select best hyperparameter for generalization. Cross-validation better estimates the volatility by giving you the average error rate and will better represent generalization error.

NOTE

> " which helps the model to find the best hyper parameter values over all the datapoints."

Remember, when optimizing a model with gridSearch, we only run k-fold cross-validation on the **training dataset**. As we still need to split the data into a training and testing set, since we still need that last testing set for the final model evaluation. You can see that we are using `fit_model(X_train, y_train)` in this project. When training a model to evaluate different hyperparameters, the testing set should never be used to train the model.

Therefore we don't need to break off a piece of our training dataset solely for validation purposes, which is why this is ideal with smaller datasets. It is common to use a train/validation/test split when tuning machine learning models to prevent overfitting on the test dataset. But setting aside a validation set reduces the number of samples which can be used for training, which is why we often run k-fold cross-validation on the **training dataset** to keep as much training data as possible.

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.

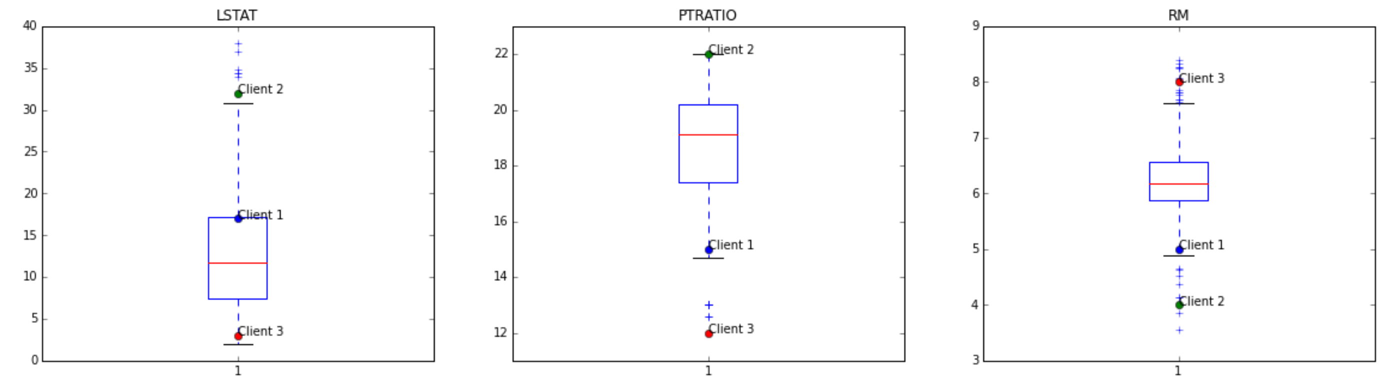
Student reports the predicted selling price for the three clients listed in the provided table. Discussion is made for each of the three predictions as to whether these prices are reasonable given the data and the earlier calculated descriptive statistics.

Excellent justification for these predictions by comparing them to the descriptive stats of the features. Love the ideas. Just remember to keep in mind the testing error here.

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

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.

Would agree. 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 PROJECT

DOWNLOAD PROJECT

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)