Predicting Boston Housing Prices

Requires Changes

3 SPECIFICATIONS REQUIRE CHANGES

Dear student,

This is an excellent first submission! 👏
There are few minor things I'd like you to revise, but given your general understanding of the topic, they should be pretty easy for you. I can tell you're on a right path to becoming a great machine learning engineer!

I wish you the best of luck and 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.**

Great start!
You've correctly leveraged the power of NumPy to get the basic statistics of your data set. Remember you should *always explore* your data before trying to train a model on it. You'll see different variations of data exploration throughout this nanodegree.

It's always important to be aware of tools you use. For example, the Pandas' Series.std() will by default give you different result than numpy.std().

```
>>> pd.Series([7,20,22,22]).std()
7.2284161474004804
>>> np.std([7,20,22,22])
6.2599920127744575
```

The code was taken from the StackOverflow thread that also explains why this is the case.

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

Excellent intuition!
A good machine learning engineer should always validate their intuition with more data exploration. Remember visualisations are powerful way of finding the latent feature correlations and presenting your intuitions to your boss or a client.

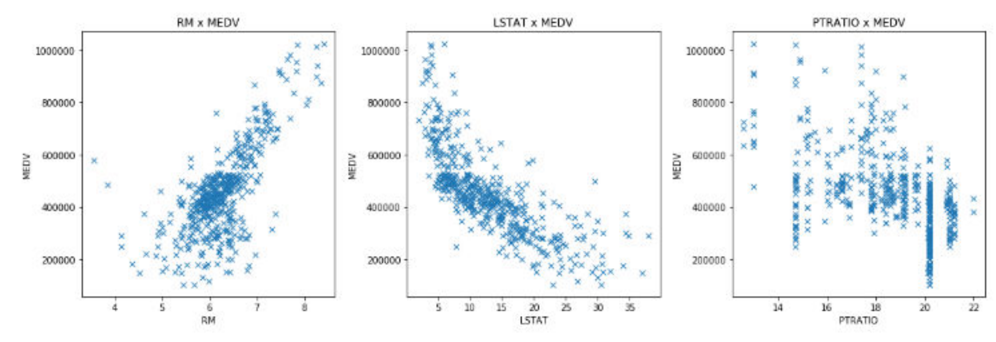
Here's a simple code snippet you can run in the Data Exploration section of the notebook to confirm your intuition about the data:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(15, 5))

for i, col in enumerate(features.columns):
    plt.subplot(1, 3, i+1)
    plt.plot(data[col], prices, 'x')
    plt.title('%s x MEDV' % col)
    plt.xlabel(col)
    plt.ylabel('MEDV')
```

You will get a plot similar to this one



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.**

You are correct.
The R^2 score of `0` means the model cannot successfully predict the target variable `y` from features `X`; while R^2 score of `1` means *the perfect* capture of the data variation—thus we can say having a score so close to 1 means we have a pretty decent fit.

On the other hand, we should always remember that evaluating any model on a single metric can be deceiving. I really recommend reading more about R^2 caveats—you will learn that the R^2 score won't tell you *everything* you need to know about your model's performance.

⟳ **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're almost there.

Although having *"a lot of"* training data can sometimes lead to overfitting model, the main sources of overfitting and underfitting are usually the model complexity (as in setting of hyper parameters).

So remember that splitting the dataset definitely **doesn't** automatically guarantee the model won't overfit.

Please rephrase your answer and think about the following:

- You train a model which performs well on the training data (so it's not underfitting).
- How do you know whether the model generalizes well on any data or is overfitting?

Would having a separate training and testing sets help in the scenario above (*knowing* whether it's a good 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 observations!
I would however argue that adding more points probably wouldn't be beneficial since the model's training/testing curves after ~300 data points are becoming flat with no tangible improvement. It's also important to note that collecting additional data in real life scenario might be really time consuming and expensive and doesn't ensure improvement in your model—thus it might be useful to plot something like these learning curves to determine whether additional data collection is required.

Please fix that in your answer.

However, note that some algorithms (mostly those in deep learning) can make use of more and more data to improve their performance.

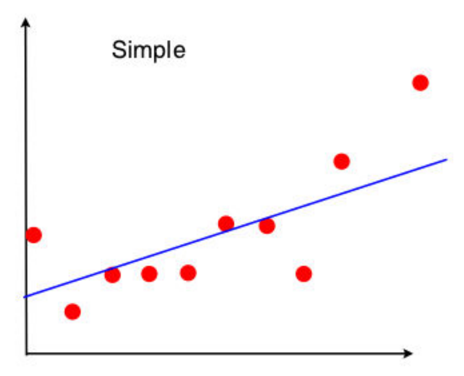
If you're interested in further reading on this topic, I can recommend these 2 articles:

- How much data is enough?
- How Much Training Data is Required for Machine Learning?

✓ **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.**

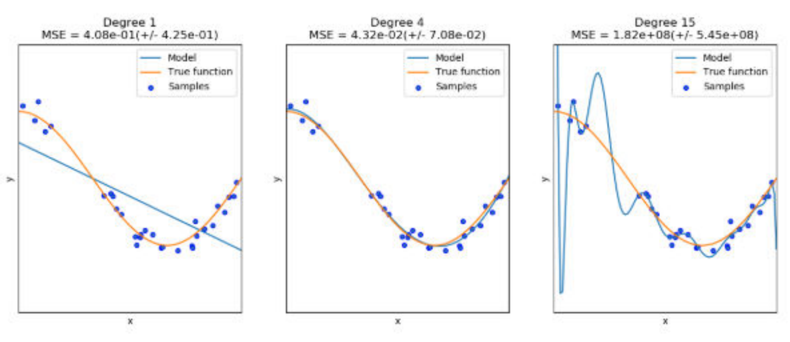
You have a pretty good understanding of bias/variance tradeoff—I like how you used the visual cues in the complexity curves graph to justify your answer.

It's also nice to visualise what the high bias/variance models look like when plotted over the actual data set. I think the following picture sums this up pretty well and can be even used to teach the basics of high bias/variance to complete novices.



(image source)

If you're interested, you can check out the sklearn's visualisation of high bias and variance.



Also, you can read more about bias-variance tradeoff on Scott Fortmann-Roe's blog.

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

Although the model with `max_depth` of `3` isn't *the best* choice here (as you found out later in the project), it is still *a very good choice* that shows you understand the complexity curves correctly.

Evaluating Model Performance

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

Looks like you understand grid search correctly.

Just note that we're tuning hyperparameters, not parameters. You can find more on the difference in this superb article.

Since grid search is an *exhaustive search* (meaning it has to train and evaluate a whole model for each hyperparameter combination), it's computationally expensive and memory inefficient. Fortunately for us, our beloved scikit-learn offers a better alternative I recommend checking out—RandomizedSearchCV.

If you're interested in how randomized search works in more detail, I recommend reading this article (Smarter Parameter Sweeps)

⟳ **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.**

You're almost there!
This is a decent description of k-fold CV, however it's missing few important parts:

- note that we only perform k-fold CV on the training data and still leave some testing set for final evaluation, that's a *really* important part
- *"on different subset of training data"* is not complete. Please specify explicitly how many bins are used for training/validation in each step and how they change step to step
- after you iterate K times, what next? How do you get a single model out of this? Do you somehow combine the results? If so, how?

If you ever find yourself in a position where you need to revise k-fold CV or teach this technique to someone else, I really recommend watching this elegant video on model selection and reading through the list of k-fold CV benefits in this Ritchie Ng's article on parameter tuning. I also believe these will help you complete this section 🙂

✓ **Student correctly implements the `fit_model` function in code.**

Your code implementation of grid search is correct.

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

Correct!

✓ **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 discussion!

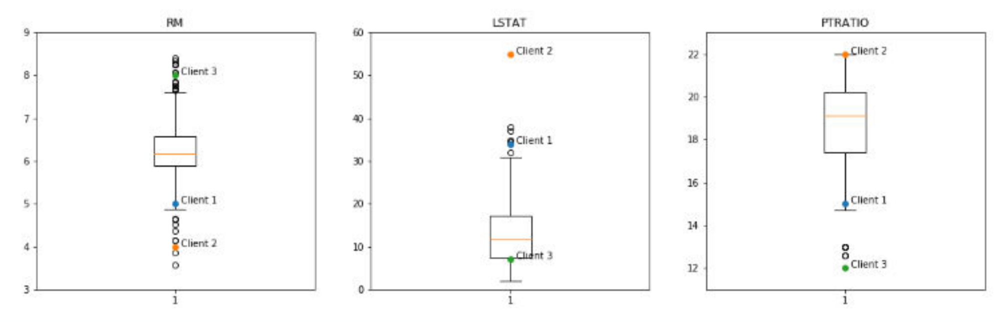
I would also recommend plotting the clients over the data set to *visualise* how their homes compare to the market in the individual features. You can do this easily by running the following snippet in the Question 10 section of your notebook.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(15, 5))

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 %s' % str(j+1), xy=(1, client_data[j][i]))
```

The plot will look something like this. Notice how these positions relate to your intuition about the feature correlations in Question 1 and compare them to your model's predictions.



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

You provided some great points on why this model shouldn't be used in real-world setting—and I couldn't agree with you more. Great job!

RESUBMIT PROJECT

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)