ECE 470 Project Report

House Pricing Model

Cobey Hollier V00893715

Kutay Cinar V00886983

Chris Dunn V00897180

Emmanuel Ayodele V00849004

July 21, 2021

GitHub Repo:
https://github.com/CobeyH/ECE470

Contents

List of Figures

1 Problem Description and Motivation

In this project a set of AIs will be developed that can estimate the price of a house given a set of attributes. Using sample data from Kaggle, machine learning models will be trained and verified using methods discussed below [1]. This will provide an unbiased program that can give consumers an approximation of how much a house is worth without relying entirely on the honesty of the Realtor. It can also be used by a land owner who is looking to sell their house to determine which improvements can easily be made to increase their house value.

Problem Statement: Housing prices are difficult to evaluate, leading to demand for a tool that is able to accurately predict housing prices.

2 Related Work

Park and Bae, 2015 [2] investigated the effectiveness of several machine learning models in Real Estate valuation. They tested C4.5, RIPPER, Naïve Bayes, and AdaBoost algorithms and concluded that RIPPER (Repeated Incremental Pruning to Produce Error Reduction) consistently produced the best result. (Dimopoulos and Bakas, 2019 [3]) analyzed several additional machine learning models to appraise Real Estate in Cyprus. They performed sensitivity analysis on their features to determine which ones most impacted sales price and tried to increase the transparency of their machine learning networks. (Oladunni and Sharma, 2016 [4]) evaluated the hedonic housing theory, to discover its value in Real Estate price prediction. They tested SVM, KNN and PCR methods which yielded results with 16.6%, 15% and 11.4% error respectively.

3 Problem Formulation

The problem search space for creating a house price model has many different approaches, from from Naïve Bayes to SVMs. There isn't an established best approach to house price prediction using machine learning due to it being a new and emerging technique. In manual appraisals, Realtors evaluate a house based on various features of the house. Most machine learning papers follow this principle, evaluating homes based on their features. The Ames dataset used for this project also follows that principle, with 80 features for each house. This project will focus only on modeling property values in Ames, Illinois. As a result, this project will not be useful for cities with different housing markets than Ames.

4 Methodology

The proposed algorithms were trained and evaluated against the data set provided at Kaggle [1]. Kaggle provides two data sets, one with the input parameters for each house, and the second is a sample output price for each house ID. These two data sets will be joined together based on the house ID and separated into a training dataset and a validation set.

Once the algorithm is trained using the training dataset, it will be tested against the values reserved for validation. The expected results will be hidden from the program and the algorithm will be used to predict the value of each house.

In this project, several different models will be used and evaluated based on their "score". The score will be determined using the root mean squared error between the model predictions and the real values of the houses. The model with the best overall score will be chosen as the winner and will be presented to be used for real house predictions. If the final model has a score above 90%, then the project will be considered successful.

For this project four different models were selected, implemented and tuned to determine which would give the best results. These models include a decision tree, a random forest, a gradient boosted decision tree and a neural network. Each of these approaches has their own advantages and disadvantages that made them potential candidates. These trade-offs will be discussed in the following sections.

4.1 Decision Trees

Decision trees provide a quick-to-train model that is easy to implement. This made it an ideal first candidate to become familiar with the training data and to generate some initial predictions. Decision trees are also tolerant of N/A values [5] which proved to be valuable with the data provided by Kaggle. However, decision trees tend to be prone to overfitting, especially since there are only 3000 samples in the data set. To remedy this, the maximum depth was gradually lowered to prevent the samples from being replicated exactly in the decision tree.

4.1.1 Feature Engineering

After some initial testing, feature engineering was attempted to improve the results of the raw tree. To do this, a Heat Map was created to visualize the correlation between each of the features in the input data. Most importantly the Heat Map showed the correlation between the sale price of the houses and each feature.

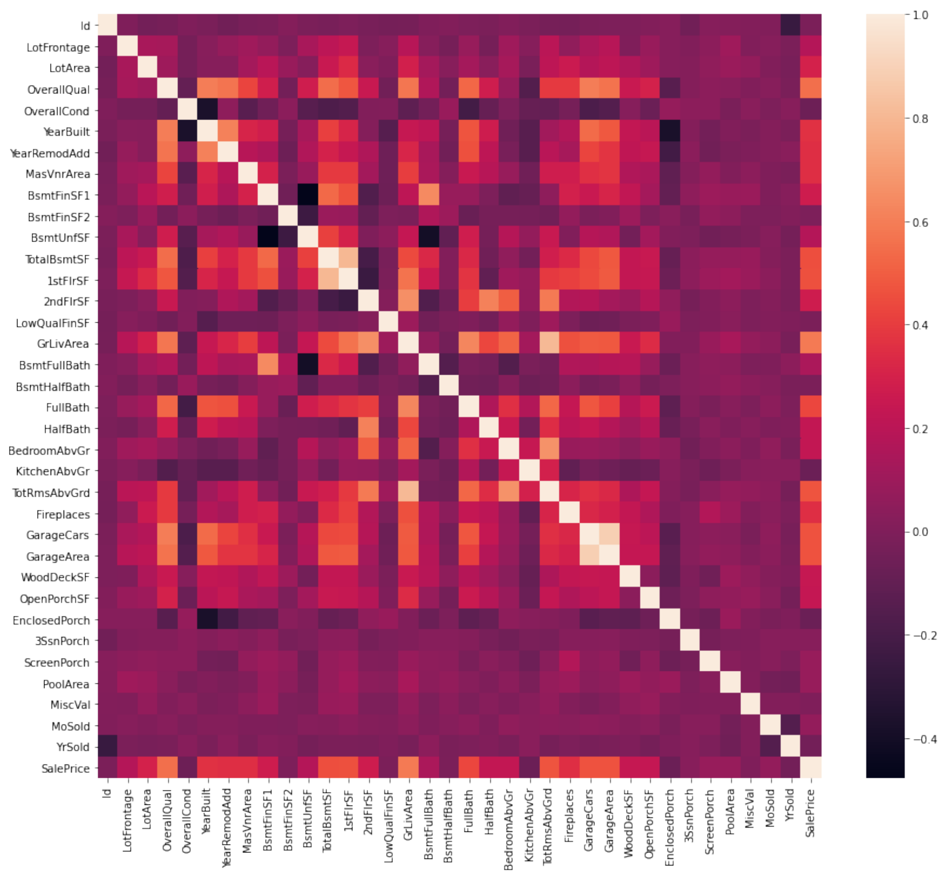

Figure 1: Feature Correlation Map

It was theorized that using only the features that are most correlated with the sale price would reduce the noise in the data and produce better results. The outcome of this experimentation will be discussed in the results section.

4.2 Random Forest

A random forest is a type of "Ensemble technique" [5] that pools the results of many decision trees to yield a more accurate result. Random forests add increased randomness as the trees grow which increases the diversity in this tree. This diversity normally produces a more accurate and more stable result [6].

4.3 Gradient Boosted Trees

Gradient Boosting is another method based on decision trees. It is an ensemble technique like random forests, but it differs both in implementation and final results. Random forests combine the results of each of the trees at the end of the process while gradient boosting combines results along the way [7]. Furthermore, while gradient boosting can out-perform random forest, it is more difficult to tune. All three of the decision-tree based models were implemented using *sci-kit learn* in Python3.

4.4 Neural Network

The final model created for house predictions was a classic neural network. For this, a model was created using *Keras* and tuned through experimentation. Unlike the other methods, the neural network required a larger degree of hyper-parameter tuning and experimentation to determine the best network topology.

To begin, the data was dummified to eliminate the need for string values in the pandas DataFrame. This essentially broke each of the enumerable columns into many new columns with each enumerable value getting its own value. For example, a column that contained values ”Blue” and ”Red” would be broken into one column for ”blue” that contained a 0 or 1 to denote if the house was blue and a column ”red” that denoted with a 0 or 1 if the house was red. This was required since the TensorFlow DataFrame didn't allow string values.

Through the process of dummification, the initial 81 columns were transformed into 305 simpler columns. These 305 new features were then each given a neuron in the input layer of the neural network. This network was then given a second layer that contained a single neuron for the output.

After the basic network was created, trained and validated further modifications were made to the network topology to attempt to improve the performance.

4.4.1 Hidden Layers

The first attempt to improve the performance involved adding a hidden layer of neurons between the inputs and outputs. The number of neurons in this hidden layer was initially set randomly and then tweaked through a series of experiments. Networks that contained 1, 2, 100, 150, 250, 305 and 610 hidden neurons were tested. The results of these networks will be discussed in Section 5. Experiments were also performed using multiple hidden layers in addition to the input and output layers.

4.4.2 Activation Functions

In addition to fine-tuning the network topology, different activation functions were tested for each of the layers. These experiments included tests with the sigmoid, softmax and ReLU functions.

5 Results and Discussion

Due to the numerous combination of models and hyper-parameters tested in this project there were a lot of results to synthesize. For this section all results will be discussed in terms of Root Mean Squared Error. This measurement was used because it was easier to read and discuss compared to the Mean Squared Error due to house prices being large numbers. The other measurement discussed in this section is the score. The score shows the percent accuracy of a model. For example a score of 90% in a set of data with a mean house price of $100,000 would have a Root Mean Squared Error of $10,000.

5.1 Decision Tree Models

5.1.1 Decision Trees

Our first results were produced by a single decision tree. Through some experimentation, different max depths were tested to prevent overfitting. The best decision tree results produced a training error of \$8,500 and a validation error of \$24,825 with a max tree depth of 10. This corresponds to a score of 80.1% for the validation data.

Figure 2 shows the score for each of the max depth values between 1 and 15. Max depth 10 consistently produced the best result.

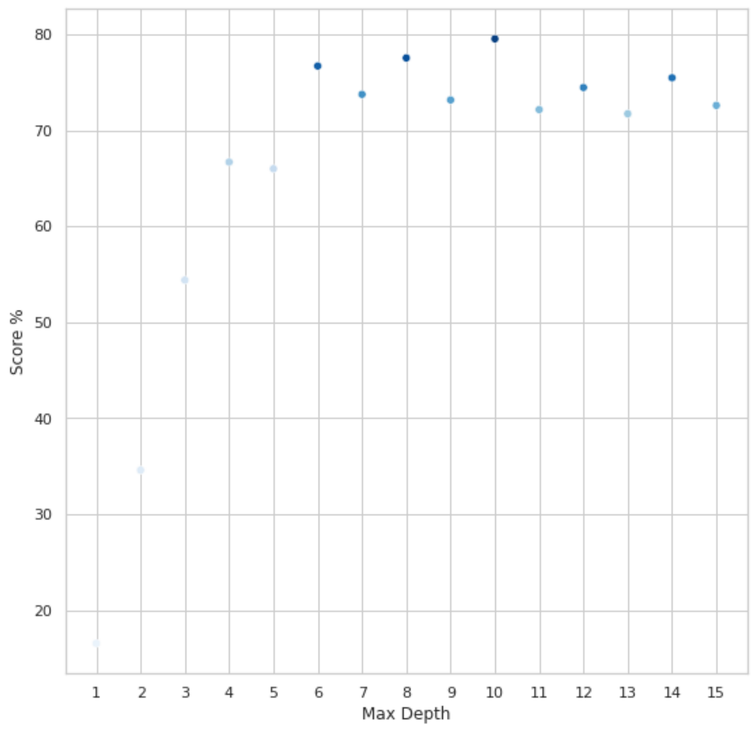

Figure 2: Decision Tree Max Depths

Figure 3 is a graphic that shows a visual representation of the first 20 sample predictions and real values in the data. While this may not be representative of the overall data set it gives some insight into the types of predictions created by the decision tree.

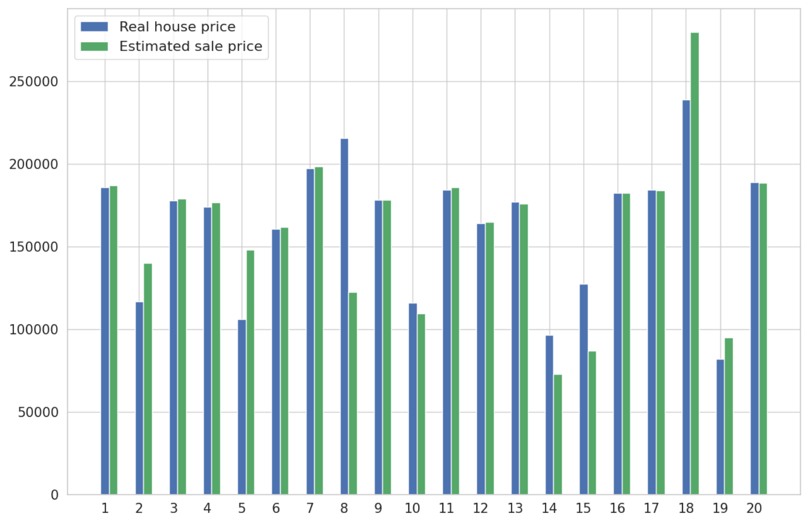

Figure 3: First 20 Sample predictions for decision tree

5.1.2 Random Forest

The random forests proved to substantially improve the results. With 5 estimators and a max depth of 15 a training error of $7877 and a validation error of $19,743 was produced. This corresponds to a score of 87.4% which is a 7% increase over a single decision tree. Through testing, it was found that 500 estimators would produce the best practical result, any more estimators only made fractional improvements. With 500 estimators, the random forest had a training error of $8079 and a validation error of $19311 which equates to a score of 90.1%, a 10% improvement over the decision tree and 3% improvement over a random forest with 5 trees.

5.1.3 Gradient Boosting

The gradient boosting created results very similar to that of the random forest. The best results were produced with 1000 estimators, a learning rate of 0.015 and a max depth of 7. This resulted in a training error of $3439 and a validation error of $15963 which corresponds to a score of 91.2%. Given this proper tuning, it outperformed the random forest by 1%.

5.2 Neural Network

The neural network was the most difficult model to use given the extensive training time demands and the many hyper-parameters to tune. Through some experimentation it was found that a plateau was often reached around 2000 epochs with reasonable results being returned in 200 epochs.

Due to these results, the network typography permutations were tested at 200 epochs to show if they had potential. If that permutation showed promising results then the network would be run again with 2000 epochs to see if it could beat the current best result.

The initial attempt only included an input layer with 305 neurons and a one neuron output layer. This network produced a RMSE of $40005 which is substantially worse than the other models. This was improved by adding a hidden layer between the input and output layers. This hidden layer was initially tested with 100 neurons but through experimentation it was found that the best results were produced with 305 neurons in the hidden layer and the use of the ReLU activation function. This combination yielded an error of $25,464, reducing the error of the previous attempt by almost $15,000.

Our final neural network contained 5 layers total. The first and last layers were still a 305 input layers and a 1 neuron output layer respectively. However, there were three hidden layers instead of just one. These three layers had 305, 100, and 10 neurons respectively, greatly increasing the training time but also improving our final result. This final neural network produced an error of $21,039, with a score of 85.3%, which unfortunately fell short of the results produced by the gradient boosting decision trees. This is likely due to the limited number of samples in the training data which may cause some form of overfitting in the network. However, the experience was still educational and was worth attempting. Figure 4 shows a visual representation of the final network.

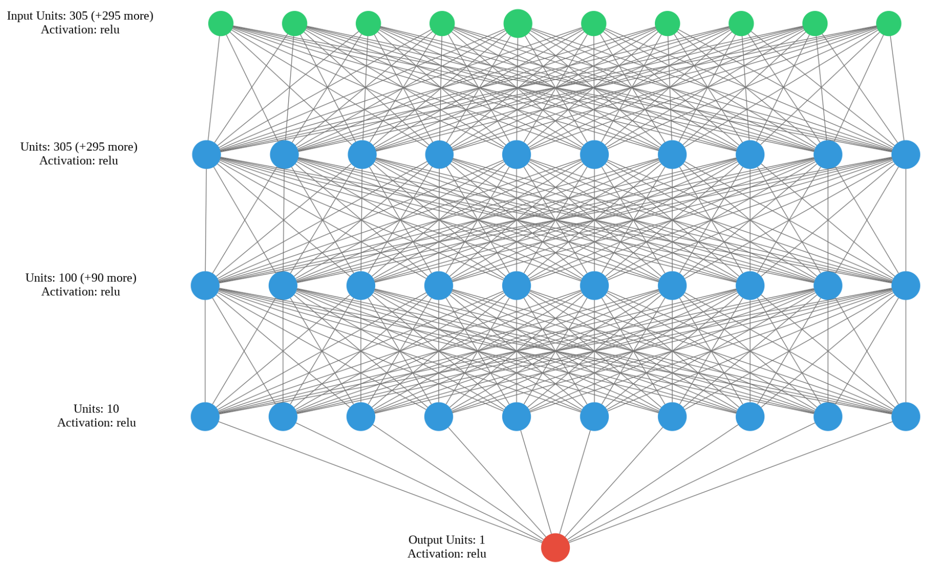

Figure 4: Network Topology

6 Future Work

While this project produced good results, there is still room for improvement. Due to our limited amount of domain specific knowledge it is possible that each of the models could be tuned further to improve the overall results. This is especially true with the neural network since most of the experimentation was done based on researched guidelines and random experimentation. It's possible that a neural network expert could get better results with the same set of data.

Additionally, it's possible that a more complicated network type could produce a better result.

Aside from improving the models, this method could be expanded to use data from other cities or more data from the same city. A bigger set of training data would likely improve many of the algorithms and using data from other cities would make our project more useful as a product. Eventually it would be ideal to have the user input which city they lived in as a feature which would be considered by the models to predict the price. However, this would require orders of magnitude more data which made it infeasible in the time allotted for this project.

7 Conclusion

Over the course of this project, 4 different models were created and tuned to predict the prices of houses in Ames, Illinois. These models included a decision tree, a random forest, gradient boosted decision trees and a neural network. Of these models, the gradient boosted decision tree produced the best result, followed by the random forest, the neural network and lastly the normal decision tree. In the future this project could be expanded by further tuning each of the models and acquiring a more complete dataset. This would allow the project to be expanded to allow users to enter the location they lived instead of only being accurate in a single city.

8 Bibliography

References

[1] Kaggle. *House Prices - Advanced Regression Techniques.* URL: `https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data`. (accessed: 05.15.2021).

[2] Park and Bae. *Using machine learning algorithms for housing price prediction: The case of Fairfax County, Virginia housing data.* URL: `https://www-sciencedirect-com.ezproxy.library.uvic.ca/science/article/pii/S0957417414007325`. (accessed: 05.16.2021).

[3] Dimopoulos and Bakas. *Sensitivity Analysis of Machine Learning Models for the Mass Appraisal of Real Estate. Case Study of Residential Units in Nicosia, Cyprus.* URL: `https://www-mdpi-com.ezproxy.library.uvic.ca/2072-4292/11/24/3047/htm`. (accessed: 05.16.2021).

[4] Oladunni and Sharma. *Hedonic Housing Theory – A Machine Learning Investigation.* URL: `https://www.researchgate.net/publication/311768113_Hedonic_Housing_Theory_-_A_Machine_Learning_Investigation`. (accessed: 05.16.2021).

[5] Stacey Ronaghan. *Machine Learning: Trying to predict a numerical value.* URL: `https://srnghn.medium.com/machine-learning-trying-to-predict-a-numerical-value-8aafb9ad4d36`. (accessed: 07.17.2021).

[6] Built-In. *A complete guide to the random forest algorithm.* URL: `https://builtin.com/data-science/random-forest-algorithm#how`. (accessed: 07.17.2021).

[7] Stephanie Glen. *Decision Tree vs Random Forest vs Gradient Boosting Machines: Explained Simply.* URL: `https://www.datasciencecentral.com/profiles/blogs/decision-tree-vs-random-forest-vs-boosted-trees-explained`. (accessed: 07.17.2021).