CS 574 Final Project Report
Ian E. Gonzalez-Alfonzo

1. Background of the project:

I was trying to predict the price of a house in Boston. People trying to buy a house, especially people who have never bought a house before, can often feel at a loss on how to set a price range for their dream house. This provides a grounded and scientific approach on how to find a price for a house (specifically in Boston) based on historical data.

2. Source of the data:

The dataset was a Boston Housing Dataset, sourced from Kaggle:
https://www.kaggle.com/altavish/boston-housing-dataset

3. Steps of operation on the data:

1. Import the Seaborn (as sns) and Pandas (as pd) libraries.
2. Download the dataset from the link above and import it into the Python program using the local filepath.
3. Load the CSV into a Pandas dataframe using pd.read_csv() (at this point, the dataset is named boston_data_unclean, as the data has not been cleaned yet).
4. Used a heatmap from Seaborn to visualize the nulls in the data. Also used the describe() function to get a rough idea of the values of the data.
5. Used the boston_data_unclean.isna().any() function to verify the results from the heatmap. This function lists by column whether there are null values, returning true for having nulls and false if there are no nulls.
6. Reassigned the dataset to boston_data after dropping all nulls using the dropna() function. The parameter of axis for dropna() was set equal to zero because that tells the program to drop all rows that have empty values.
7. Verified that all the null values were dropped by running boston_data.isna().any() and making a new heatmap on the boston_data dataframe.
8. From a survey I conducted towards potential new owners, I determined the most relevant features to include in the model were the per capita crime rate by town, the weighted distance to five Boston employment centers, the average number of rooms per dwelling, the proportion of owner-occupied units built prior to 1940, and the percentage lower status of the population. This was the X or independent variables. The y or dependent variable was the median value of owner-occupied homes in US dollars by the 1000's.
9. After importing the decision tree regressor and test-train split from sklearn, a boston_model was created and the dataset was split into train and test data. The model was then fit to the train data.
10. The predict() function of the boston_model was used to obtain a prediction based on the model. The testing data was passed as a parameter to the predict function.
11. After the predictions were obtained, two error metrics were obtained: an R^2 and a mean average error. These will be discussed in the results section.
12. An optimization method was applied to the decision tree: since different amounts of tree nodes would yield different error values, the decision tree with the best of a set of different node amounts was chosen and set as final_decision_tree_model. "Best" in this context was defined as the model whose number of nodes yielded the best mean absolute error.

13. Since Random Forests are usually better models than decision trees, I also created a Random Forest model on the train data after importing RandomForestRegressor from sklearn.ensemble.
14. For this model, predictions were obtained when applying it to the test data, in the same fashion as in the Decision Tree model. R^2 and mean absolute error values were obtained for this model as well.

4. Results:

MAE, Decision Tree, no optimization: 3.513
MAE, Decision Tree, optimized (to 25 nodes): 3.513
R^2, Decision Tree, optimized (to 25 nodes): 0.631
MAE, Random Forest: 2.834
R^2, Random Forest: 0.814

5. Findings in the project:

These models are very accurate. Since R^2 is a value which as closer it is to 1, the more accurate it is, one can see that especially the Random Forest model is very accurate. Optimizing did not improve MAE because the default value for the nodes of the Decision Tree was already at the optimized amount of 25 nodes. However, when this is not the case, optimization still is favorable.

Although I initially tried to use all the features in the dataset to create my model, using a more limited range of features provided better results. Also, switching from a Decision Tree model to a Random Forest model also helped. Also, I was experiencing overfitting from using the same data to train as to test. I resolved this by splitting the data into test data and training data. One more problem I faced was how to select features that gave low error. I solved this by interviewing a set of potential future house buyers and getting their feedback on preferred features. This resulted in accurate models.

6. Location of the code:

All code, including my practice code, and all datasets can be found at
https://github.com/Astroworld97/CS574FinalProject

7. Acknowledgement:

No outside help was used except for the tutorial mentioned in section 8. This tutorial was written by Dr. Dan Becker.

8. References:

All procedures were based off of the “Intro to Machine Learning” course from Kaggle, found at kaggle.com/learn/intro-to-machine-learning.