UDACITY

PROJECT

Predicting Boston Housing Prices

A part of the Machine Learning Engineer Nanodegree Program

PROJECT REVIEW | CODE REVIEW | NOTES

Meets Specifications

SHARE YOUR ACCOMPLISHMENT

Please check out the gridSearch comment in the corresponding section, but you are ready to move on.

Data Exploration

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

All correct! And good use of Numpy!

Student correctly justifies how each feature correlates with an increase or decrease in the target variable.

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.

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.

You have captured the need of a testing set for evaluation of our model. 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.

> "Increase in training points: An increase in training points makes the model to generalize, so as we increase the training points the training error increases as the model is trying to generalize which causes a reduction in testing error. So in the graphs for different depths as the training score reduces as the training points increase but the testing score increase as the training points increase."

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).

Rate this review

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.

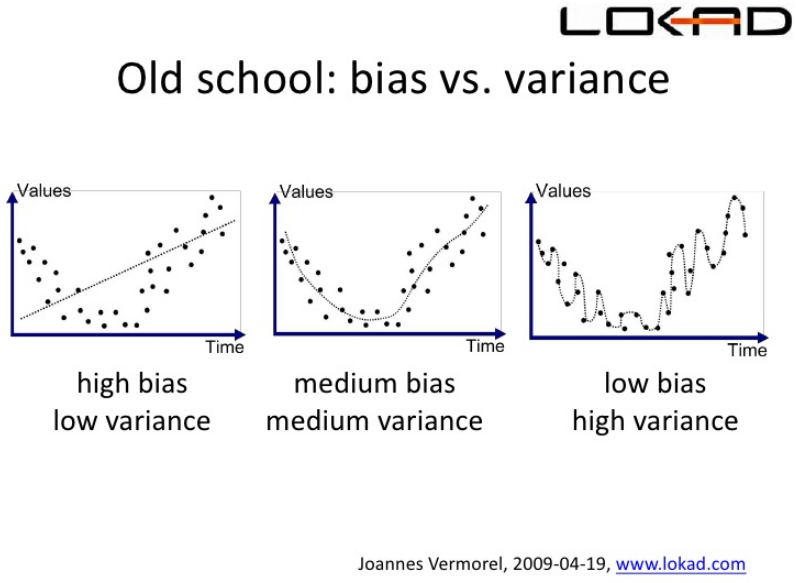
> "So at a max depth of 1 the model shows that both the training score and the testing score are less which means the training error and testing errors are high and the model could not predict the actuals to a reasonable level. At a max depth of 1 model suffers from high bias."

Spot on! 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

---

> "As the testing score reduces beyond 4, the training score increases, this corresponds to reducing bias as model complexity increases. In conclusion the training score is closer to 1 and the testing score is farther from the training score which is lesser, this indicates high variance."

Good. 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.

Nice ideas! As GridSearch simply is an exhaustive searching 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.

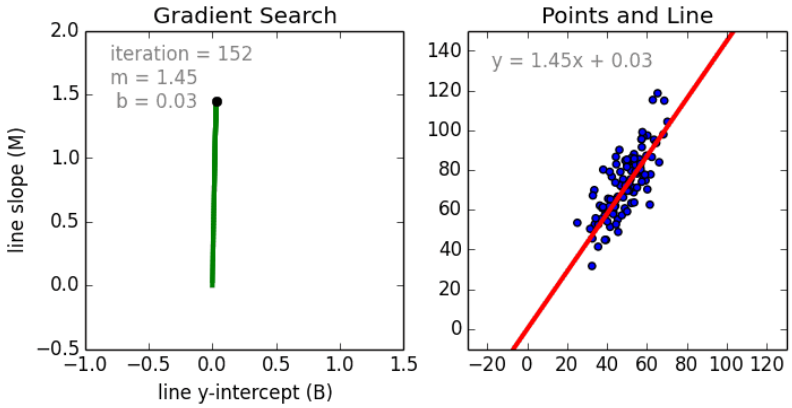
If you would like to learn about some more advanced techniques and combining gridSearch with other techniques, with the notion of Pipelining, check out this blog post brought to you by Katie from lectures

- (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.

> "My answer is correct because the K-fold cross validation helps us determine the optional parameters such as in case of linear regression k-fold cross validation helps us determine optimal slope, intercept, and co-efficients of the model. So when various all these models optimized by k-fold cross validation are arranged in the form of grid, the grid search subsequently helps us in determining the hyper parameters such as quadratic or polynomial linear regression models."

Quadratic or polynomial for a linear regression model is a valid argument here. However "optimal slope, intercept, and coefficients of the model" are the **parameters** for the model, NOT the hyper-parameter values for the model. These are optimized by something like Gradient descent and how the model is actually **trained**.



In result the model will have different values for the slope, intercept, etc..., but the gridSearch algorithm does not change these values in the actual algorithm. This is a key difference.

Let's understand what the grid search does with the example we have. 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 out 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.

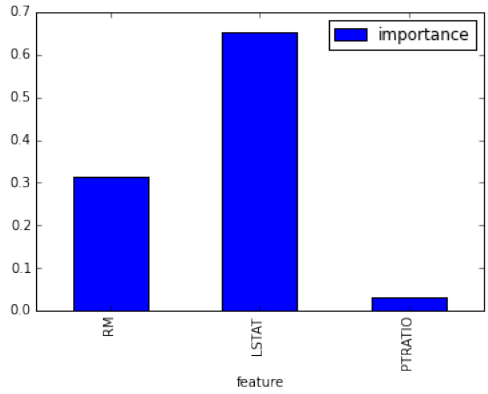
Student correctly implements the `fit_model` function in code.

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



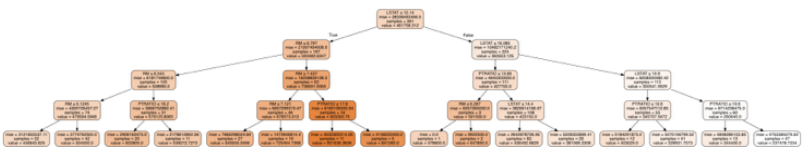
You will see this more in the next project!

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

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