Report:

Project 1, Boston Housing Prices

1) Statistical Analysis and Data Exploration

- Number of houses: 506
- Features per house: 13
- Minimum price: 5.0
- Maximum price: 50.0
- Mean price: 22.5328063241
- Median price: 21.2
- Standard deviation: 9.18801154528

2) Evaluating Model Performance

Measure of model performance :

I settled on median absolute error, **med_ae**, as a metric because it seemed to be providing smoother error curves than mean squared error, **mse**, or mean absolute error, **mae**. This might be do to the median's resistance to outliers. I'm not sure how legitimate this reasoning is, but I ran the following code at the GridSearchCV section;

```
metrics = {'median_ae' : m.median_absolute_error, 'mean_ae': m.mean_absolute_error, 'mse':
           m.mean_squared_error}

 scorer = make_scorer(metrics['median_ae'], greater_is_better=False)
 reg = GridSearchCV(regressor, parameters, scoring=scorer)


for key in metrics:
   found_param = []
   for i in range(200):
       scorer = make_scorer(metrics[key], greater_is_better=False)
       reg = GridSearchCV(regressor, parameters, scoring=scorer)
       reg.fit(X,y)
       found_param.append(reg.best_params_['max_depth'])
   print key +"'s max_depth choice on average: ", np.mean(found_param), " with std: ", np.std(found_param)
```

Basically running grid search using each error metric 200 times and checking on average what was the recommended best max_depth choice. This resulted in these data;

- **med_ae**'s max_depth choice on average: 5.685 with std: 0.696975609329
- **mae**'s max_depth choice on average: 5.105 with std: 1.172166797
- **mse**'s max_depth choice on average: 5.425 with std: 1.73042624807

Which support somewhat the observation that the error curves for mse and mae were more erratic. Using med_ae, GridSearch would find a best max_depth between 5 and 6 more consistently. I guess some of this would only matter if we were trying to automate the selection of max_depth. Other than that all three error metrics were providing similar results and seemingly could be used interchangeably. Accuracy, Reason for train/test data split :

The cross validation train_test split is the first step in constructing all the model diagnostics we perform in the code with the methods learning_curve and model_complexity. Without the test set, we would not be able to gauge whether the model is tending to over-fit the data. A model can be made to fit a certain data set exactly, but this does not guarantee it will generalize well to yet unseen data. We can spot patterns that indicate over fitting in training vs test error plots for our model.

Grid Search :

Grid search provides a convenient method for tweaking the parameters of the model (in this case, just max_depth, for tree regression) in a more automatic way. Instead of;

- **i.** Manually choosing different values for max depth
- **ii.** Then running fit with each
- **iii.** Then testing them on the test set to find which yields the smallest training/test error combination

GridSearchCV by default splits the data 3-ways, I believe randomizes its order, and then systematically trains each parameter (only a list of values for max_depth in our case) on 2/3 of the split and tests it on the remaining 1/3. Then repeats with different 2/3, 1/3 parts of the data. It then picks the parameter that yielded the smallest cross-validation error on average out of all trials.

Cross Validation :

Cross validation is important because say we train different versions of a model, i.e. using different parameters, on a training set, then test them on the test set. If we then choose the model that had the smallest error, in a sense we fit it specifically to the test set. We have lost a way to measure how well the model will generalize to new data. By having an additional cross-validation set, we can tweak the model by choosing parameters to minimize error on it, AND still have a separate test set with which to measure future performance. Grid search does a version of this by alternating between 2/3 and 1/3 splits in the data (as I understood it) and using ALL the data for training, cross-validation, and testing.

3) Analyzing Model Performance

Model Complexity Graph:

A max_depth between 5-6 is what we want, since it agrees with the behavior of the Model_complexity_graph. Here, a clear divergence between the test and training error curves can be seen between max_depths of 5 and 6. At that point the training error curve continues its descent towards zero (as it over-fits) while the test error curve levels off horizontally. So that after 5-6 max_depth the model would start to approach a perfect fit to the training data (training error), while not predicting unseen data with any better accuracy (test error).

Learning Curve Graphs:

Similar behavior can be gleaned from the learning curve graphs. Basically for all these curves, training error starts at zero and test error starts at a max. This is explained because few data points can always be fit by a model, while the chances of that same model predicting an unseen example are low (heavily biased). As the amount of data is incremented the curves approach each other since it is harder to fit all examples perfectly (training set), but bias is being reduced (test set). For low values of max_depth parameter (i.e. the first few learning curve graphs displayed) this convergence levels off at a high error. This indicates that the model is biased overall. Neither the training set nor the test set are being fit particularly well. As max_depth increases the horizontal where the two curves are converging, moves towards the x-axis. This is happening because the more complex model is capable of fitting the training set better AND generalizing to the test set better. After a max_depth of 5-6 however, only the training error horizontal continues this trend as it starts to over fit the data while no better prediction is occurring with the test data.

4) Model Prediction

Final model choice: max_depth = 5
I settled on 5 first from visual inspection of the learning curve graphs and complexity graph, but also from the average value GridSearchCV was spitting out as mentioned above in part 2 of the report.
For the given data point;

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

this model predicted 20.96776316. This number is well within one standard deviation of the mean for all data calculated in part 1. Under that criteria at least, the prediction seems reasonable.