**Final Project: House Price Predictor Using Machine Learning**

By Kokil Dhakal and Dan Hoogasian

1. Description of Problem

Our project was based on the house price predictor dataset. The goal was to be able to predict the price of houses based on their attributes, using machine learning. This service would be used by people looking to know the price of a house for a variety of reasons.

Real estate business has a great impact on the national economy. That is why predicting house prices has great significance for buyers, sellers, real estate agents and other related economic professionals. A good prediction of house price is also important for builders, investors, and tax assessors. We used machine learning techniques which are widely used for the price prediction due to its remarkable advantages as compared to traditional methods.

1. Description of Data and Pre-processing

The dataset used has 2919 instances of 81 columns. One column contains the ID numbers, and another has the sale prices. The other 79 columns contain attributes. The attributes are both qualitative and quantitative, and are things such as year built, number of half-baths, kitchen quality, and type of lot. The data included a file briefly describing the attributes and what the possible values of each attribute could be. The houses were in the small city of Ames, IA.

The data we received was raw data, which is not ideal for analysis, so it required significant cleaning. Some attributes like length of house street, type of alley, fireplace quality, pool quality, fence quality, and miscellaneous features had large portions of the data missing. For convenience, we dropped these attributes before analysis since they also had little determination on the sale price. After dropping these columns, there are still columns that have NaN values which needed to be fixed before doing analysis. For convenience, we dropped all NaN values instead of imputing null values. This left us with a 7 by 1,438 cleaned data set ready for processing.

The data needed a good amount of preprocessing before it could be analyzed. Some of the attributes weren’t applicable to a large portion of houses, so they had zero or no values. The qualitative attributes were converted to a numerical representation for the analysis. Outliers were determined as being more than three standard deviations from the mean of each attribute and were removed for analysis. This included removing the 19 houses that sold for more than $426,000 and one that sold for under $35,000.

1. Basic Methods and Techniques

Many of the data attributes didn’t have a significant determination on the sale price, so we used a linear regression model function from the scipy module to determine the correlation each attribute had on the sale price. In this model, it is called the R-value or the Pearson correlation coefficient. There are many ways to calculate the correlation between variables, but for we used one of the most standard techniques here, for our first extension idea. It has a range of -1 to 1, with 0 being no correlation, 1 being a perfectly positive correlation, and -1 being a perfectly negative correlation. We set the threshold for significance at -.6 and .6. This gave us the following six attributes to use for our machine learning program: overall quality, exterior quality, above ground living area square footage, kitchen quality, size of garage in car capacity, and garage square footage. Some of these features might not be significant in most parts of the county, but likely due to themed-west location, these features were more important.

We used the matplotlib module to get some basic plots of the data. Histograms are a good way to visualize continue data, which our predicted values were. When looking at a histogram of the R-values, it holds a mostly normal distribution, skewed slightly left with more values being slightly more positive than negative. Scatter plots are good to visualize the data when you initially look at it. The scatter plots for each attribute compared with the sale price are all sloping upward, as we would expect.

The sklearn module was used for the machine learning. We used train\_test\_split from the model\_selection function package to divide our data into two-folded sets. It is important to split the data into appropriately sized test and train sets because you want to make sure you have sufficient data for both parts. It is a bad idea to test on the same data you used for training, which is called data leakage. This causes your model to be unrealistically accurate. We looked at the accuracy scores of this and it proved to be immensely true.

Many of our attributes had values that were of different types, which meant their nominal values were not comparable. We used some statistical functions from sklearn to calculate a scaled score of the attributes, which normalized the values to comparable numbers.

1. Extensions, and Remaining Methods and Techniques

The first algorithm we used for predictions was the linear regression algorithm. The accuracy of our algorithm on the testing data was 80.2% with a mean squared error of 884,000,000. We used a graph from the seaborn module to visualize the predicted versus actual outcomes, which shows its accuracy and lack of outliers.

Second, we used the gradient boosting algorithm from the ensemble package of sklearn. The accuracy was slightly better that the linear regression’s, usually at 80.9% with a mean squared area usually between 855,000,000 and 860,000,000. We used the same plot as we did with the linear regression model to visualize the data, but this time the data looked more accurate in the lower range of values than the higher range. It also held a curved line of best fit, as we would expect.

Lastly, we used the Random Forest Regressor algorithm. Since our data was discrete and we were using supervised ML, we used the regressor version, instead of the version for unsupervised ML. The accuracy was around 80.8% with a mean square error of 858,000,000. Both of these last two models had versions for clustering with unsupervised learned, so we chose the regression versions.

We used some bar plots from the seaborn module in conjunction with the standard matplotlib module to visually compare the mean squared errors and accuracies. Linear regression models typically work better with less complicated or less detailed datasets, so we expected this model to do worse than it did, compared to the other two. This was likely due to our small selection of attributes. Though, it did have a slightly lower accuracy.

Researching the different algorithms for this extension idea proved how many different algorithms there were. We were also considering implementing KNN and neural network algorithms, as they also looked applicable to tour data, but we chose only these three as they seemed to be the best fit. It was interesting to see the varieties of all the algorithms out there.

Although we might have had better luck with different algorithms, what we used appeared fairly accurate. Although it was surprising to see such similarity in results from all three.

1. Conclusion and Concepts Learned

One thing we learned is that there is much work needed to preprocess the data. For example, converting the data from categorical to quantitative data is complicated and time consuming. Randomly assigning numbers to categorical elements that have order is time intensive, so we couldn’t do this. If we did this, we would have had a better selection of attributes and our models would have been more accurate. We also learned dealing with missing values is time consuming, but also essential.

When looking at the wide array of attributes the data has, it is hard to see. Some more exploration into other algorithms would have been….