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

I am very impressed with your understanding here. 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

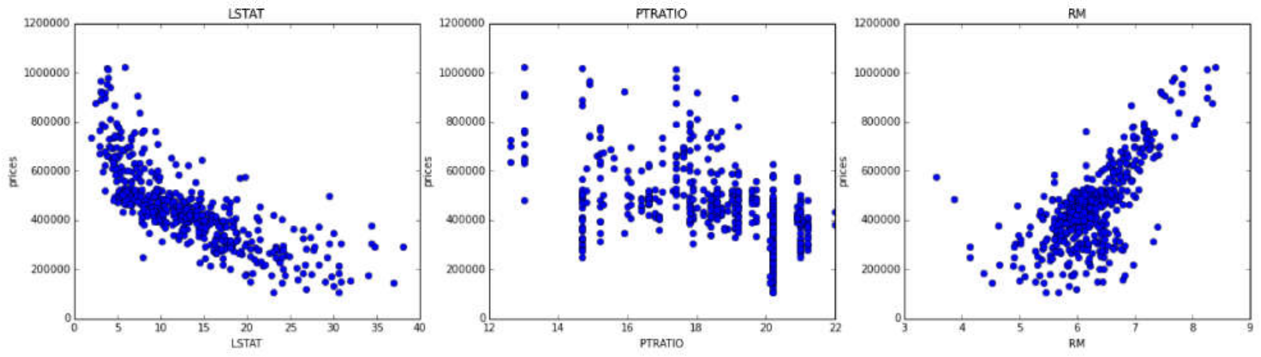
All requested statistics for the Boston Housing dataset are accurately calculated. Student correctly leverages NumPy functionality to obtain these results.

Nice adjustment here and great job utilizing the power of Numpy to receive these statistics!!

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)
    plt.plot(data[col], prices, 'o')
    plt.title(col)
    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.

Nice with your comment of "*A model predicting the mean/expected value would get an R^2 of only 0, explaining no variance at all.* " It seems you really have a good grasp on r^2.

- R-squared = Explained variation / Total variation

Code Note: We can also check out the linear relationship between the 'True Values' and 'Predictions' with a 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_inde
x()
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.

"*we have to analize the performence of our model, so that we can know how accurately our model predicts the target value for the unseen data. So we divide our data set into training and testing sets so that we can simulate and see what whould happen if we applied it to new data in reality.*"

Awesome!! 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.

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.

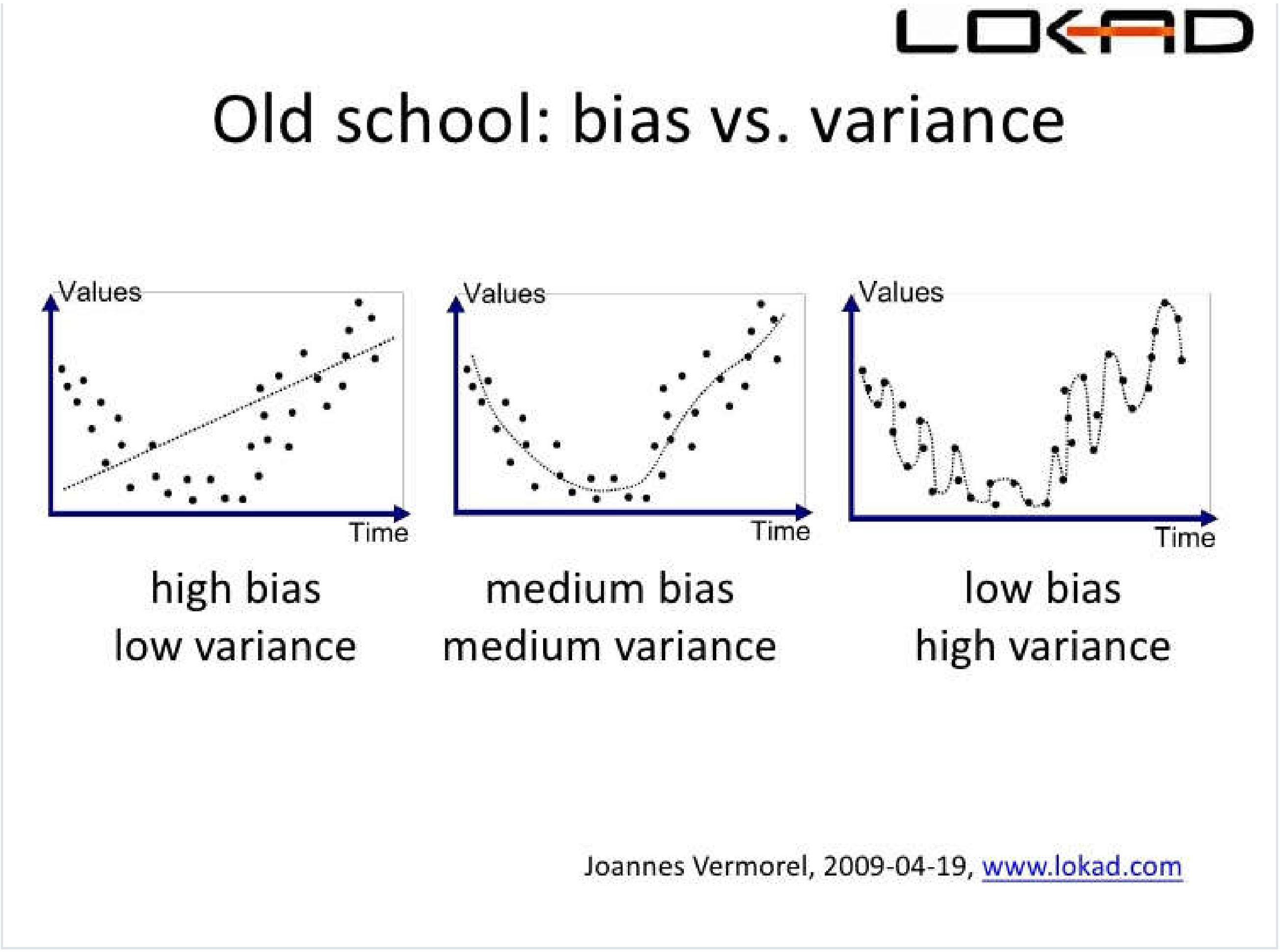
Love your analysis. As the training score decreases and testing score increases makes sense, since with little amounts of the data we simply memorize the training data(no generalization), then when we receive more and more data points we can't simply memorize the training data and we start to generalize better(higher testing accuracy)

And you are correct that "*There after even though more training points included there is not much effect on the both scores, I don't see any benifit of adding more training points for the same model complexity.*" If we look at the testing curve here, we can clearly see that it has converged from 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.

Glad that you have analyzed multiple max depths as your analysis exceeds expectations here.

- As a max_depth of 1 suffers from high bias, visually this is due to the low training and validation scores(also that they 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.

I would choose the same based on the visual here. Great justification for your choice.

Evaluating Model Performance

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

Great idea to bring up max depth from this project, as this is a great way of understanding how this technique works. As GridSearch simply is an exhaustive search through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out 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.

Good work with your description, as K-Fold CV is probably the most used CV technique in practice.

Really like your discussion in terms of the benefits of its application when used with grid search when optimizing a model. Especially with your comment of "*with more and more grid points, we are more and more likely to find a point that is good only by chance.* " As we are not reliant on one data split for parameter tuning, reducing the chances of overfitting. 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.

Note: Just note that we only perform gridSearch and K-Fold on the **training set data**, shown here with `reg = fit_model(X_train, y_train)` since we still need that last testing set for the final evaluation.

Student correctly implements the `fit_model` function in code.

You have correctly implemented GridSearchCV. Nice use of the `range` command. Just note that we actually don't need to create a `list` and can simply do

```
params = {'max_depth':range(1,11)}
```

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

Good work! GridSearch searches for the highest validation scores on the data splits.

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

Good justification for these predictions by comparing them to the housing prices and features of the dataset. Just remember to keep in mind the testing error here.

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
plt.hist(prices, bins = 30)
```

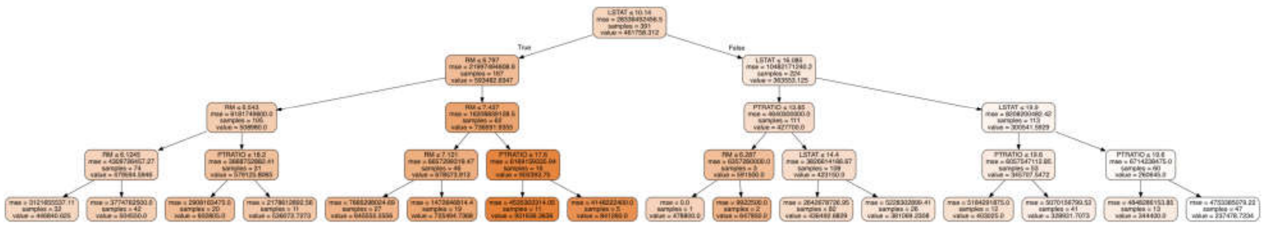
```
for price in reg.predict(client_data):
    plt.axvline(price, c = 'r', lw = 3)
```

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 since 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

Have a question about your review? Email us at review-support@udacity.com and include the link to this review.

RETURN TO PATH