**Meets Specifications**

**Quality of Code**

**Specification**

All coding sections run fine without error. Students do not alter the starting code beyond what is requested.

Meets Specification

**Data Exploration**

**Specification**

All requested information about the Boston housing data is accurately supplied to within a tolerance of +/-0.05. Students use the numpy library in the provided template code to obtain results.

Meets Specification

**Reviewer Comments**

Well done correctly updating the section.

**Evaluating Model Performance**

**Specification**

Student uses an appropriate error scoring metric from the list provided and provides ample reasons for doing so.

Meets Specification

**Specification**

Student provides a valid reason why data is split into training and testing data. Student implements this split in code.

Meets Specification

**Reviewer Comments**

Thanks for using the proper train test split approach.

**Specification**

Student explains the importance of cross validation and why it is useful for grid search to use. If a student modifies grid search beyond the default 3-fold cross validation a reasonable justification is provided.

Meets Specification

**Specification**

Student properly implements grid search and justifies why.

Meets Specification

**Analyzing Model Performance**

**Specification**

Student correctly identifies the relationship between the training and test error as training size increases.

Meets Specification

**Specification**

Student provides analysis for both max depth 1 and 10 learning curve graphs. Both graphs have ample explanation if they suffer either high bias/underfitting or high variance/overfitting and are reasonably justified.

Meets Specification

**Specification**

Student identifies how the training and error curves relate to the increasing model complexity.

Meets Specification

**Specification**

Student clearly picks an optimal model from the model complexity graph with reasonable justification.

Meets Specification

**Model Prediction**

**Specification**

Student’s model gives a valid housing price with detailed model parameters (max depth) reported.

Meets Specification

**Reviewer Comments**

**Pro tip:** You can set the max depth to the value you have found being most relevant:

regressor.max\_depth = 5 # setting max depth to a value you prefer

# Fit the learner to the training data

regressor.fit( X , y )

prediction = regressor.predict( x )

print "Price Prediction: " + str( prediction )

**Specification**

Student compares prediction price to earlier statistics and justifies why the price is reasonable.

Meets Specification

**Reviewer Comments**

**Pro tip:** To assess if your prediction is reasonable, besides from comparing it with the median, the mean and checking if it is included in one standard deviation range, you could use SKlearn to find the nearest neighbours of the feature vector and see how your result compares with them.

from sklearn.neighbors import NearestNeighbors

def find\_nearest\_neighbor\_indexes(x, X): # x is your vector and X is the data set.

neigh = NearestNeighbors( n\_neighbors = 10 )

neigh.fit( X)

distance, indexes = neigh.kneighbors( x )

return indexes

indexes = find\_nearest\_neighbor\_indexes(x, X)

sum\_prices = []

for i in indexes:

sum\_prices.append(city\_data.target[i])

neighbor\_avg = np.mean(sum\_prices)

print "Nearest Neighbors average: " +str(neighbor\_avg)

<http://scikit-learn.org/stable/modules/neighbors.html#finding-the-nearest-neighbors>

**Additional Reviewer Comments**

Dear student,

I'm quite impressed by the scale of your improvements you made it for a very good submission. I’ve left some (new and old) pro tips in case you might be interested in learning more about some specific topics.

Congratulations on passing you exam!