**Questions and Report Structure**

**1) Statistical Analysis and Data Exploration**

There are 506 houses contained in the Boston data set. Each house contains 13 features describing the house. I would like to know what house characteristics these features represent but was unable to get a description from the sklearn website. The housing prices range from 5 .0 to 50.0 with an average of 22.53, a median of 21.2, and a standard deviation of 9.19. Trulia gives a graph of the median sales price [here](http://www.trulia.com/real_estate/Boston-Massachusetts/market-trends/) with values over $400k between 2012 – 2015. Thus the median price of 21.2 is 18000 times smaller. The housing price might be $1000 per square foot but 18000 is not a reasonably sized house. Thus I really don’t understand what the price dimension means either unless these homes were in the poorest areas. I would think that 2 of the features would be longitude and latitude. However the (lat,long) for Boston is (42.36,-71.06) and the example features for the house we want to predict are [11.95, 0.00, 18.100, 0, 0.6590, 5.6090, 90.00, 1.385, 24, 680.0, 20.20, 332.09, 12.13] and thus none of these features are close to those numbers so I would conclude that lat and long are not features in the dataset

**2) Evaluating Model Performance**

I chose the mean absolute error as most appropriate error measure as the outcome intuitively tells me how much sale price error I can expect from my model. For this model we get an average absolute error of approximately 2.42. Thus when the model predicts a 20.00 sales price the actual sales price should be between 22.41 and 17.59 on average. The mean squared error could have worked I would just have needed to do the sqrt math to get the same result. Obviously the regression metrics needed to be chosen for this data sample as we are predicting a continuous number and not a finite class. Also I don’t believe the median absolute error is appropriate since I want the measure to be sensitive to outlier errors as I want to be aware when the model is way off and to see that in my error measure.

The housing data needs to be split into training and testing data so that we could measure the generalization of the model used to predict the sales price. If all the data was used for training, we would just get a measure of how well our model fit the training data and not have a measure of how well the model worked on data points it had not seen before. Also if all the data was used for training you could just over fit the data to reduce your error and would not see that your model was over fit and would not generalize well to new data.

In the first part of the boston\_housing program it prints out the learning curves that show how the testing and training errors converge as the amount of training data increases. Then the model complexity plots the end of those learning curves with the maximum training data for various tree depths or complexities of the underlying model. Thus we can manually plot these model complexity curves and pick out the model that produces the lowest average test error with the minimum complexity.

Gridsearch automatically searches over a parameter set of the model and picks the model parameters producing the lowest average test error during cross validation. Thus gridsearch automates the process of the cross validation procedure over several parameters by doing that process in one subroutine call. For instance in my code, I ran gridsearchcv over the parameters of max\_depth 1-10 and max\_features 11-13. Gridsearch then returned the best parameters as being: max\_features 12 and max\_depth = 7. If I didn’t have gridsearchcv I would have had to run cross validation over the model separately for 30 different parameter combinations in order to pick the best combination.

Cross validation is useful since it allows us to use all our training data and all of our test data in building our model. With the K-fold model the test and train data split is cycled through K tests and the results are averaged. Thus with 3 folds there are 3 different splits run where in each split a different 1/3 of the data is used for testing. These 3 splits are run separately and the results are averaged. We use cross validation with gridsearch because cross validation is used to score each of the parameter combinations such that gridsearch can optimize over the grid of parameters it is searching through

**3) Analyzing Model Performance**

Looking at all the learning curves, it is clear that as the number of training points is increased, the training set error increases until it converges at a given error while the test set error decreases until it converges to a given error. As the complexity of the model increases it takes more training data for the test and the training errors to converge. As can be seen from the curves for a max depth of 10, the test error and the training error seem not to be converging as there is an insufficient amount of training data.

At a max depth of 1 the errors for test and training converge to a higher level as the model underfits the data and the model exhibits high bias as the error converges to about 5.2. At a max depth of 10, the data is overfit as we can see the error in the training data is much lower than the test data .44 versus 2.42 respectively. The max depth of 10 model suffers from overfitting or high variance.

Looking at the model complexity graph, the test error converges to its lowest amount of about 2.42 at a max-depth of 6. After the max depth of 6 the error in the test set doesn’t reduce while the training error decreases further as the model is overfitting the training data. Thus the max depth of 6 best generalizes the data as the test data error is minimized and the complexity of the model is at its lowest.

**4) Model Prediction**

I complicated the gridsearch procedure by searching over both max-depth and max-feature. The best model occurred at a max-feature of 12 and a max-depth of 7 nearly matching my estimate from the curve above at a max-depth of 6. Note that for the above estimate the max-features was fixed at 13. It bothered me that the simulation was giving different answers on different runs so I fixed the random state variable in the code to a constant. I changed the random state several times to arrive at the optimized model displayed

The model predicted the housing price to be 20.0. The prediction is pretty close to the mean so I have no reason to think the model has an error.

Is the model useful?

With an error of 2.42, I don’t think the accuracy is good enough . The entire data set had a mean of 22.53 with a standard deviation of 9.19, while my predicted value had a value of 20.0 with a standard error of 2.42. Yes, the model is better than predicting with the mean of the population but I don’t think a realtor should trust this model. The fact that I don’t see a latitude or longitude parameter in the feature list as described above means that the model isn’t considering location. I thought real estate was all about location, location, location