UDACITY

Logout

PROJECT

Predicting Boston Housing Prices

A part of the Machine Learning Engineer Nanodegree Program

PROJECT REVIEW | CODE REVIEW | NOTES

Meets Specifications

SHARE YOUR ACCOMPLISHMENT

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.

All correct! And good use of Numpy!

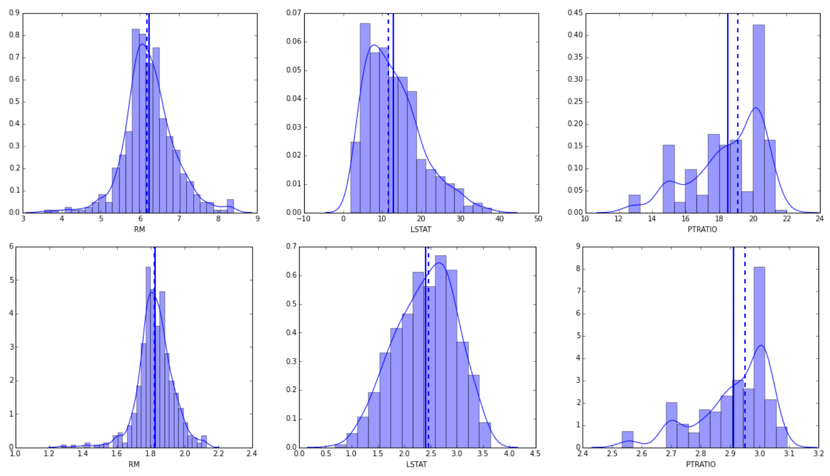
Student correctly justifies how each feature correlates with an increase or decrease in the target variable.

Love the regplot idea! Could also plot histograms. Do we need any data 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
)
```

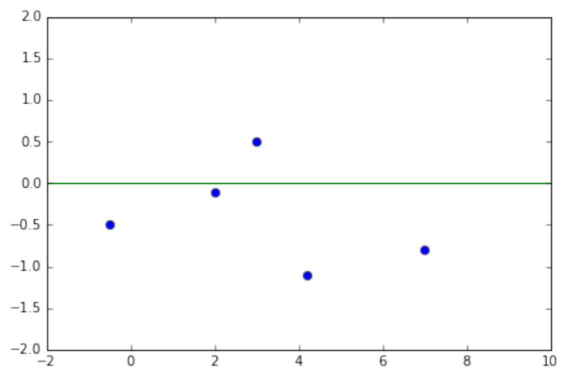


Developing a Model

**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.**

Another interesting thing to check out for regression problems would be a residual plot. A residual plot is a graph that shows the residuals on the vertical axis and the independent variable on the horizontal axis. If the points in a residual plot are randomly dispersed around the horizontal axis, a linear regression model is appropriate for the data; otherwise, a non-linear model is more appropriate.

```
import matplotlib.pyplot as plt
trueValues = [3, -0.5, 2, 7, 4.2]
predictions = [2.5, 0.0, 2.1, 7.8, 5.3]
residuals = [i - j for i, j in zip(trueValues, predictions)]
plt.plot(trueValues, residuals, 'o',[-2,10], [0,0], '-')
plt.ylim(-2, 2)
plt.tight_layout()
```



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 reasons! 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.

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).

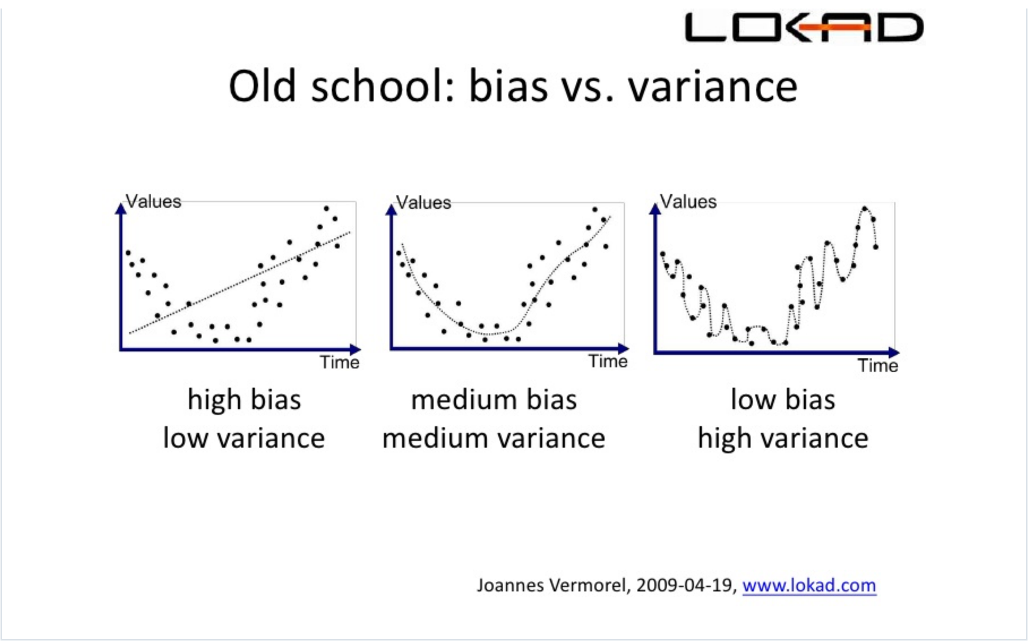
As in the beginning it is beneficial, but at the end if we look at the testing curve here, we can clearly see that it has converged to its optimal score, so more data is not necessary.

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.

If you would like to learn more, check out these links and visual

- http://scott.fortmann-roe.com/docs/BiasVariance.html
- http://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning/
- http://insidebigdata.com/2014/10/22/ask-data-scientist-bias-vs-variance-tradeoff/



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.

> "Take depth in decision tree for exameple, grid search will try to find the best fitting depth as its goal by trying all settings of depth"

To expand on how this works -- 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 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.

> "One of the main benefits for using cross-validation instead of using the conventional validation (e.g. partitioning the data set into two sets of 70~80% for training and 20~30% for test) is that there is not enough data available to partition it into separate training and test sets without losing significant modelling or testing capability. Cross-validation combines (averages) measures of fit (prediction error) to derive a more accurate estimate of model prediction performance."

It is one benefit, actually not the main, however. Consider this -- 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 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. Thus cross-validation better estimates the volatility by giving you the average error rate and will better represent generalization error.

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.

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.

> "But I wonder why it doesn't look at the gap between testing score curve and validation curve."

It would be much harder to actually optimize this and also note that the testing score represents new, unseen data. So whenever this score is maximized, this means the best prediction for future data.

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.

Great feature*importances* plot here! Excellent justification for these predictions by comparing them to the features and descriptive stats of the housing prices. This is always an important step in any machine learning project.

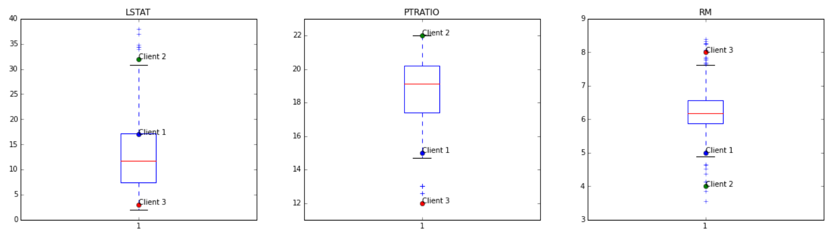
A more advanced and great idea would be to compare these to the descriptive stats of the features. We can compute the five number summary of the descriptive stats of the features with

```
features.describe()
```

For example -- Client 3 RM value is in the top 25 percentile in the data, while being near the minimum value for LSTAT, and since an increase in RM would lead to an increase in MEDV. An increase in LSTAT would lead to a decrease in MEDV. The predicted price near the maximum value in the dataset makes sense.

Maybe also plot some box plots and see how the Client's features compare to the interquartile range, median, whiskers

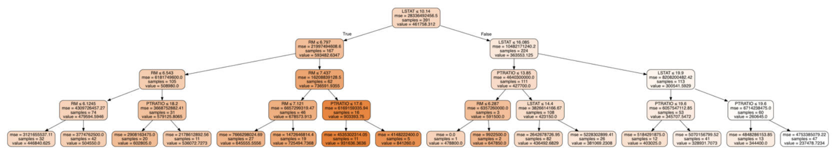
```
import matplotlib.pyplot as plt
plt.figure(figsize=(20, 5))
y_ax = [[3,9],[0,40],[11,23]]
for i, col in enumerate(features.columns):
    plt.subplot(1, 3, i+1)
    plt.boxplot(data[col])
    plt.title(col)
    for j in range(3):
        plt.plot(1, client_data[j][i], marker="o")
        plt.annotate('Client '+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.

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
# using a depth of 4 for visualization purposes.
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