Predicting House Prices with Linear Regression and Neural Networks

Nir Bekker and Loki Sassone

ECE 196 Spring '21 April 30

Abstract

We decided to predict the prices of houses from the Kaggle dataset 'House Prices - Advanced Regression Techniques'. We used PCA analysis on the 50 out of 79 variables we deemed important to make a linear regression model to predict the prices of houses. We also used a neural network called MLPRegressor from the scikit learn python library to predict the prices of the houses. We achieved an average difference of 30,000 dollars between the predicted and the actual house price for the linear regression model and an average difference of 12,000 dollars for the neural network model.

1 Introduction

One of the most essential building blocks of our society is the idea of an investment. An investment is a current sacrifice of something valuable that in return will benefit us in the long run. Investments are present in every aspect of our lives. Getting a degree is an investment that costs us about four years and in return will help in starting a career. Of course, the most common investment is a financial investment. For many people, buying a home is the largest financial purchase they will make. Therefore, it is extremely important that people buy the right home. When purchasing a home, one must consider many attributes of the house such as lot area, number of bedrooms, and number of floors. Since a house has so many attributes we wanted to create a model that could easily predict the prices of houses and help the buyer make a good financial investment.

2 Problem-Solving Approach

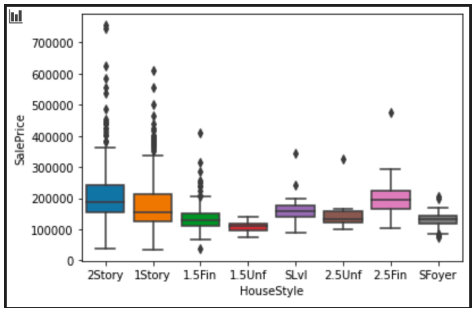
The best approach to tackle this problem is to divide it into sub-problems. The first sub-problem is to build a machine learning model that is able to predict the current value of the home. For the rest of the paper we describe our design and solution process for this sub-problem.

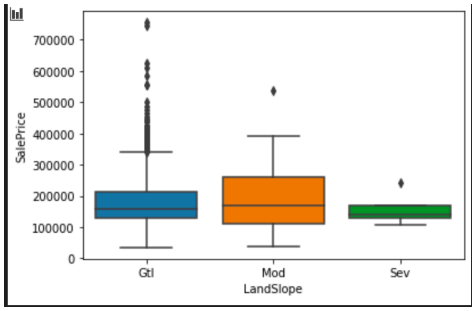
We begin tackling the sub-problem by finding a feature rich data set. We decided to use the Kaggle dataset, 'House Prices - Advanced Regression Techniques'. The dataset has 79 different attributes of 1400 homes and their final sales price in Ames, Iowa from 2006-2010. Processing the data to remove invalid entries is an important step that must be done before implementing any machine learning algorithm. We will be using the pandas python library for handling the data. After the dataset is cleaned we intend to implement linear regression and a multi layer perceptron regressor and compare their performance. Both machine learning algorithms will try to predict the sales price of homes based off of its most important features. We will be using the scikit-learn python library for implementations of machine learning algorithms and analysis of results.

3 Solution

3.1 Data Processing

Initially the Kaggle house prices dataset had many string variables. For example one of the variables named 'Street' had two options of 'gravel' or 'paved'. Since linear regression and neural networks take number values as input, we converted all of the string variables into numbers . We did this by converting each option of a variable to the numbers 10, 20, 30 and so on. So 'gravel' would now be replaced with a 10 and 'paved' would be replaced with a 20. Next we decided to visualize each variable's effect on the sale price of the house to determine if that variable is needed to be included. We used the python library Seaborn for data visualizations. Below are two box-plots for the variables 'HouseStyle' and 'LandSlope'.





The first box-plot for 'HouseStyle' shows that the different variable options affect the distribution of the price dramatically. We can see that the average price of the house jumps around between different styles, therefore we chose to keep this variable. However, for 'LandSlope', we can see that the distribution and the average price of the house are very

similar across different Land Slopes. Therefore, we chose to exclude this variable from the data. We reduced the data set this way from 79 attributes to 51. Before implementing linear regression, we wanted to further reduce the dataset to lower the computation time (for the Neural network we leave all 79 attributes). We used Principal Component Analysis(PCA) from the scikit-learn library to narrow down our dataset to have only 8 attributes that cover about 98 percent of the variance in the data. PCA eliminates variables that are closely related and chooses the most prominent variables in the data-set that account for as much variance as possible. Lastly, we split the data into testing and training data and began implementing the two machine learning algorithms.

3.2 Linear Regression

Now that the data is processed and cleaned, we can implement linear regression. Linear regression is an algorithm that predicts an outcome from a linear combination of data values and corresponding weights as coefficients. In this case the eight remaining attributes in our dataset and eight weights will be linearly combined to equal the sale price. The equation will look like: $w_1x_1 + w_2x_2 + \ldots + w_8x_8 = prediction$. Linear regression will adjust the values of $w_i, i \in (1, 8)$ to minimize a loss function. The loss function used here is the sum of the differences between the predicted house price values and the real house price values squared. Hence, when the loss function converges, the weights w_i stabilize and passing test data through the linear function will get a prediction. Our linear regression model peaked in performance with a 30197 dollar average difference between the real house price and the models predicted house price. The following picture shows the final weights $w_i, i \in (1, 8)$ of the linear function and the first five House prices vs the predicted house prices.

```
Coefficients:
 [  5.15272207 -87.99887449  26.32510143 -83.24382806  30.70260305
 -58.53808066 -26.46988854  58.29291649]
```

```
:
print(y_test[:5])
print(y_pred[:5])
```

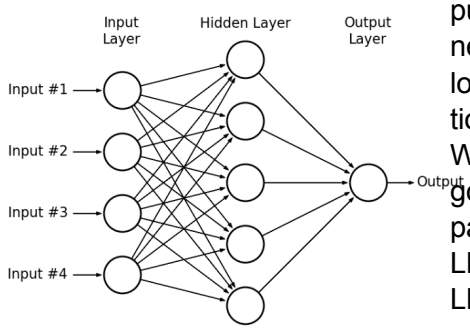
```
[111000. 538000. 226700. 110000. 125000.]
[105890.42820902 375945.18587186 193972.64094548 106645.06417968
 127020.39177032]
```

For example, the price of the first house was 111000 dollars and our linear regression model predicted about 105890 dollars. In the next section we try to improve our results using a neural network approach to predicting house prices.

3.3 Neural Network

Since our linear regression model didn't predict the house prices as well as we hoped, we decided to use a neural network model. We used the MLPRegressor function from the scikit learn python library. MLPRegressor is a Multilayer Perceptron regressor that optimizes the squared loss function using

either stochastic gradient descent or the Broyden–Fletcher–Goldfarb–Shanno (LBFGS) algorithm. A Multilayer Perceptron (MLP) is a type of artificial neural network which is based on connections between different nodes in the input, hidden, and output layers.

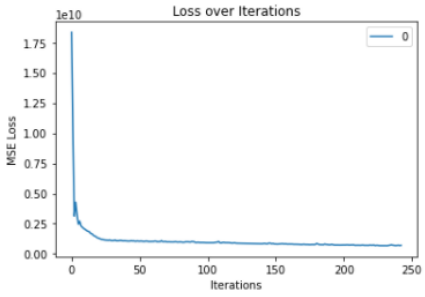


A MLP takes the weighted sum of all input nodes, adds a bias, then passes through a nonlinear activation function. Squared loss takes the summation of the square of the difference between the predicted house sales price and the actual house sales price. MLP consists of many different hyperparameters, values used to control the learning process of the MLP. For example hidden layer size would be a hyperparameter as it is a parameter that would change how in depth the MLP would train. Some other examples of hyperparameters are the learning rate of the optimizer and the alpha regularization value used to avoid overfitting the training data. The MLP for our model consisted of a hidden layer with 1000 nodes. We ran different hidden layer sizes of 20, 50, 100, 500, 1000, 2000 and we found that after a hidden layer size of 1000 the accuracy of our model didn't increase. The activation function we decided to use was the Rectified Linear Unit (ReLU) function. The ReLU function outputs either the input if positive, and 0 if the input is negative. The other options of tanh, logistic, and identity activation function had lower accuracy than ReLU. We decided to use the LBFGS algorithm as our solver because compared to stochastic gradient descent, LBFGS had the highest accuracy. LBFGS updates the weights in each node by using the following equation:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - H(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k).$$

x(k) and x(k+1) are the node weights to be updated. H(x) is the Hessian of the input, which is a square matrix of the second order partial derivative of the output from each node being optimized. The gradient of f(x) is the first order partial derivative of f(x). The above equation is derived from minimizing the second-order Taylor-series expansion to find the step size used for updating the weights. We calculated the accuracy for our model by taking the average of the difference between the predicted house price and the actual house price. We noticed that the training dataset and testing dataset error had very similar accuracy and

decided that the default alpha value of 0.0001 was sufficient. We also noticed that the learning rate only affected how fast the model would converge so we kept the learning rate to a constant 0.01 and kept the maximum iterations of our model to 1000. As you can see from our loss plot below, the model would converge at around 240 iterations.



Our best MLPRegressor model achieved an average difference of 12,000 dollars between the predicted house price and the actual house price. Considering the prices of houses in our dataset ranged from 100,000 to 300,000 dollars, a 12,000 dollar difference is quite accurate. We also made a way to calculate percent accuracy by dividing this difference between predicted and actual sales price by the actual sales price. Below is an image of how we would calculate accuracy and percent accuracy using the numpy python library for array manipulation.

```
y_pred=model.predict(X_test)
print("Test Accuracy:")
print(1-np.mean(abs(y_pred-y_test)/y_test))
print(np.mean(abs(y_pred-y_test)))
y_pred=model.predict(X_train)
print("Training Accuracy:")
print(1-np.mean(abs(y_pred-y_train)/y_train))
print(np.mean(abs(y_pred-y_train)))
```

Our best percent accuracy was 92 percent.

4 Conclusion

Although we got good accuracy with our neural network model, we ran into many challenges and setbacks along the way. Our first setback was the amount of time it took to visualize all 79 variables in the Kaggle dataset. Furthermore, converting all string variables to number values and getting rid of null values took quite some time. Also, training the MLPRegressor model took anywhere from 10 to 30 minutes each time, making it very difficult to tune the hyperparameter values to achieve more accurate results.

Predicting current house prices is a good stepping stone for being able to predict future house prices. Being able to predict future house prices is a much more accurate approach to helping people make smart financial decisions. For example, a home owner can anticipate the price of their house in some amount of years and make a larger profit. We can improve our working model to be able to make these sort of predictions. Keeping the model the same but adding more variable such as average price of homes in the area and the rate population change could steer the model to predict future house prices.

5 References

- https://scikit-learn.org/stable/modules/linear$_{m}odel.htmlelastic-net$
- https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.htmlsklearn.decomposition.PCA
- https://scikit-learn.org/stable/modules/generated/sklearn.neural$_{n}etwork.MLPRegressor.html$
- https://builtin.com/data-science/step-step-explanation-principal-component-analysis
- https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview
- https://numpy.org/doc/stable/reference/
- https://towardsdatascience.com/bfgs-in-a-nutshell-an-introduction-to-quasi-newton-methods-21b0e13ee504