UDACITY

PROJECT

Predicting Boston Housing Prices

A part of the Machine Learning Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Excellent work, you nailed it!
👏👏👏

Awesome work you did there, advancing a lot, you should be really proud!

You have shown a firm grasp of the concepts presented here and are good to go.

Keep going and good luck!
Paul

PS. You can find me on Slack as `@viadanna`

Data Exploration

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

Good job calculating statistics, this exploration is very important as a starting point to understand the dataset.

Using `numpy` is recommended as it's a very fast and efficient library commonly used in machine learning projects. Check this reference on statistical functions included.

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

Nice intuition on the correlation between MEDV and the features.

This intuition is essential in machine learning projects to evaluate if results are reasonable.

Excellent use of plots to confirm your intuition 👍

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.

Great answer!

Metrics like this are important since the datasets found on Machine Learning projects rarely can be evaluated visually as this example.

It is possible to plot the values to get a visual representation in this scenario:

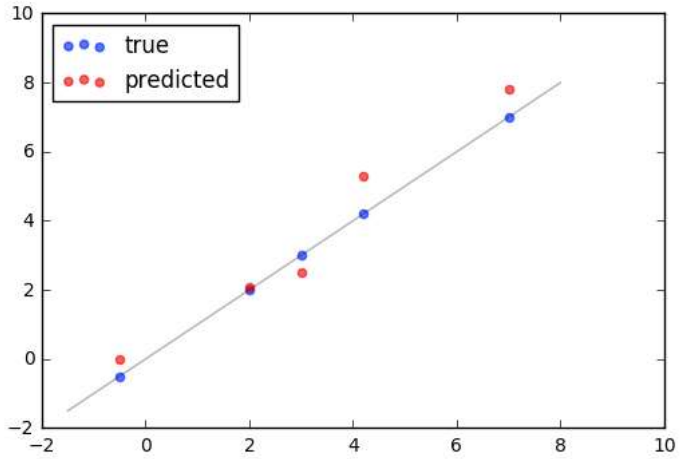
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

true, pred = [3, -0.5, 2, 7, 4.2], [2.5, 0.0, 2.1, 7.8, 5.3]
# Plot true values
true_handle = plt.scatter(true, true, alpha=0.6, color='blue', label='true')

# Reference line
fit = np.poly1d(np.polyfit(true, true, 1))
lims = np.linspace(min(true) - 1, max(true) + 1)
plt.plot(lims, fit(lims), alpha=0.3, color='black')

# Plot predicted values
pred_handle = plt.scatter(true, pred, alpha=0.6, color='red', label='predicted')

# Legend and show
plt.legend(handles=[true_handle, pred_handle], loc='upper left')
plt.show()
```



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.

Good answer!

It's really important to have an independent dataset to validate a model and identify its ability to generalize predictions. It is essential to identify *overfitting*.

Check out this article for an interesting further reading on this subject.

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.

Good job, there's little benefit in increasing the size of training dataset further.

Notice that it's easy for any model to memorize a few training points, that's why there's a large gap between training and testing accuracy with a small training size. As more points are added, it's increasingly difficult for the model to memorize all data points, and it starts to learn how to generalize instead.

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 answer, you correctly identified high bias and high variance.

In short, high bias means the model isn't complex enough to learn the patterns in the data, resulting in low performance on both the training and validation datasets.

On the other hand, high variance means the model starts to learn random noise, losing its capacity to generalize previously unseen data, resulting in a very high training score but low testing score.

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

Good choice! I'd choose the curve for `max_depth = 4` as well for the highest validation score.

The curve also shows a high validation score for `max_depth = 5`, but I'd still pick the above considering Occam's Razor which applied here basically means that for two or more equivalent answers, the simplest one is the correct one.

Evaluating Model Performance

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

Excellent description of GridSearch!

In short, GridSearch exhaustively searches for a combination of hyperparameters resulting in a model with the best performance.

Notice the *exhaustively* word? Training and validating a model for every combination of hyperparameters available is computationally intensive and can be unfeasible. If you are curious check the alternative RandomSearch.

Nice use of SVM as an example.

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.

Nice explanation of the k-fold technique.

In short, this technique will split the given dataset into K folds and iteratively use each one as validation set while using the rest as training set, averaging the results in the end.

I like to describe the benefit of this technique as being able to use the training set as multiple training and validation sets, preventing biases on the dataset from skewing the observed performance and causing GridSearch to return a model that's optimal for a subset of the dataset.

Student correctly implements the `fit_model` function in code.

Excellent implementation 👍

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

Well done!

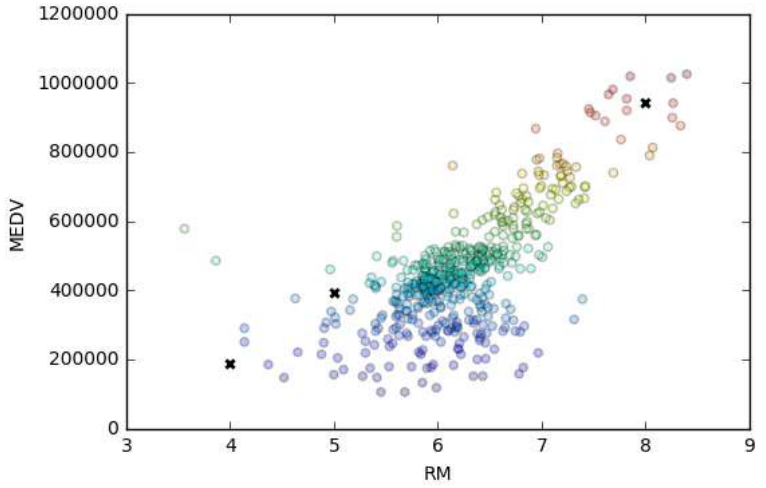
Student reports the predicted selling price for the three clients listed in the provided table. Discussion is made for each of the three predictions as to whether these prices are reasonable given the data and the earlier calculated descriptive statistics.

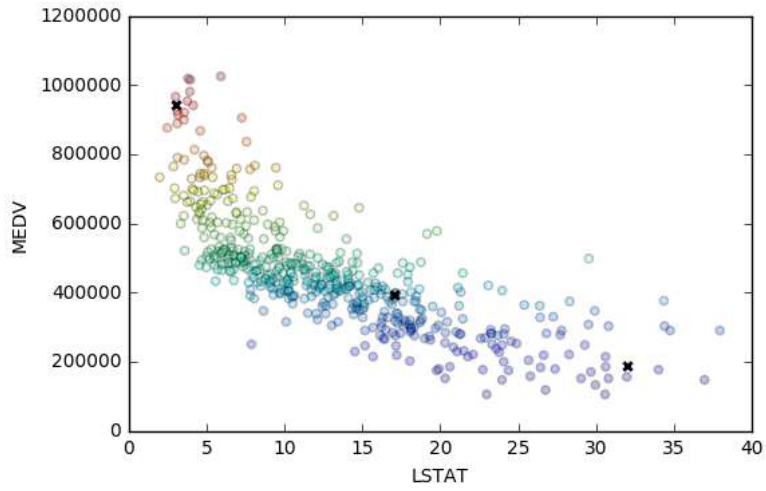
Good job using the features and statistics to justify your answer.

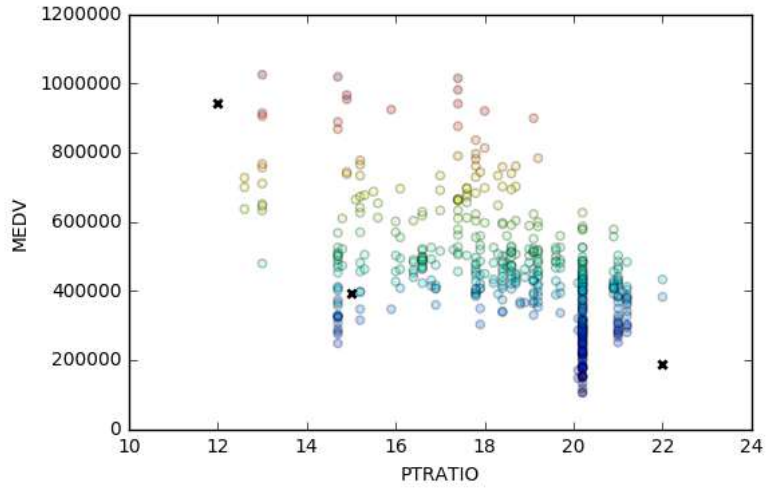
You could also plot the client data against the dataset to get a better view of their features:

```
from matplotlib import pyplot as plt

clients = np.transpose(client_data)
pred = reg.predict(client_data)
for i, feat in enumerate(['RM', 'LSTAT', 'PTRATIO']):
    plt.scatter(features[feat], prices, alpha=0.25, c=prices)
    plt.scatter(clients[i], pred, color='black', marker='x', linewidths=2)
    plt.xlabel(feat)
    plt.ylabel('MEDV')
    plt.show()
```







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

You have good arguments that show this model shouldn't be trusted in a real-world setting, lacking features and updated data.

I just missed the *Range in prices* shown in *Sensitivity* in your discussion of the robustness of this model.

DOWNLOAD PROJECT

RETURN TO PATH