UDACITY

‹ Return to Classroom

Predicting Boston Housing Prices

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Very nice adjustments here, check out the corresponding sections for even more insight! 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.

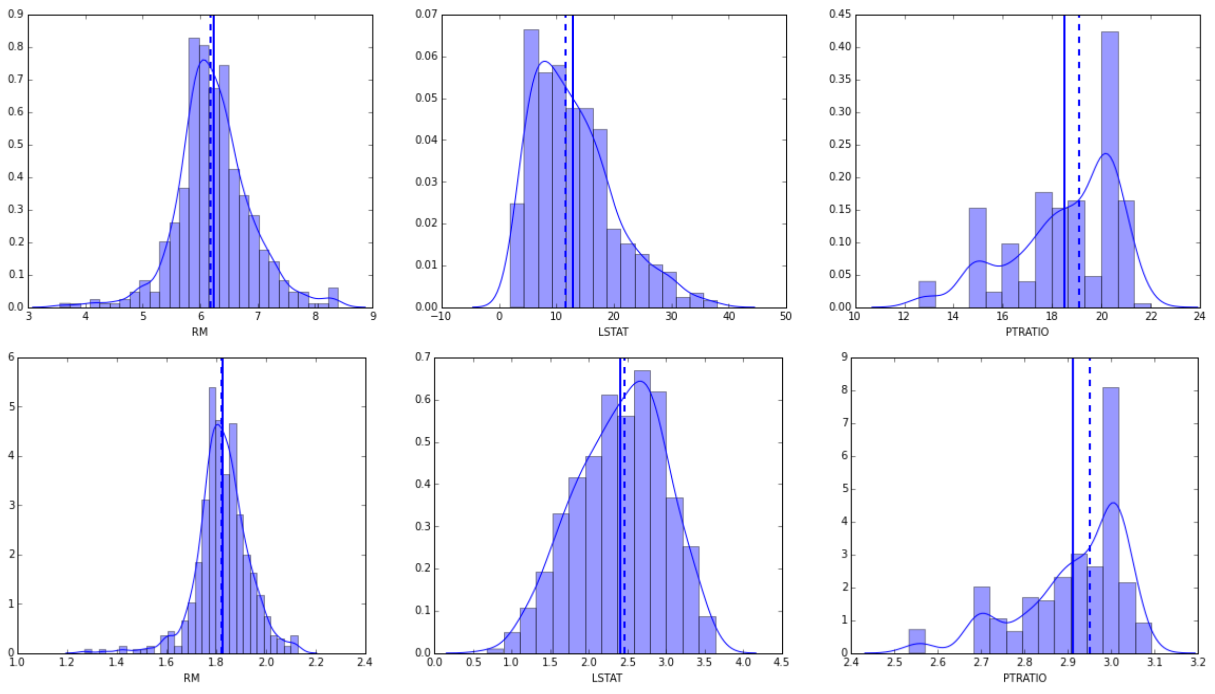
Typically, in machine learning we desire to have our features to be Gaussian distributed. Therefore, could also plot histograms. Do we need any feature transformations? Maybe a log transformation could be ideal.

```
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.

Would recommend expanding a bit more. How does this result compare to the optimal score? How do the true values and predictions compare? etc... Might want to check out this explanation of how R2 score works. Could also think about if more data points would allow us to be more confident in this model?

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.

Great ideas. 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.

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).

> "Generally speaking, the more training data there is for the machine to learn, the better the model would be. However, it comes at the expense of less testing data for examination, which is also very important. If the two curves converge at some point, there would be no need to have more training data."

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.

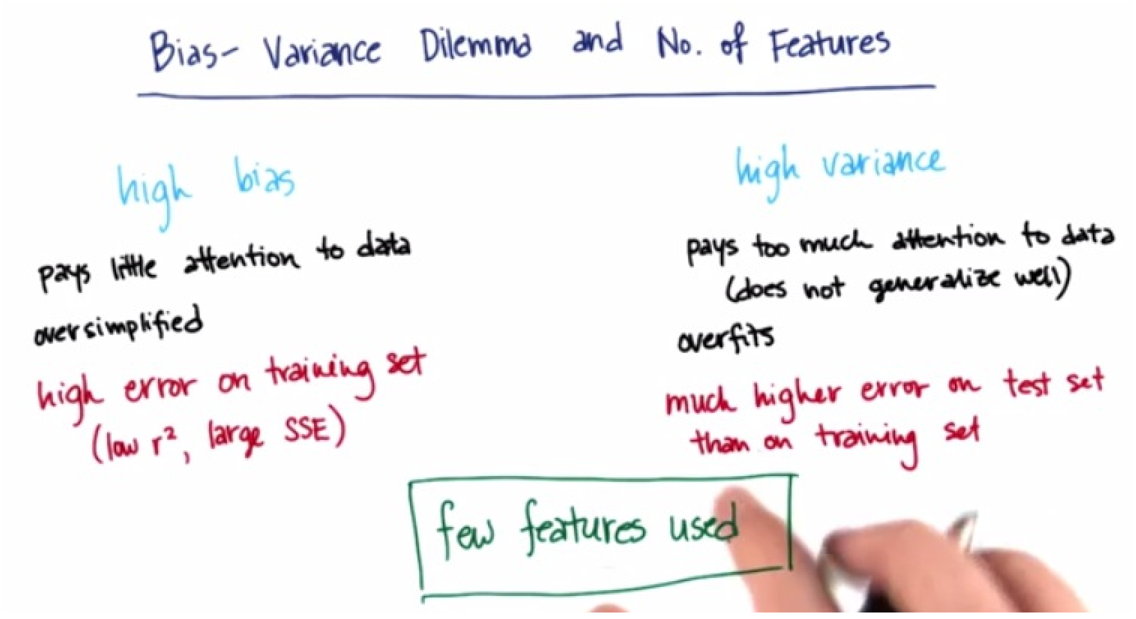
Student correctly identifies whether the model at a max depth of 1 and a max depth of 10 suffer from either high bias or high variance, with justification using the complexity curves graph.

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

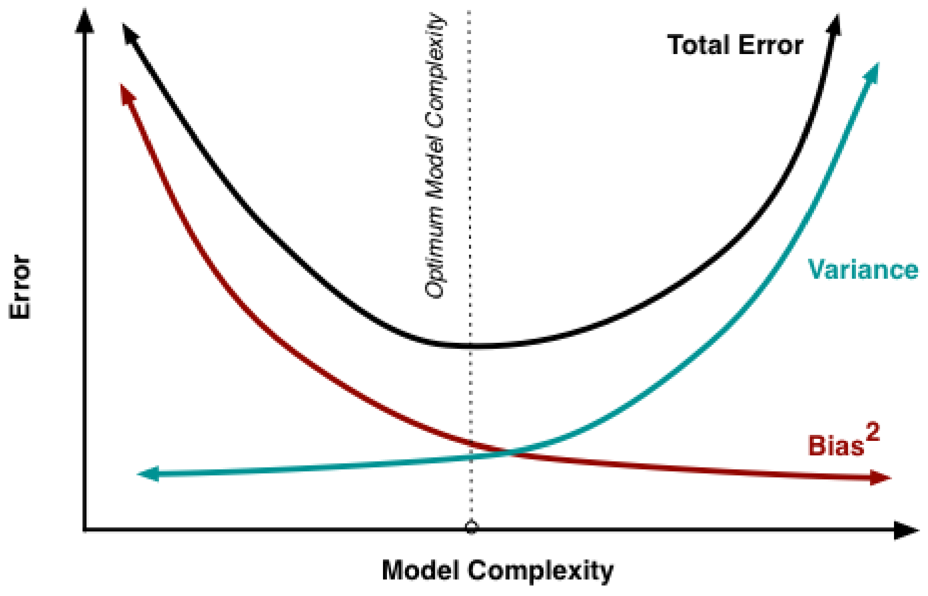
new unseen data


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.

I would choose the same! As we are definitely looking for the highest validation score(which is what gridSearch searches for). And we are also looking for a good bias / variance tradeoff(with close training and validation scores). And the less complex one is definitely recommended 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.

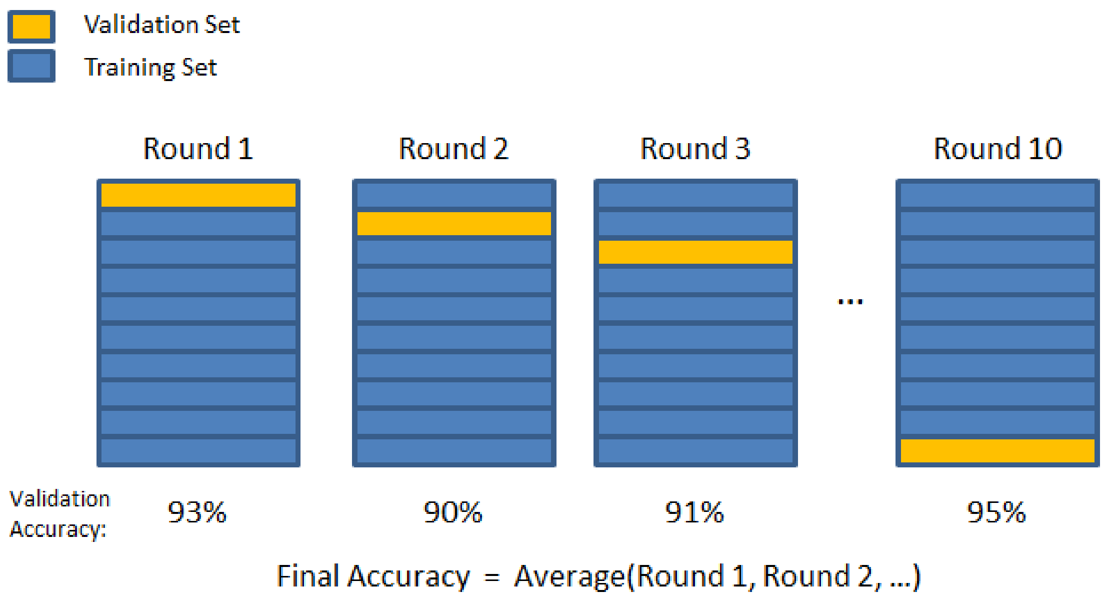
Nice ideas! As GridSearch simply is an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set.

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.

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.

To give some detail in terms of the entire process here. I would say that k-fold cross-validation with 10 splits is described as the following:

- The training data is split into k folds, (cv = 10)
- We train a model on the k-1 folds, k times, and use the remaining fold as the validation set (9 for train and 1 for validation here)
- For each model we calculate the validation error (10 errors rates here)
- Then at the end of the k-fold cross-validation technique, all of the validation error rates are averaged together. (single number)



Then after we get the results from hyper-parameter tuning from gridSearch with k-fold cross-validation, we run the model on the held out testing dataset to determine how this model would work with **never before seen data.**

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

The K-Fold CV technique allows us not to set aside a validation dataset. It is common to use a validation to tune the model to prevent overfitting on the test dataset. Setting aside a validation and test sample then reduces the number of samples which can be used for learning. While the test set is important to set aside, CV allows us to remove the reliance on a separate validation dataset which is beneficial when using a small dataset.

The additional benefit is to prevent overfitting from over-tuning the model during grid search. There is a risk of overfitting on the test set because the parameters can be tweaked until the estimator performs optimally. 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 (or tune hyper-parameters solely the training 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.

Student correctly implements the `fit_model` function in code.

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

Congrats! 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. Love the ideas. Just remember to keep in mind the testing error here.

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

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

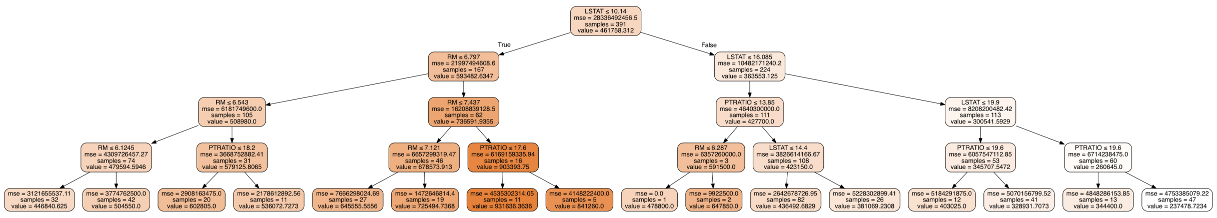
```
import matplotlib.pyplot as plt
for i,price in enumerate(reg.predict(client_data)):
    plt.hist(prices, bins = 30)
    plt.axvline(price, lw = 3)
    plt.text(price-50000, 50, 'Client '+str(i+1), rotation=90)
```

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

Rate this project