Project 1 - Predicting Boston Housing Prices

Udacity - Machine Learning Nanodegree

Alexis Tremblay - 23 novembre 2015

Project Description

You want to be the best real estate agent out there. In order to compete with other agents in your area, you decide to use machine learning. You are going to use various statistical analysis tools to build the best model to predict the value of a given house. Your task is to find the best price your client can sell their house at. The best guess from a model is one that best generalizes the data.

For this assignment your client has a house with the following feature set: [11.95, 0.00, 18.100, 0, 0.6590, 5.6090, 90.00, 1.385, 24, 680.0, 20.20, 332.09, 12.13]. To get started, use the example scikit implementation. You will have to modify the code slightly to get the file up and running.

When you are done implementing the code please answer the following questions in a report with the appropriate sections provided.

Statistical Analysis and Data Exploration

```
Number of data points             : 506
Number of features                : 13
Minimum house price               : 5,000.00
Maximum house price               : 50,000.00
Mean house price                  : 22,532.81
Median house price                : 21,200.00
Standard deviation of house prices : 9,188.01
```

Evaluating Model Performance

Which measure of model performance is best to use for regression and predicting Boston housing data?

Mean Absolute Error

Why is this measurement most appropriate? Why might the other measurements not be appropriate here?

We have seen a few different measurement for performance for both classification and regression. Since this is a regression problem, Precision, Recall, Accuracy and F1 score are useless. As for regression we have seen the mean squared error (MSE) and the mean absolute error (MAE). I would argue that using the MAE is preferable because we do not necessarily want to put more weight on outliers. Using MSE will square the error and that will be heavy. Some houses will be very expensive and they should not play a major role in predicting the house prices.

Why is it important to split the data into training and testing data? What happens if you do not do this?

Testing with the same, or part of the same, dataset that was used for training will introduce bias in the predictions. The model will make predictions on the same data points that it used for training, thus resulting in cheating. If we were to chose a model based on those predictions, the results could be catastrophic. We would have no idea whatsoever what the performances would be on examples that the model never saw before.

Which cross validation technique do you think is most appropriate and why?

At this point in the class we have only seen K-Fold, so I'm gonna go with that. As far as I can tell, all the other CV techniques are variations on K-Fold. The main advantage of K-Fold is that it allows to train on everything and test on everything.

What does grid search do and why might you want to use it?

Hyperparameters can only be learned empirically. It's something that we have to see if it works well or not. Doing it manually is tedious and fortunatly librairies like Scikit Learn makes it super easy to do. It will do all the possible permutations of the specified hyperparameters and run the learning algorithms with them. It will select the best ones and use them for further predictions. It's all nicely wrapped and easy to use. So basically grid search saves a lot of hassels and programming time, but takes more time to train. So if the dataset is too big or you have a learning algorithm that naturally takes a lot of time to train, like neural nets, then you might want to think about it twice or at least do a smart selection first and not just throw a range of hyperparameters to try at random.

Analyzing Model Performance

Look at all learning curve graphs provided. What is the general trend of training and testing error as training size increases?

Training and testing error are quite high with a small depth for the decision tree. The testing error soon stabilizes no matter the depth of the depth of the decision tree whereas the training set keeps getting lower and lower, creating a huge gap between the testing and training error curves.

Look at the learning curves for the decision tree regressor with max depth 1 and 10 (first and last learning curve graphs). When the model is fully trained does it suffer from either high bias/underfitting or high variance/overfitting?

At depth 1 we have a problem of underfitting, high bias. The error is high on both the testing and training set. The model is unable to generalize.

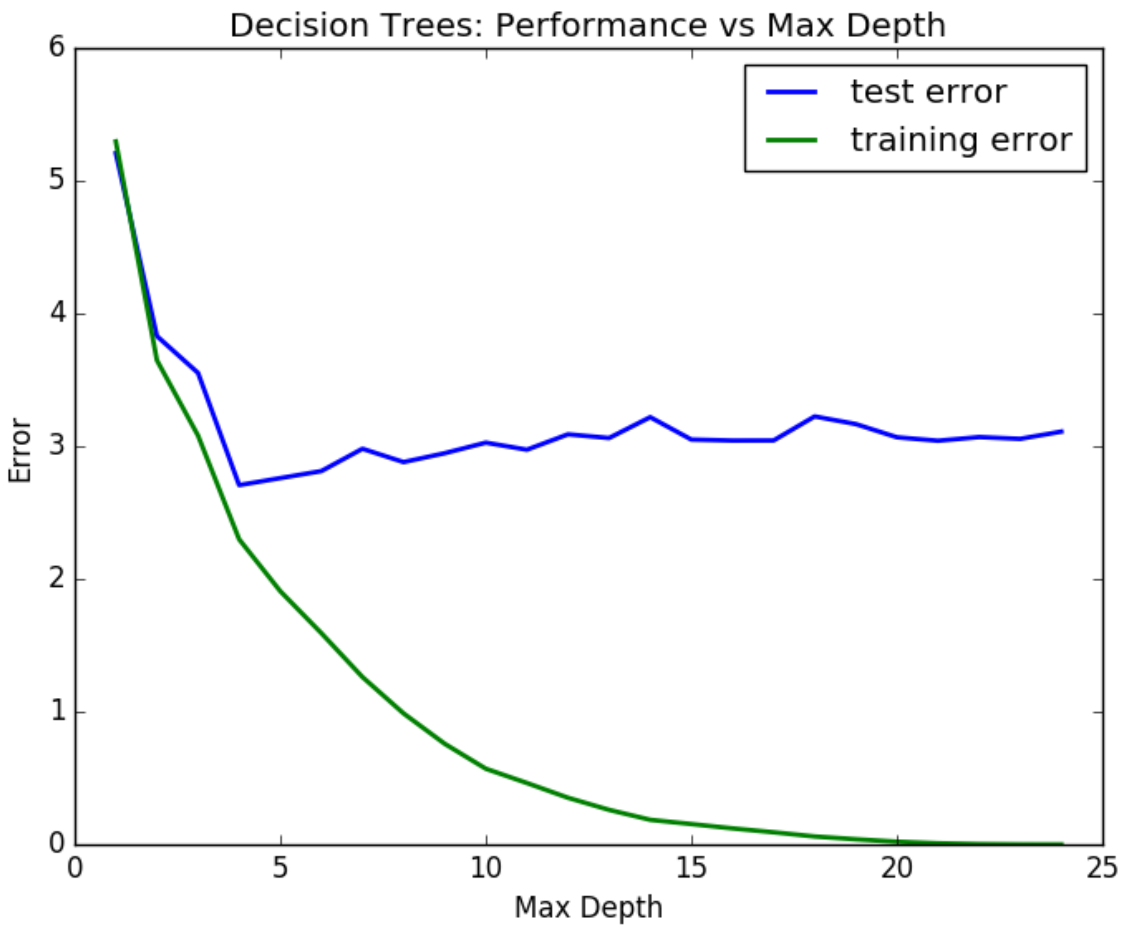
At depth 10 we have a problem of overfitting, high variance. The training error is close to zero but the testing error stays the same, far from the training error. That indicates an inability to generalize well to unseen data.

Look at the learning curves for the decision tree regressor with max depth 1 and 10 (first and last learning curve graphs). When the model is fully trained does it suffer from either high bias/underfitting or high variance/overfitting?

As the depth increases, the variance is getting higher. An overfitting problem starts to appear where the training error is close to zero but the testing error stays the same. That indicates an inability to generalize well to unseen data.

Look at the model complexity graph. How do the training and test error relate to increasing model complexity?

As we saw with the previous graphics, are the complexity of the model increases, the gap between the testing and training error grows wider. Testing error stays approximately the same after a certain point. So there is no need to increase the complexity beyond that point.



Based on this relationship, which model (max depth) best generalizes the dataset and why?

Eyeballing it, a depth of 4 or 5 is good enough for generalization. Beyond that the training error is getting lower, but not the testing error. If increasing the complexity of the model does not allow better generalization, then we should stick to a point where testing and training error are closer. We will save computation time as a side effect.

Model Prediction

Model makes predicted housing price with detailed model parameters

```
GridSearchCV(cv=None, error_score='raise',
       estimator=DecisionTreeRegressor(criterion='mse', max_depth=None,
max_features=None,
           max_leaf_nodes=None, min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, presort=False, random_state=None,
           splitter='best'),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'max_depth': (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)},
       pre_dispatch='2*n_jobs', refit=True,
       scoring=make_scorer(mean_absolute_error, greater_is_better=False),
       verbose=0)

Best params: {'max_depth': 6}

Best estimator: DecisionTreeRegressor(criterion='mse', max_depth=6,
           max_features=None,
           max_leaf_nodes=None, min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, random_state=None,
           splitter='best')

House: [11.95, 0.0, 18.1, 0, 0.659, 5.609, 90.0, 1.385, 24, 680.0, 20.2,
332.09, 12.13]

Prediction: [ 21.62974359]
```

Looking at the best params we have a best depth at 6. It's a bit different than my expectation, but the numbers do tell that it is the best score.

Compare prediction to earlier statistics

The predicted price is 21,629.74$. It is inside one standard deviation from the mean and not too far from the mean and the median.
The interquartile range of the provided examples is 17,025-25,000, so the predicted price is well between the outliers range.

```
Minimum house price                 : 5,000.00
Maximum house price                 : 50,000.00
Mean house price                    : 22,532.81
Median house price                  : 21,200.00
Standard deviation of house prices : 9,188.01
```