Ari Burian
Project 4 Executive Summary

When I first looked at the data, I noticed that there were 7,468 rows in total, and 2,542 of them were missing the price of the house. Since our goal is to categorize each row as either “high price” or “low price”, I decided to drop entirely the rows without a price, because without a price we wouldn’t be able to categorize the prices.

The first column in our data is a string, which includes the street address, city, and state. I wrote a function to capture the house number, street direction (N, E, S, or W) and street name separately in their own columns, and disregarded the city and state because they are all Chicago, IL. Having the house number in numeric form I can now use it in my models. At this time, I haven’t incorporated the street direction or street name, but with additional time I would like to turn the four cardinal directions into categorical variables, and use that it my modeling. While cleaning this column, there were several instances that the information didn’t follow the expected pattern, in which case I replace the incorrect information with a house number of zero.

The second column includes the number of bedrooms, bathroom, and the total square feet. I again wrote a function to split these 3 values and return them as integers, and cleaned up any non-uniform entries.

I also turned the price column into integers in place of strings. As a final check, in case any of my integer columns included null values, I replace any null entry with the mean of that column, so that they shouldn’t skew the data.

Finally, I created dummy variables for all the different possible entries in the “status” column.

Now that all of my given information is clean and ready to use, I added one final column to help decide my target of ‘high price’ or ‘low price’. This column which I added is the price per square foot, which I obtained by simply dividing the list price by the number of square feet of the property. From my very first step, I know that I have a price for each row, and, as I just mentioned above, I filled in any null values in square feet with the mean of all the property, so the price per square foot will be reasonable in every instance, even if not exact.

In order to draw a line between high and low price, I plotted a histogram of the price per square foot of each property. The histogram was clearly skewed to the right, so I chose to make the cut-off the median and not the mean, so really high price per square foot properties wouldn’t have a strong effect on my cut-off line. The median is $194/sq. foot. Anything more expensive than that is classified as ‘high price’, anything below that is ‘low price’. I added my target column of ‘is_high’, coded 1 for high price and 0 for low price.

I used the 2 models we learned in class to attempt to best predict the “is_high” target. First, I made a kNearestNeighbors model. I used as my predictors many of my numeric columns: zip code, number of bedrooms, number of bathrooms, the square feet, and the dummy variables of the status. I took out price because that directly impacts what is considered ‘high’ or ‘low’, but I left in square feet because it isn’t intrinsic when setting the price to feed in the square feet. I also took out the house number, because with it I was getting higher training scores, but lower test scores, implying that it was overfitting to my data.

I did a test-test-split, and then experimented with different k-values. Here is a table of approximate scores I got with different k-values:

| K-value | Train Score | Test Score |
|---|---|---|
| **1** | 0.988 | 0.813 |
| **2** | 0.872 | 0.751 |
| **3** | 0.858 | 0.749 |
| **5** | 0.821 | 0.763 |

With a k-value of 1 I constantly got the highest training scores, and although I initially thought that was overfit, it consistently gave the highest test scores of any k-value. This ends up being my best model, with an approximate Training Score of 0.988, and a Test Score of 0.813.

The second model I made was a logistic regression. I made a Seaborn heat map to look at correlations, and then I did a grid search to find the best model type of logistic regression. I first did a test-train-split, and then did a grid search on my training data. The best fit I found was an L1 (Lasso) and a C-value of 100. I made sure to check numbers larger and smaller than 100, to be sure that 100 was indeed the best. I fit my training data to my lasso linear regression, and got an r-score of 0.738 on my training data, and 0.746 on my test data. So my model is not overfit and very consistent between data sets, but yet still does not perform as well as my kNearestNeighbors model. If I was more concerned about consistency over overall performance, I might use this model instead of the kNearestNeighbors model.