Machine Learning Project

Real estate price predictor

Authors: Moe A. and Rajin B.

5/8/2023

Topic Research:

The objective of the project is to build a ML model that utilizes current economic trends and Zillow house price data to predict the average price of a single-family home in the next quarter (3 months from current date). We envision this use for a ML model to be useful for real estate interments. It has the potential to help investors make better decisions by predicting future property value. It could also be used by homeowners to help estimate the future value of their house, helping to find the right time to buy or sell. On top of this, real estate agents can use this to better estimate market conditions thereby allowing them to better advise their clients.

In researching this topic there were a few things that were clearly going to be challenges we would have to overcome. The most prevalent challenge is the limited data out there. Small datasets pose more issues for model training and can hinder accuracy. Another challenge was out lack of domain knowledge. Our lack of domain knowledge coupled with the complexity of the real estate market is a significant challenge. The real estate market is influenced by many outside factors and by not fully understanding these relationships we hinder model performance. Some complex factors include the political landscape that bleeds into economic policy. This is something that is hard to quantify and teach a model on.

In researching similar projects, we saw a large majority had used a housing dataset provide by Kaggle. This included more detailed information of the house such as bedrooms, square footage etc. A few other datasets were self-assembled, utilizing web scraping to assemble an up-to-date dataset containing the latest information. These models tended to have more complexity but did offer more accurate predictions on current market trends.

In creating a model that predicts a house price value, common methods used were linear regression, neural networks, and occasional decision trees. Linear regression models utilize a y =

mx + b linear equation to best fit the inputs and predictions to the actual house values. While decision trees split the training data into many subcategories based on feature splits. Once you are deep enough and it reaches a prediction leaf the prediction is outputted and is expected to be similar to the houses who had similar attributes. Neural networks are similar to the idea behind linear regression. Fitting equation to the data except they consist of many nodes each with its own prediction and these combine to form a more complex model and more accurate prediction. All these models approach the problem differently but solve the same problem.

Recently some more complex models have used images of the house along with other information about houses to predict prices. This would be a combination of a convolutional neural network mixed with another type. The SOTA method (preferred method) is using recurrent neural networks. These are the go-to for real estate price prediction over time. They are used to predict future prices.

The metrics commonly used in a regression model are root mean error and mean absolute error. In our case for a regression model, we will be using mean absolute error, but the testing will use back testing as we are predicting three months ahead and want to determine that accuracy. You therefore need back testing because you can't use future cases to predict past ones.

Dataset:

The dataset used is a custom dataset complied of data from five separate sources. These included the Federal reserve data (inflation, vacancy rate, mortgage rates) and the Zillow data which included the raw weekly sale price and the estimate house value based on Zillow calculations. This custom dataset contained values past the 2008 housing crash to the current date. This resulted in us having limited data to train on. One the main issues with creating a

custom dataset was all the sub datasets recorded their data on different time scales. This means that some like the vacancy rate are reported every month while the mortgage rates are weekly.

The most current sub datasets (five total) were downloaded and complied into a single dataset. The dataset was comprised of records from 2008 onward to current date. Due to the time scale not matching up, we took the weekly approach and any columns that didn't have values in it due to being reported bi-weekly or monthly we filled in the values from the previous filled row. For example, the vacancy rate was reported monthly, for all weeks in that month it will hold the same vacancy rate. This allowed us to sync up the data and create a timescale that worked for all the data.

In reviewing the dataset re were few areas for potential bias. The feature adjusted price and adjusted value and next quarter relies on the Zillow dataset and their ML predicted house value. This essentially could add the biases from their models/datasets into our trained model. The interest and vacancy rates do not add bias as these are reported values by the Feds, and the date doesn't not add any bias at it's the chronological order for the data.

Data Analysis:

	interest	vaccancy	adj_price	adj_value	next_quarter
count	747.000000	747.000000	747.000000	747.000000	747.000000
mean	4.180750	7.882329	91929.621839	85955.680745	91585.281447
std	0.820783	1.560739	10492.184365	10287.494750	10146.662927
min	2.650000	5.600000	62248.590821	66602.818831	62248.590821
25%	3.635000	6.800000	84988.264243	78293.570231	84988.264243
50%	4.080000	7.300000	92446.164861	86512.591473	92286.069883
75%	4.615000	9.300000	98425.285613	92042.996096	97904.139387
max	6.700000	11.100000	119191.182657	106048.474106	118889.752618

Figure 1.0: *Breakdown of dataset distribution*

Data analysis was done using the pandas describe function. This gave information about the counts, means, standard deviations, min and max values, and percentile values. It shows the full breakdown of each column. This was extremely useful in better understanding our data. Using the counts, we were able to see how many missing values were present in each column. The mean gave a numerical value of where the central point in the data was for each feature. The standard deviation provided a measure on how spread out the data was from one another, and the min and max value provided information on the boundaries of the values. The percentiles gave us information on the distribution of the data.

Data cleaning:

The first data cleaning technique used was combining the datasets and adding missing values for those that didn't have weekly values. This was done to put all the data on the same time scale and to increase the number of data we had to train the model on. The second technique used was feature selection. This involved creating columns that took the inflation rate and house price and value and adjusted the house value based on inflation. Next quarter column was also added and involved going 12 weeks into future and pulling the price of houses that week and using it as the value for next quarter 12 weeks prior. Due to this method, there were 12 data entries at the end of the dataset we had to remove as they did not have a future next quarter value. On top of this, to simply the dataset we removed features we didn't need as our goal was only predicting next quarter values. These removed feature was the cpi (inflation rate). We adjusted the Zillow sale value and home value each week to take the inflation rate into account. This then got the actual value of the home rather than the price based on inflation. This was

needed as our goal was to observe the home value if it increased and we had to account for inflation.

Data Processing

A total of three data processing steps were done. The interest column and vacancy columns were normalized between 0-1. This was done to keep the columns on the same scale. It is useful for modals to have data on the same scale as to not place a higher importance on larger numbers, because numbers inherently have a hierarchy. Along with this the date column that was acting as the dataframe index was converted from the YYYY-MM-DD format into three separate columns. Year, Month, and Day. The Month and day were normalized, and the year was left alone. We choose this because the day and months are values that never change. Every year there are always twelve months in the year and the same days in each month. Normalizing would scale them down on the same scale as other columns. The year was not normalized because the years there is no upper bound. We thought that if we normalized and used this model years down the road the new year would be difficult as we would have to renormalize all the years again, this is because the MinMax scaler was used to normalize and changing the max would change what each years normalized value would be. The reason the three other columns of adjusted price, adjusted home value and house price next quarter were not normalized was due to the goal of the model predicting a house price. If we scaled these values down the model output would not be as relevant and would be difficult to interpret.

Model Implementation:

The models used were linear regression, KNN, and a neural network. The linear regression and KNN models were used from the sklearn library while the neural network was built using keras.

Linear regression works by modeling the relationship between the input variables and the output. It fits a linear line to minimize the difference between predicted values and actual values. Each iteration using gradient descent the loss is reduced to produce the best possible fit. KNN is a non-parametric algorithm used for regression tasks. The algorithm works by finding the K closest data points to the inputs and it makes a prediction based on those datapoints. It takes an average of the K nearest datapoints to make the prediction. It measures closeness of the datapoints in Euclidean distance in a multi-dimensional space that is how many inputs the model has. A feedforward neural network has the direction of information flow one direction, forward. The multiple layers of the model connect to one another and the input layers pass information to the hidden layers where a function (relu) transforms the information onto the next node until it reaches output layer. During training the weights of the neurons (nodes) are altered to minimize the loss. It uses gradient descent to update the weights. Linear regression was chosen as it is used as a baseline because of its simplicity and usefulness to compare to other models. KNN was chosen because its complex and can be very accurate for regression, however, it is slow and requires lots of training time and memory. The neural network was chosen because it's the most complex of the models, but it offers more control to the developer. There is full control over the layers of the network and the neurons, and the activation function used. This is why neural networks are often able to model complex relationships.

Linear regression model is easy to interpret and explain model and it is very efficient in terms of computation. The downside is it uses a linear relationship, and this is often insufficient for more complex relationships. KNN is useful because it can capture non-linear relationships within data, and it is excellent at handling outliers with datasets. The downsides are it requires a large amount of memory depending on the dataset size, as it saves all the values. It also requires lots of tuning to get high performance. The feedforward neural network has strengths in its ability to capture non-linear relationships and it can learn hierarchical relationships, which in this project can allow the model to place higher importance on certain features over others. The downsides are the computational cost for training and the model's tendency to overfit due if dataset is small, which is true in our case (less than 1000 entries).

Model Training and Tuning:

Of the models tested, we looked at the graph of error plotted against number of samples showing both training and validation error to detect overfitting and underfitting. From the data it showed for all three models that the training and validation error decreased with an increase in number of training samples. Looking at the graphs more carefully the neural network may have been overfit as there was a drastic decrease in training and validation errors and then as data increase the errors stayed relatively the same. This could indicate that we trained using to many epochs and the data was becoming memorized by the model. To address this, we changed the number of epochs the model was trained on, and the graph showed a very similar pattern. We believe this not to be a sign of overfitting after testing it and resulting in no change of pattern. In terms of underfitting there was no indicating that any model as the number of data samples increased the training and validation errors decreased. Indicating the model was indeed learning.

In our original draft we believed the neural network was underfit as it had a large MAE (mean absolute error) value. However, after changing the prediction to back testing this was resolved and had a relatively low value.

After the initial models were trained and we had some numbers to work with, we created a function to test a combination of hyperparameters and to report the combination that resulted in the best r-squared value, indicating the data was fitting the features better. The hyperparameters for the linear regression model were copy x, fit intercept, number of jobs, and positive. Copy x determines if input data should be copied. Fit intercept determines whether to use an intercept in the fitting of data. Number of jobs determines number of CPU cores to use in training. Positive ensures there are positive constraints on the coefficients of the model. The hyperparameters for the KNN model were nearest neighbors, weights, algorithm, leaf size, and metric. The nearest neighbors are the number of neighbors to consider for each datapoint. The weights adjust how the points in each neighborhood are weighted. Algorithm is what is used to compute the nearest neighbors. The leaf size affects memory of the network. The metric parameter chooses how the distance metric is calculated. For the neural network the hyperparameters were the number of hidden layers, the hidden layer size, and the activation function. These all related to the structured of the neural network and the functions to use for predicting.

The images below show the results from training and finetuning each model. The error metrics from training drastically improved for all three models after hyperparameter tuning.

```
MSE:  35094683.9957574
MAE:  4522.082486313134
R-squared:  0.11768965596841952
Training time: 0.005304915564400809
Best parameters: {'copy_X': True, 'fit_intercept': True, 'n_jobs': -1, 'positive': False}
Best R-squared: 0.5324503401609866
Best MSE: 4601943.833363999
Best MAE: 1817.749758269859
```

Figure 2.0: Linear regression metrics from training and hyperparameter tuning.

```
MSE:  68146861.40584347
MAE:  6697.8772745555025
R-squared:  -0.7132703271792076
Training time: 0.005274432046072823
Best parameters: {'algorithm': 'auto', 'leaf_size': 10, 'metric': 'euclidean', 'n_neighbors': 15, 'weights': 'uniform'}
Best R-squared: 0.33920776875207836
Best MSE: 6503969.513683878
Best MAE: 2020.5639845755836
```

Figure 3.0: KNN metrics from training and hyperparameter tuning.

```
MSE:  90069023.00731722
MAE:  7611.791732511393
R-squared:  -1.2644122023090842
Training time: 0.2548755577632359
```

Figure 4.0: Neural network metrics from training

```
Best parameters: {'num_hidden_layers': 3, 'hidden_layer_size': 128, 'activation': 'relu'}
Best R-squared: -3.4420509106914183
Best MSE: 176686552.59265766
Best MAE: 10339.61438729282
```

Figure 5.0: Neural network metrics from hyperparameter tuning.

Results:

Evaluating the models, we believe accuracy not the best measurement as the regression model is often never on the mark exactly with a continuous prediction. Instead, we believe the mean squared error is a good representation of the model performance as it indicates how far off the predictions were from actual values. Looking at the figures above from training and validation the linear regression model preformed the best with the mean absolute error of 1817.75. Meaning the predictions were on average $1817.15 away from the actual value.

```
Metrics from best Linear regression model
MSE:  38142481.58596632
MAE:  5089.312278647648
R-squared:  -0.25924733769810526
Model size: 1055 bytes

Metrics from best KNN model
MSE:  53815685.503942594
MAE:  6417.155178962315
R-squared:  -0.7766871970430913
Model size: 84022 bytes

9/9 [==============================] - 1s 3ms/step - loss: 8474119168.0000 - accuracy: 0.0000e+00
2/2 [==============================] - 0s 6ms/step
10/10 [==============================] - 0s 3ms/step - loss: 8088593408.0000 - accuracy: 0.0000e+00
2/2 [==============================] - 0s 5ms/step
12/12 [==============================] - 0s 2ms/step - loss: 6365272064.0000 - accuracy: 0.0000e+00
2/2 [==============================] - 0s 6ms/step
13/13 [==============================] - 0s 3ms/step - loss: 1792275712.0000 - accuracy: 0.0000e+00
2/2 [==============================] - 0s 4ms/step
15/15 [==============================] - 0s 2ms/step - loss: 292627296.0000 - accuracy: 0.0000e+00
2/2 [==============================] - 0s 6ms/step
17/17 [==============================] - 0s 2ms/step - loss: 97761176.0000 - accuracy: 0.0000e+00
2/2 [==============================] - 0s 4ms/step
18/18 [==============================] - 0s 2ms/step - loss: 69579632.0000 - accuracy: 0.0000e+00
1/1 [==============================] - 0s 23ms/step
5/5 [==============================] - 0s 2ms/step
Metrics from best neural network model
MSE:  46393562.614513434
MAE:  5845.3209405148
R-squared:  -0.5316510038023088
Model size: 251241 bytes
```

Figure 6.0: Final metrics for all models on test dataset

Figure 6.0 shows the results from testing all three models on the test dataset. In comparing their metrics, the linear regression model preformed the best with the mean absolute error of 5089.31. Followed by the neural network and then the KNN model. Addressing the loss and accuracy shown for the neural network, the accuracy of 0 is not an error. This is present because the model did not predict any houses to the exact value. This makes sense and gives a false sense of poor model performance as it is very unlikely the model will ever be exact with no error. This is the exact reason the mean absolute error was used in evaluating the models.

The training time for the models were as follows. Linear regression was 5.32 ms, KNN was 5.27, and the neural network was 254.88 ms. This was as expected with the neural network taking the most training time due to the number of layers and size of each layer. The training

time between the linear regression model and KNN were negligible where KNN was expected to take longer. We believe this to be due to the lack of size from the dataset. The KNN model did not have much data to memorize. In larger datasets this would likely take longer.

The sizes of the models were linear regression being 1,055 bytes, KNN model being 84,022 bytes, and the neural network being 251,241 bytes. This is what was expected and is no cause for alarm.

In terms of speed the KNN model preformed the best, but it was very close with the linear regression model. In terms of size the linear regression model took significantly less space while preforming better than all the other models.

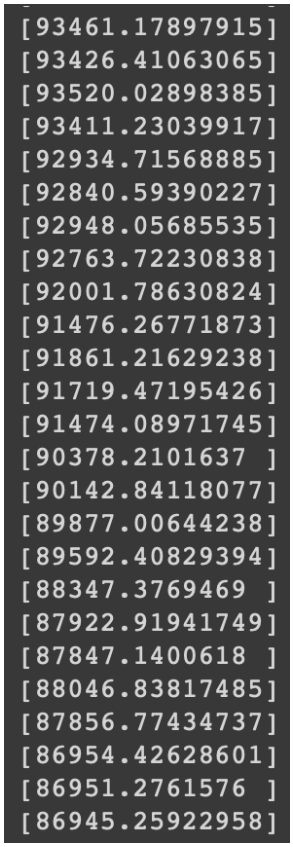

Figure 7.0: Sample output of the predictions the model predicted (linear regression model)

Discussion:

Based on the results and training and finetuning the linear regression model clearly performed better than all the models. This was seen with a better performance in both the training and validation stage as well as the final testing.

It was shocking that the neural network did not perform the best as that was what was initially expected. We believe the linear regression model was the best performer because the lack of data. Only having 742 rows of data with seven features and one target is not a lot of data to train a neural network successfully. This could have resulted in the model underfitting to the data and not having enough data to learn from. Due to the lack of data, it makes sense why the simplest model would perform the best. It has the easiest time to generalize the trends and fit a linear relationship using little data.

The models were not trained on a dataset with no data transformations, so it is unclear what affect our data transformations and normalizes played in model performance. However, we did play around with a multitude of hyperparameters and with all models were able to observe significantly improves to the model performance. This can be seen in figures 2-5. A large majority of the hyper parameters followed similar trends as other research articles suggested. Specifically with the neural network, increase in layers and layer size significantly reduce the model error.

If we were to go back and redo a part of the project, we would focus more time on data collection and looking for more complex relationships between house prices and different variables such as house area, tax rate, crime rate in the area, the level of schools etc. From our previous homework and projects, we saw how much importance goes into the data prior to training models and see this as the most beneficial area to focus on to improve model

performance. We also acknowledge we only used back testing to train our models and could try other methods such as cross validation to see if this had any effect on the model training.

Looking back a more beneficial approach would have been to make this a classification problem rather than regression. Our approach on predicting a value on a continuous scale is extremely hard for a model to learn the patterns within the data, it would have provided similar benefits to just predict 0 or 1 whether the house value would go up or down in the next quarter. While this may not have given a value, it still would accomplish most of the goal of helping buyers predict when the right time to buy and sell in the market.

Prior to releasing this model in production, we would have to investigate how the model would perform on raw data. We had normalized and beautified the data but in production the data will not be so nice and clean. We would need to learn more about how to reduce the risks to break the model with invalid or drastically different data. When the model is ready to be released in production it is important to provide users with as much information to make the results more interpretable. Doing this is hard because the models act as a backbox sometimes and we are unsure about the relationships being modeled. We think it would be helpful for the user to see training data distributions and predictions based on what the inputs were. This gives the user at least more information about what different types of properties were valued at and seeing their attributes allows the user to see trends for the pricing.

References:

https://samanemami.medium.com/how-to-measure-model-training-speed-249f34df492b

https://builtin.com/data-science/feedforward-neural-network-intro

https://www.mathworks.com/help/stats/what-is-linear-regression.html#

https://www.stessa.com/blog/10-real-estate-investing-metrics/

https://github.com/dataquestio/project-walkthroughs/tree/master/house_prices

https://tryolabs.com/blog/2021/06/25/real-estate-pricing-with-machine-learning--non-traditional-data-sources

https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4

https://www.kaggle.com/c/house-prices-advanced-regression-techniques