Prediction of Housing Prices

Syeda Afrah Shamail

February 2020

Contents

1 Introduction and motivation

Over the past six years the US housing prices have been steeply increasing with nearly 3-5% increase every year. There are a lot of factors that affect housing prices such as location, school district, time of the year the house is purchased, size of the house, age of the house etc. At present, in St. Louis there is a high demand and low supply for houses priced below $500,000 and low demand and high supply of houses priced above $500,000. As a prospective house owner I am curious to know what factors affect housing prices.

2 Data Description

To explore further the relationship between housing prices and other factors I have chosen the USA_Housing dataset from kaggle. This dataset can be found at: https://www.kaggle.com/vedavyasv/usa-housing.

The owner of this dataset did not leave any description about how and when the data was collected and from which state in US. But the columns of the dataset are self explanatory.

The dataset has 5000 rows and the following 7 columns:

- Avg. Avg. Area Income
- Avg. Area House Age
- Avg. Area Number of Rooms
- Avg. Area Number of Bedrooms
- Area Population
- Price
- Address

All the columns except the address column are numeric float type.

3 Data Cleaning

Fortunately the dataset does not have any missing values so we can use all 5000 rows. But, Our last column address has varying values and would not be helpful in predicting the house prices. For this reason I have decided to remove this last column from the dataset. Also, I decided to change the columns names so they follow convention of decapitalized words separated by an underscore(_). The new column names are:

- avg_area_income
- avg_area_house_age

- avg_area_num_of_rooms
- avg_area_num_of_brooms
- area_pop
- price

4 Data Visualization

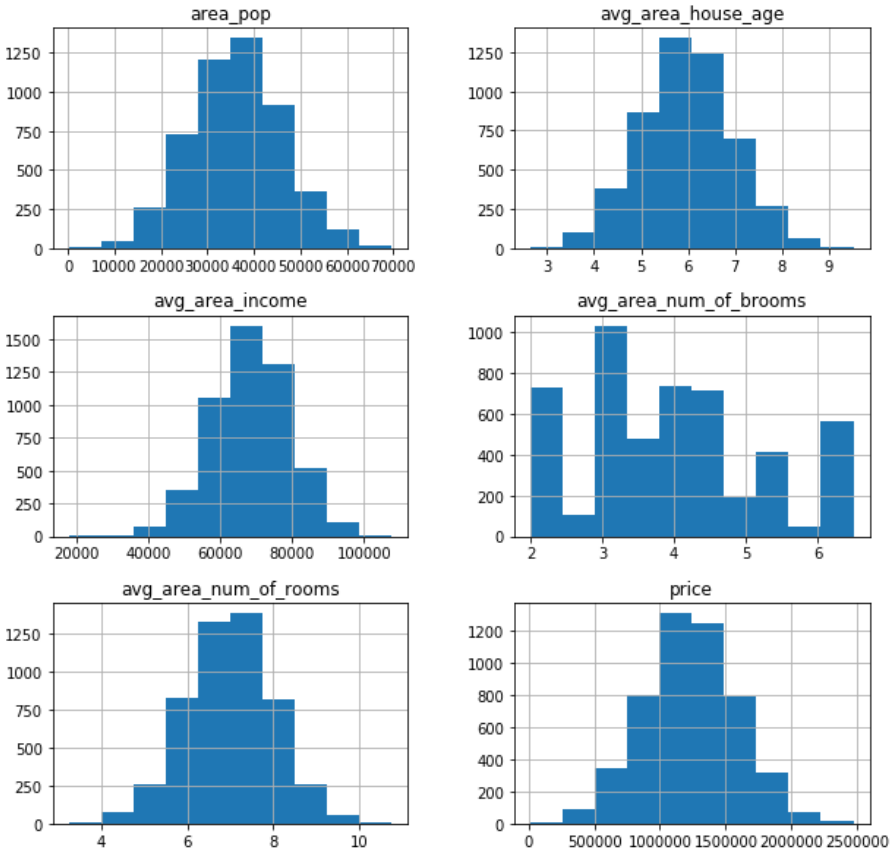
Data visualization is the representation of data or information in a graph, chart, or other visual format. It communicates relationships of the data with images. This is important because it allows trends and patterns to be more easily seen.[1] Figure 1 shows some statistics for our dataset like mean, standard deviation, etc.

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

Figure 1: Dataset statistics

To observe the width of spread I've plotted histograms in Figure 2.

A histogram is a bar graph of raw data that creates a picture of the data distribution. The bars represent the frequency of occurrence by classes of data (In our case it distributes it into different numerical bins). A histogram shows basic information about the dataset, such as central location , width of spread , and shape [2].



From the histogram we can observe that:

- For most of the rows (>1000) the average area population falls between 30,000 and 40,000.
- For most of the rows(>1000) the average age of the house falls between the range of 5.5 and 6.75 years.
- For most of the rows(>1250) the average area income falls between 60,000 and 80,000.
- The data for average area of bedrooms differs largely with one large peak at 3 (for around 1000 houses) and smaller similar peaks at 2,4 and 6.
- For most of the rows(>1250) the average area of number of rooms falls between the range of 6-8.
- For most of the rows (>1200) the Price range falls between 1 million and 1.5 million.

I used a scatter plot to understand the bivariate relationship between our input features.

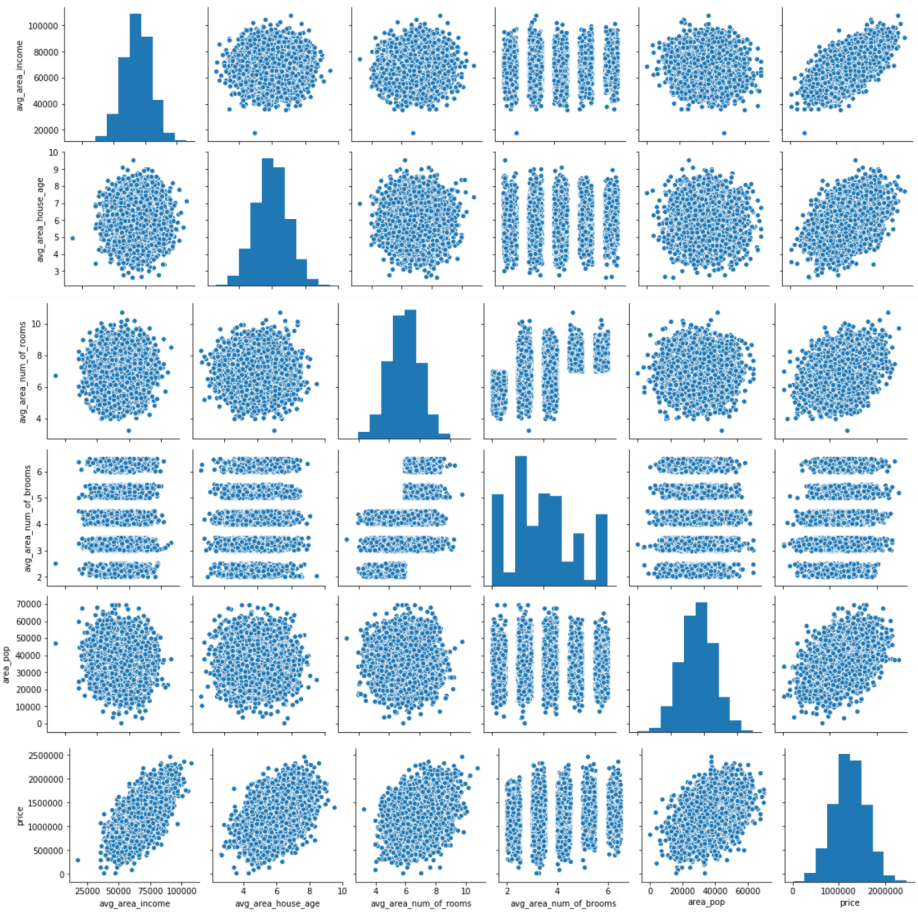


Figure 2: Scatter Plots

As we can observe from Figure 3, Our average area number of bedrooms (4th row) doesn't have a linear relationship with any of our input features including the price column. It would make sense to remove this column from our consideration. The rest of the columns seem to have a linear relationship with the target column price. Average area income seems to have the best/strongest linear positive relationship with Price. To confirm the findings I produced a heat map as shown in Figure 4.

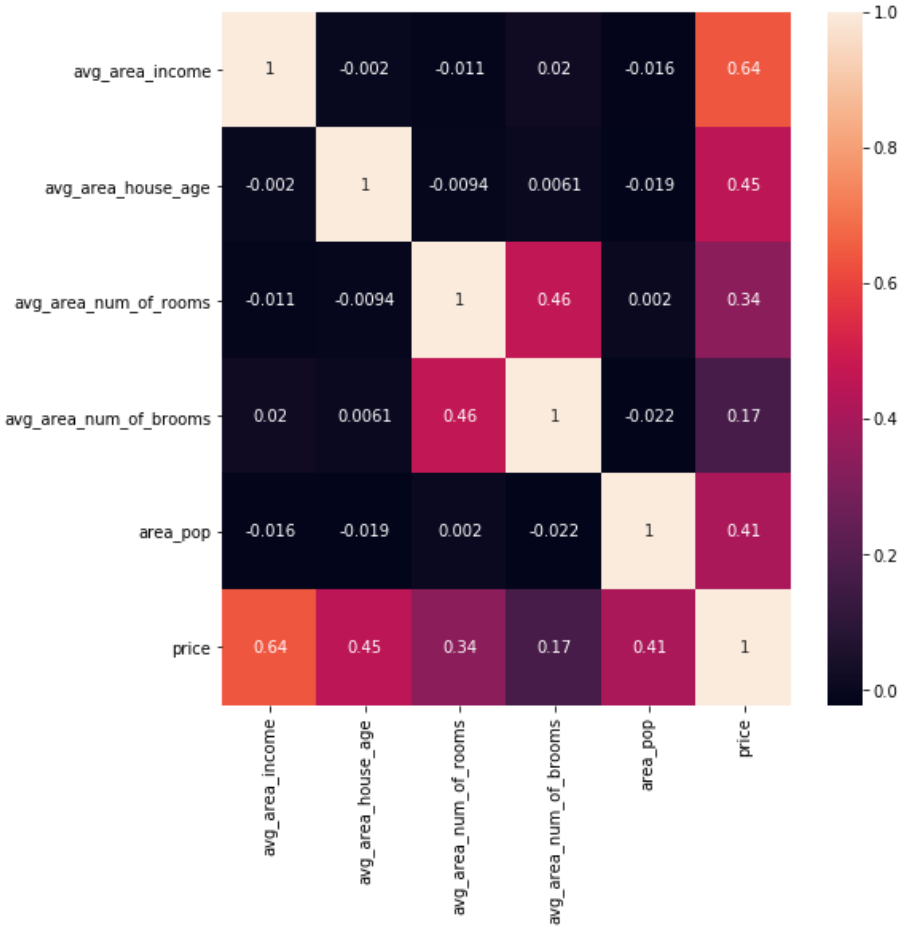


Figure 3: Heat Map

From the heat map it is obvious that price and average area income have the most correlation followed by average area house age , area population, average area number of rooms and average area number of bedrooms. Since the correlation between average number of area bedrooms and price is not completely zero this means there is some level of correlation between them therefore I have decided to keep it for now.

5 Data Normalization

Data normalization is done so we can get all the values of the features in a certain range. This helps the model to predict the data easily. At the moment our data is distributed in a very large range as shown in figure 4 and figure 5.

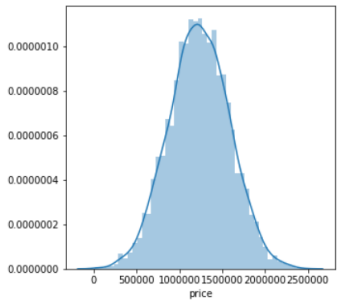


Figure 4: Target (Price) Distribution

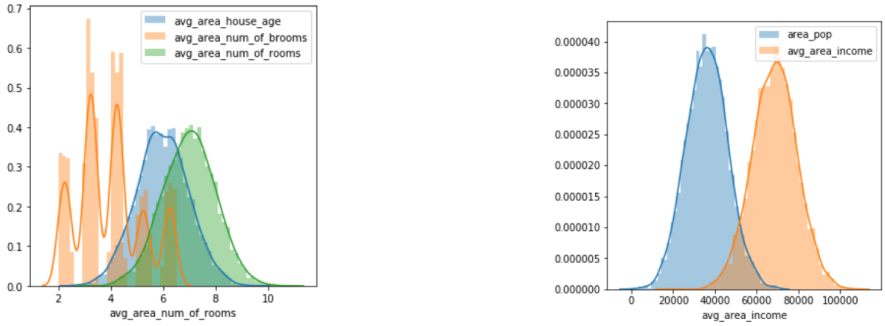


Figure 5: Input Distribution

As we can observe (from figures 5,6) all our plots except for the average area number of bedrooms (avg_area_num_of_brooms) have a normal distribution (Bell shaped).

In the initial phases of the project, to normalize my data, I took the mean and standard deviation for my input features and I divided the output feature for training data with the maximum value (Ymax) and divided the output feature for validation data also with the same maximum value (Ymax). But as I progressed from the initial phases, this normalization was not working well since the difference in numbers were too large. Therefore, I decided to take the log of the output feature and after prediction, I convert them back by taking exponent. Ofcourse, I understand that taking log and exponent always occurs in loss of information and I was willing to take that risk at this point.

6 Data Splitting

I have decided to do a 30%-70% split for the validation and training set. I trained the model with different combinations and found that this split was the best. Figure 6-8 shows the learning curves with different splits.

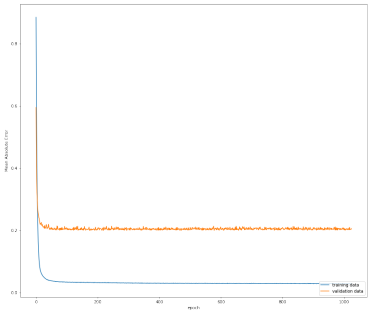


Figure 6: Too small training set (90%-10%)

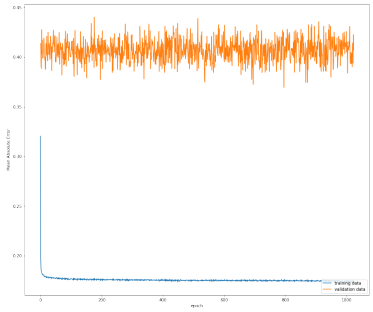


Figure 7: Too large training set (10%-90%)

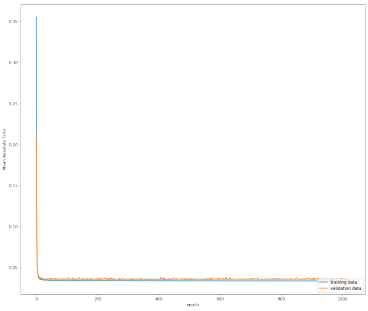


Figure 8: validation-30% training-70%

As we can observe from the learning curves our 30-70% split curve looks smoother than the others.

7 Evaluating different architectures

Based on the output feature of a dataset, the architecture usage varies. Since my problem is a regression problem a neural network with linear activation should work best. To confirm my theory, I evaluated the dataset against a logistic regression, linear regression, neural network with linear activation in last layer and neural network with sigmoid activation in last layer architectures. Figures 9-12 plots the learning curves and figure 15 compares the mean MAE and minimum val_loss attained during training these models.

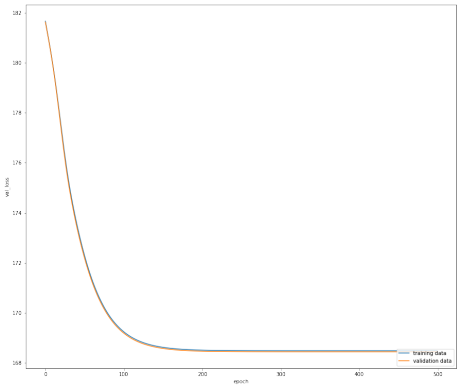


Figure 9: logistic regression

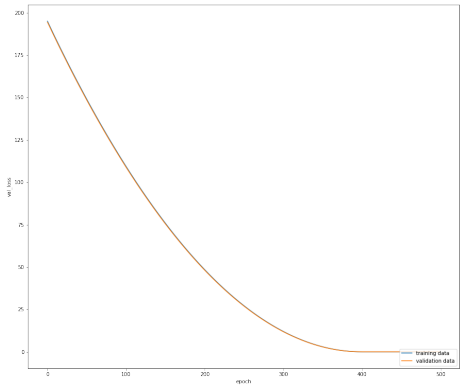


Figure 10: linear regression

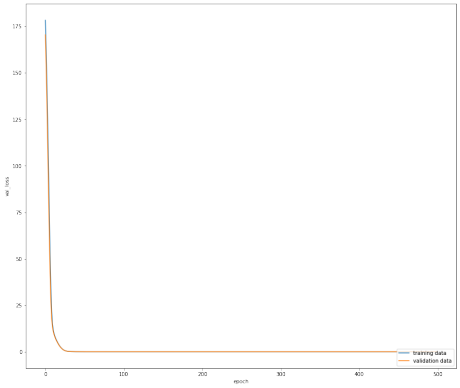


Figure 11: NN with linear activation

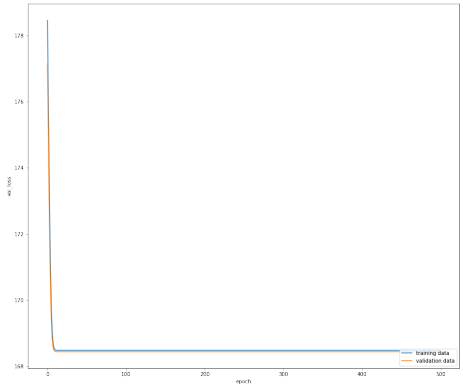


Figure 12: NN with sigmoid activation

Model	Mean MAE	minimum val_loss
Logistic regression	1232766.2617	168.44471740722656
Linear regression	420972.7194	0.015223732218146324
Neural network with 3 layers (12,8,1) and last layer having linear activation	397426.1049	0.010386980138719082
Neural network with last layer having sigmoid activation	1232767.9800	168.44471740722656

Figure 13: comparing mean MAE and minimum val_loss for different architectures

Even though our learning curves for linear activation in last layer and sigmoid activation in last layer look nearly same, when we compare the mean MAE and minimum val_loss values, linaer activation clearly does better than the rest. This confirms our theory.

Now that we have neural network with linear activation narrowed down for us, we can try different layers and neuron combinations to get the best model for our dataset. Figures 14-19 has the learning curves for a few different neural network architectures and figure 20 has the list of mean MAE values for these architectures.

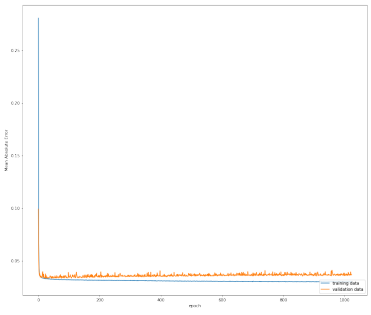


Figure 14: 6 layers: (18,12,10,8,4,1)

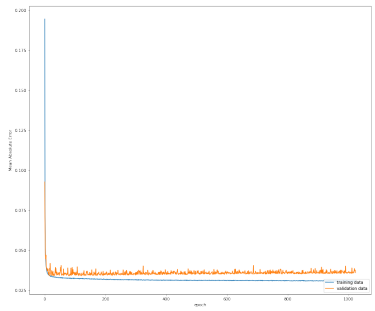


Figure 15: 4 layers: (20,12,8,1)

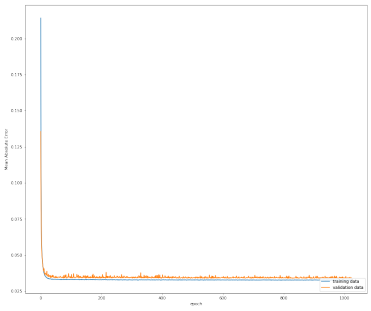


Figure 16: 2 layers: (12,1)

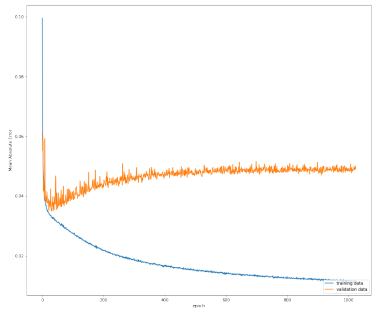


Figure 17: 3 layers: (200,100,1)

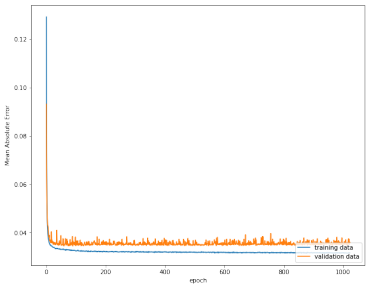


Figure 18: 3 layers: (20,10,1)

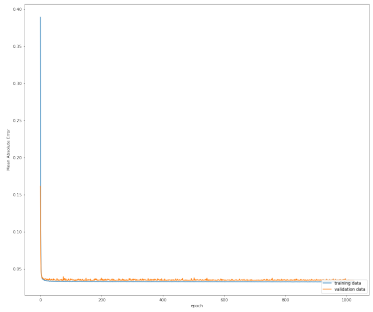


Figure 19: 3 layer: (12,8,1)

Number of layers	Number of neurons	Mean MAE
6	18,12,10,8,4,1	388321.4292489513
4	10,8,4,1	399140.1031792837
4	11,15,17,1	396602.8153225489
4	20,12,8,1	382494.8293113924
3	20,10,1	391449.1705888384
3	12,8,1	382156.49549582344
3	200,100,1	402105.31263154256
2	12,1	384993.38015162887

Figure 20: Comparing mean MAE for different neural network architectures

As we can observe, the best learning curve and the least MAE is achieved when we have 3 layers with 12,8 and 1 neuron respectively. The rest of the learning curves either look slightly underfit, overfit or have too many fluctuations.

8 Overfit models

Now that we have narrowed down the number of layers and number of neurons in each layer for our neural network, In this section we see how big our neural network model needs to be to overfit our data. We can do this by increasing the number of layers, increasing number of neurons and epochs during training. After the model has been trained, we predict the training set. I have tried different combinations in an attempt to get the lowest mean MAE. Figure 21 compares different combinations of layers, neurons and epochs with their mean MAE.

Number of layers	Number of neurons	Epochs	Mean MAE
5	256,256,256,256,1	1000	419710.421
6	256,250,150,100,50,1	700	407455.452
6	256,250,150,100,50,1	1000	383967.0292
6	256,250,150,100,50,1	2000	415189.291
6	256,250,250,200,100,1	2000	379906.062
6	256,256,256,256,256,1	2000	385185.843
7	256,256,256,256,256,256,1	3000	385761.5310

Figure 21: Comparing mean MAE in overfit architectures

As observed, the architecture with 6 layers containing 256 neurons in the first five layers and 1 neuron in the last layer gave us the best mean MAE. Further increasing or decreasing the number of neurons or the epochs did not improve our mean MAE. As such when we have a lot of data (number of rows) as I do in my dataset it can be hard to overfit the data.

8.1 Overfit model with output feature as an additional input feature

Now we try to overfit the architecture with the output feature as an input feature. We do this to reassure that our pipeline for the model is working. Figure 22 compares the mean MAE and minimum val_loss between the architectures when we supply output feature ad an input feature and when we don't.

Neural network	Number of layers	minimum val_loss	mean MAE
Without target output as input	6	0.018305	379906.062
With target output as input	6	0.005372	345138.504

Figure 22: Comparing minimum val_loss with and without output feature as input feature

As we can observe, providing output feature as an input feature gave us a much lower minimum val_loss and mean MAE when compared to the models where we didn't provide output feature as an input feature.Figure 23 plots the learning curve.

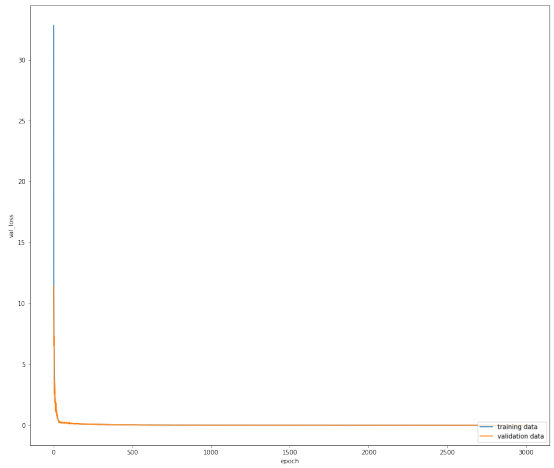


Figure 23: Learning curve when output feature is also an input feature

As we can observe, the curve does not have any fluctuations and is much smoother than any of the learning curves observed before.

9 Visualizing our neural network

We have done a lot of architecture comparisons in our previous sections and came up with the best model that has 3 layers with 12, 8 and 1 neurons in each layer respectively. Figure 24 visualizes this neural network for us [3].

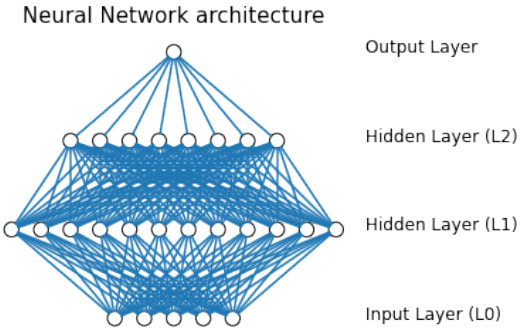


Figure 24: Neural network visual

10 Building a function for our neural network

In this section we build a function for our neural network model. After we training our data, we can extract the weights and bias that the model has learned during the training. We can use these weights and biases to create our own neural network model/equation. This is helpful because, When we provide a set of input features (5 inputs in our case) the equation will return the predicted output. Our output from the function should would be a close match the prediction made by the model when we used model.predict(). In this way, We would save the time spent on normalizing the input and training the model, and directly produce a predicted Price for a given sent of input features. I used the seed function before training my model so I can get reproducible results.Figure 25 compares has the output for the first 10 rows in the validation set.

Prediction by function	Prediction by training
14.034068347656216	14.221711
13.96186188531922	14.240022
13.864968635714236	13.786507
13.603162937087989	13.758002
14.40800418111987	13.962963
14.112935908028573	12.858111
14.699887463187517	14.459498
13.734147384209122	13.902208
14.171679212653501	14.256341
13.19280376274179	13.926971

Figure 25: Comparing predicted output by function and model

The mean error rate between the output by trained model and output by function is 0.3685. Figure 26 plots the histogram for the error rate.

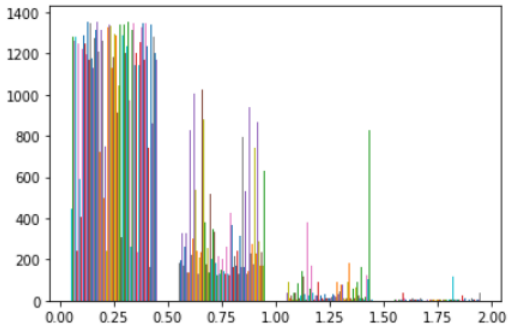


Figure 26: Plot showing error rate in prediction

As we can see most of the error rate falls between 0-1.5 and less than 100 fall between 1.5-3 which is still considerably small.

The weights and bias extracted from the trained model are a close approximation to the prediction by the model. This is why the values predicted by the model and by the equation/function are not exactly same. Therefore, a mean error of 0.36 is considerably good.

11 Feature importance and reduction

In this section we rank our input features according to their importance and train the model by iteratively removing the least ranked features. For this section I decided to not normalize my output feature to get the most accurate ranking.

For feature reduction I first trained each input feature alone and predicted the output price. This gave me the ranking of each input feature with respect to the mean MAE I got when I predicted the output feature price. Figure 27 plots the mean MAE achieved after training each input feature individually in the same order as the input columns. Figure 28 ranks all the input feature according to mean MAE (highest rank to lowest rank).

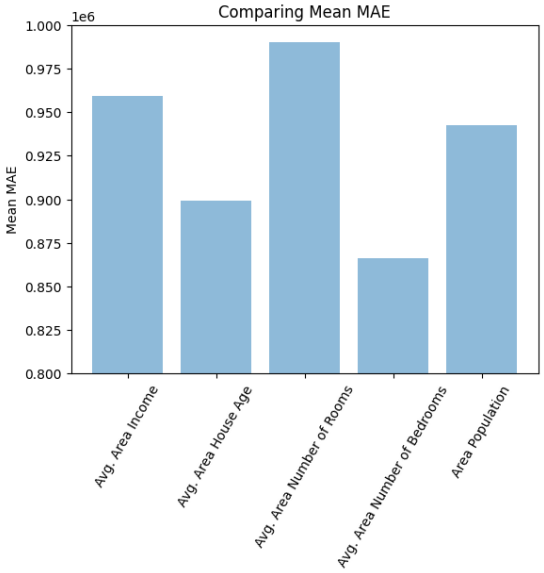


Figure 27: Comparing individual input features mean MAE

Rank	feature	Column number
1	Avg area number of bedrooms	3
2	Avg are house age	1
3	Area Population	4
4	Avg area income	0
5	Avg area number of rooms	2

Figure 28: Ranking input features

In phase 1 of the project we drew a heat map (Figure 3.) and found the degree of correlation among all the features. We found that price and average area income had the most correlation with output feature price and average area number of bedrooms had the least correlation with price. Comparing our heat map conclusion with the conclusions drawn from figure 28 and 27, we can see that, even though our heat map told us that average area number of bedrooms has the lowest correlation with price, Our model ranked this feature as the highest.

Now, I iteratively removed the input features starting from the lowest ranked feature and compared the mean MAE. Figure 29 plots the mean MAE after each input feature was removed.Figure 30 has the column names for the plot in figure 29.

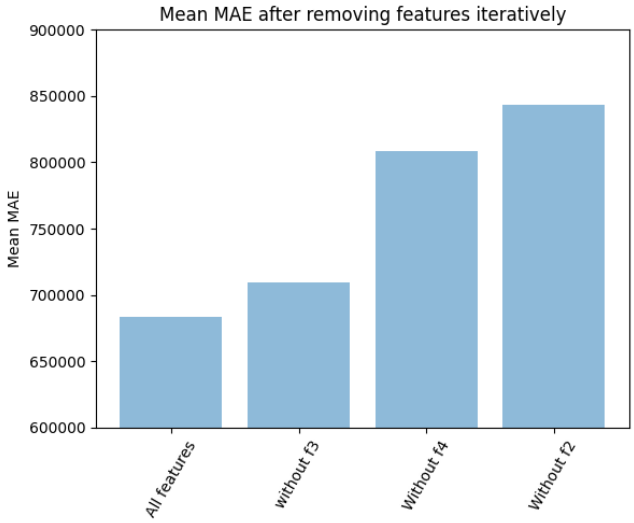


Figure 29: Comparing mean MAE of feature reduced model

Column Numbers	Column features
0	Avg. Area Income
1	Avg. Area House Age
2	Avg. Area Number of Rooms
3	Avg. Area Number of Bedrooms
4	Area Population

Figure 30: Column numbers for plot

We can observe that as we keep removing input features our mean MAE keeps increasing. This is expected because a model should perform better if more data in the form of features is provided to it. In the next section we compare our neural network model with XGBoost model.

12 Evaluating dataset using XGBoost

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance that is dominative competitive machine learning.[4] It is much easier to build models using XGBoost because here we don't need to normalize our data. We shuffle our dataset and split our dataset just like we did for our neural netwrok model. XGBoost builds a forest of trees after analyzing our dataset. The users can specify the number of trees and the maximum depth of the trees in the forest. Figure 31 shows an example of a tree from the

forest and figure 32 compares the mean MAE achieved from neural network and XGBoost.

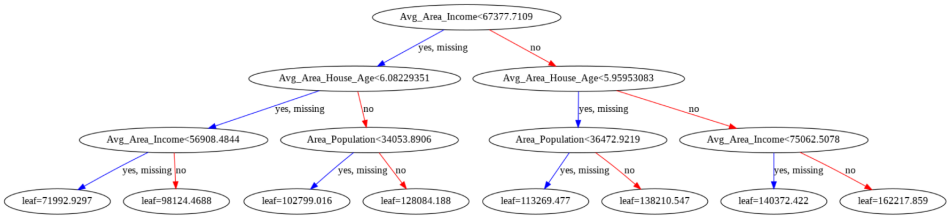


Figure 31: Example of tree created by XGBoost

MAE XGBoost	MAE neural network
86380.06	384046.15

Figure 32: Comparing mean MAE

As we can see the mean MAE we get with XGBoost is nearly half of what we achieved after using neural network. Clearly XGBoost helped us achieve a better model for our data.

13 Feature importance and reduction using XGBoost

XGBoost easily gives us a plot for feature importance using Fscore. Figure 33 shows the plot for feature importance. Figure 34 has a column names and ranking for figure 33.

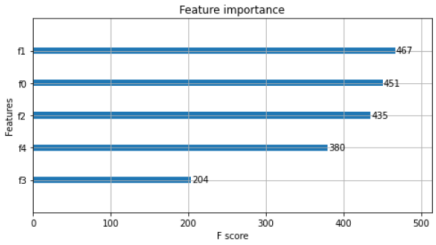


Figure 33: Feature importance using XGBoost

F	column name
f0	Avg_Area_Income
f1	Avg_Area_House_Age
f2	Avg_Area_Number_of_Rooms
f3	Avg_Area_Number_of_Bedrooms
f4	Area_Population

Ranking	column	column name
5	f1	Avg_Area_House_Age
4	f0	Avg_Area_Income
3	f2	Avg_Area_Number_of_Rooms
2	f4	Area_Population
1	f3	Avg_Area_Number_of_Bedrooms

Figure 34: Ranking features according to their importance

I iteratively removed the features and trained the model starting from the least ranked feature. Figure 35 compares the mean MAE after removing the input features iteratively.

Removed columns	MAE
nil	87657.13
f3	85515.96
f4,f3	147951.64
f2,f4,f3	180591.62
f0,f2,f4,f3	261431.434

Figure 35: Comparing original model to feature reduced model

As we can observe from the table we get the least mean MAE when we trained our model without feature f3, which is average area number of bedrooms. Further removing features form the dataset increases our mean MAE which is understood because our model has lesser information to work with.

In comparison to this model our neural network model showed us that average area number of bedrooms was the most important feature. XGBoost achieved the lowest mean MAE when compared to all the different models that I trained using neural network therefore I have decided to accept the model produced by XGBoost.

14 Future Improvements

Going ahead, I would like to try this problem as a classification problem by splitting my output feature into bins, for example: <10K, 10K-50K, 50K-100K, 100K-500K, 500K-1M, >1M.

I would also like to try and run this problem using binary classification by just having two outcomes expensive (Houses that are priced above 600K) and not expensive (Houses prices above 600K). Ofcourse what expensive means is different for different people.

I would also like to get my hands on a real life dataset hopefully one from Missouri and create a model for that. This would be very interesting and helpful for all potential home buyers.

15 Conclusion

Regression problems like predicting housing prices can be difficult to solve. It usually takes a lot of data for our model to learn and predict accurately. The artificial neural network model that I built still has some way to go because our XGBoost gave us a much lower mean MAE in comparison.

Neural network models are hard to fine tune, getting the right number of layers, number of neurons, setting patience level for model check pointing can be difficult. In XGBoost we only have to deal with two parameters that are number of trees and depth of trees. Neural network works best if we normalize our features, But in XGBoost we don't have to normalize our features. Therefore neural network can be hard to implement when compared to XGBoost, But, Artificial neural network models are very good at extracting deeper correlations between features when compared to XGBoost.

In all the different phases I have learnt a lot about the different ways to normalize data, Which activation functions to use for different kinds of problems, How to overfit data, How to analyze feature importance so we can omit unnecessary features from the dataset, How to effectively use early stopping and model checkpointing, How learning curves can give us a good idea of how our model is learning from the data and How to build our own model function from weights and bias.

References

1- Guide to data visualization from: https://www.import.io/post/what-is-data-visualization/

2- Histogram from: https://www.khanacademy.org/math/pre-algebra/pre-algebra-math-reasoning/pre-algebra-picture-bar-graphs/v/histograms

3- Neural network visual: https://stackoverflow.com/questions/29888233/how-to-visualize-a-neural-network

3- How to Develop Your First XGBoost Model in Python with scikit-learn: https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/