UDACITY

PROJECT

Predicting Boston Housing Prices

A part of the Machine Learning Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Requires Changes

1 SPECIFICATION REQUIRES CHANGES

This is a very impressive submission. Just need one minor 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.

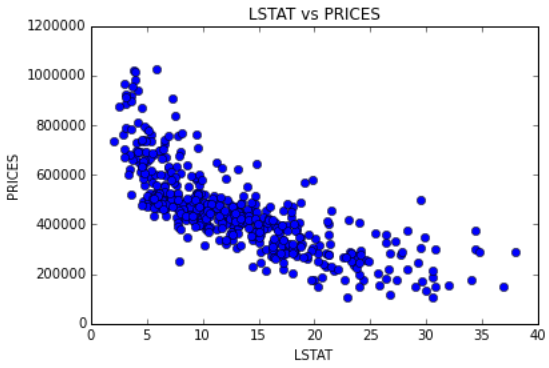
Student correctly justifies how each feature correlates with an increase or decrease in the target variable.

> "hence MEDV will be higher as LSTAT value increases"

Really sorry that the previous reviewer missed this, but relook at this comment. You have this backwards. As LSTAT is the percentage of all Boston homeowners who have a greater net worth than homeowners do in the neighborhood we are considering. Therefore, the higher it is, the lower is the net worth of the actual people in the neighborhood compared to the rest, that is the reason for the negative correlation: It means that the neighborhood has a lower net worth than average so you should expect lower housing prices.

As we can confirm this by plotting the LSTAT vs the housing prices

```
import matplotlib.pyplot as plt
plt.plot(features.LSTAT, prices, 'o')
plt.title('LSTAT vs PRICES')
plt.xlabel('LSTAT')
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 ideas here. Could also think about if more data points would allow us to be more confident in this model? Maybe even look into hand computing this with

```
y_true_mean = np.mean(y_true)
SSres = sum(np.square(np.subtract(y_true, y_predict)))
SStot = sum(np.square(np.subtract(y_true, y_true_mean)))
score = 1.0 - SSres/SStot
```

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.

In short, 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. Also note 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.

Really should also mention that in the initial phases, the training score decreases and testing score increases. Which 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).

> "Having more training points always benefits the model, algorithm gets trained well. But in this case, we see that training and testing scores are converging at one point and at a decent score, hence adding more training points in this case is less likely to be beneficial"

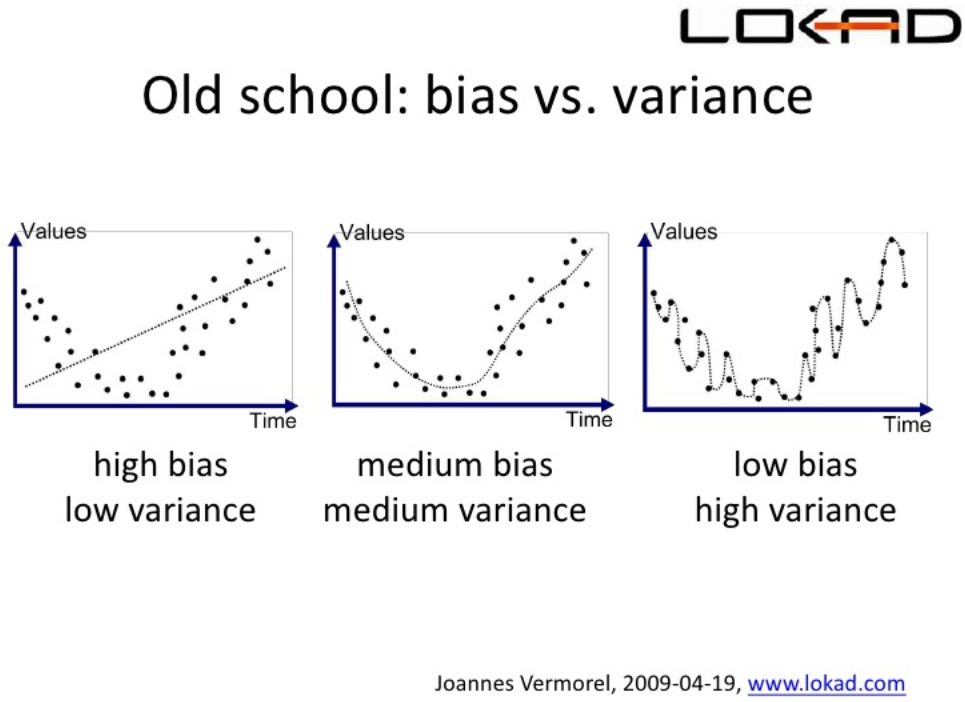
Correct! As in the beginning it is beneficial, but at the end if we look at the testing curve here, we can clearly see that it has converged to its optimal score, so more data is not necessary.

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

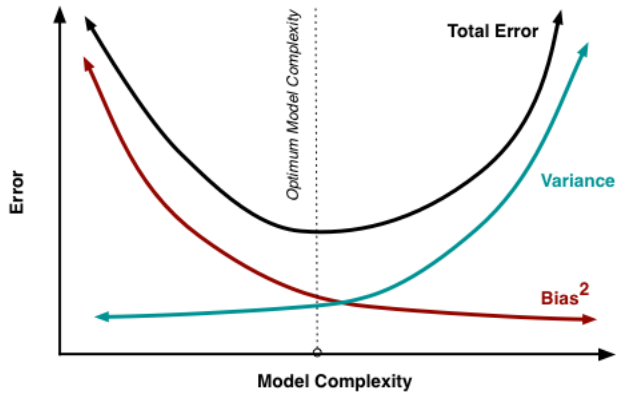


Student picks a best-guess optimal model with reasonable justification using the model complexity graph.

> "Bias and variance trade-off is optimal ie. model doesnt seem to overfit the data but at the same time has a good training score. Similarly, validation score is the highest among other depths too."

Exactly! 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).

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 description! As you can see that we are using max depth, a decision tree and r square score in this project. Can also note that since this trains on the "cartesian product" of the parameters, 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.

> "During grid-search, using the same training/validation set for various combinations of hyperparameters might lead to the model learning the data-points (ie. possibility of overfitting) and hence we could get misleading results. This issue is mitigated with the use of k-fold, as the training/validation sets are not the same set always and hence better results."

Correct! This is an extremely important concept in machine learning, as this allows for multiple testing 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 testing
X_train, X_test, y_train, y_test = cross_validation.train_test_split(iris.data, iris.target, test_size=0.3, random_state=0)

# Build an SVC model for predicting iris classifications using training data
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 clf.score(X_test, y_test)

# We give cross_val_score a model, the entire data set and its "real" values, 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 scores

# And the mean accuracy of all 5 folds:
print scores.mean()
```

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.

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 and descriptive stats. 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.

> " First, its very inconsistent - range of prices from the different trials is very high and inconsistent. Secondly, number of features used does not totally do justification to the quality of the house and it pricing. There needs to be additionals features to make better predictions. Third, the dataset is very outdated."

Would agree, as this is quite a large range, the dataset is quite old and probably doesn't capture enough about housing features to be considered robust!

RESUBMIT

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)

RETURN TO PATH

Rate this review

Student FAQ