Predicting Final Sale Price of Homes using Machine Learning

Hailey Johnson, Nathan Mitchell, Katie Myers

Data Science Department, Colorado State University

DSCI 478: Capstone Group Project in Data Science

Dr. King

February 21, 2024

Our project aims to utilize various machine learning techniques in order to predict the final sale price of a home using multiple explanatory variables related to different aspects of the homes that are on sale. The information regarding these homes comes from the Ames Homes dataset which was arranged by Dean De Cook. We will first outline some of the problems that we ran into when first starting this project, such as the correct way to format the data, and then we will move into the different models we used in order to solve the problem. We will first outline the first models we used, LASSO and Ridge Regression, and the results garnered. Second, we will discuss our various approaches using Random Forest and the different results from the models fit. Finally, we will discuss our attempt at implementing a Neural Network in order to solve this problem and our experience with training it. After the models we used have been discussed, we will conclude with an overview of our results and discuss the model that gave us the best RMSE – the model we used for our final Kaggle submission – the LASSO.

First, we will outline one of the more basic problems that we ran into – getting training and testing data for our model. The Kaggle competition comes with these two built in, but the testing data is missing the final sale price (as these prices are what the competition is judged on). Thus, we decided to split up the training data that was given to us even further into training and testing. We decided to use 20% of the training data for testing and the other 80% to train our models. There were only 1,460 rows in our training set, so we weren't keen to split our data up even further, but we wanted to at least get an estimate of the overall testing error considering we wouldn't know the actual testing error, so we opted to split our data.

Once we started to train our models, one of the first problems we ran into was that the machine learning models we wished to use have no inherent way of dealing with categorical variables. As we quickly learned, most of our variables – nearly 50 of 80 – were categorical.

Within these categorical variables were multiple different levels, not just two (one level for 0, one level for 1 as in typical dummy coding). We didn't want to throw out nearly 50 variables, nor did we want to manually encode these dummy variables by creating multiple new columns for each variable. Thus, we did some research and found a Pandas command that would do this automatically (pd.get_dummies). After adding the dummy variables and dropping the columns that contained the categorical variables, we were left with over 800 columns to use for training for the Random Forest and Neural Network approaches.

The implementation in R was slightly different. We wanted to use the 'regsubsets' function from the package "leaps" to do best subsets selection so that we could decrease the predictors needed and increase interpretability without losing too much valuable information. However, this proved to be too difficult because of how large our dataset was, and how many columns were added by our categorical predictors. We decided to use all of the available information, using One-Hot encoding so that we can use our regression techniques. We were left with a similar amount of columns after using the One-Hot encoding on the training set, about 800--however, there were different amounts of levels within the categorical variables between the training and testing set. To accommodate this, we had to perform the One-Hot encoding on the combined dataset with training and testing together, and then split them back up after the encoding was added. This resulted in a shocking 5,005 total columns. This was certainly not ideal, but since we didn't want to lose any of the information, we decided to continue and implement the LASSO and Ridge Regression models with this data.

Before implementation, we had to decide on a metric to use for comparison across models. The Kaggle project uses the RMSE with the logarithm of the predicted value and the logarithm of the observed sales price for evaluating submissions. They do this so that the results are better protected against outliers--“errors in predicting expensive houses and cheap houses will affect the result equally” (Montota, 2016). However, for the purposes of model selection in our project, we decided to use the standard RMSE metric so that our errors would be in the same units as the final home sale price (USD). Then we can directly compare our models’ errors against each other, as well as against the overall standard deviation of final home sale price, which was $79,442.50 in the training dataset.

The first two models that we fit were the LASSO and Ridge Regression. We decided to use shrinkage methods because of how many predictors we had--we knew that not all of them would be useful or relevant in our goal of predicting final sales price. Before implementing these methods, we standardized all numeric predictor variables to ensure that the weights of each were balanced against final sales price, and that no predictors had more power over others. We used the cross-validation approach in order to pick the optimal value of lambda, the shrinkage parameter, for both LASSO and Ridge Regression. Interestingly, the optimal value of lambda varied greatly between LASSO and Ridge Regression, the latter being much higher than the former. The RMSE also differed greatly between them. The LASSO model’s RMSE was only $30,011.74, while the Ridge Regression model’s RMSE was $49,426.45.

Random Forests was a clear choice for us to use not only because it was one of the models outlined by the Kaggle competition description but because it made implementation easy. Other machine learning models typically require the values belonging to each feature to be normalized, but Random Forest does not have this stipulation. This saved us a few steps and a few extra lines of code in our pre-processing. Another reason this was a favorite was because Random Forest does not require the use of cross validation as this process is built into the algorithm. This got us close to a final model, but we still wanted to tune some hyperparameters in order to get the best model that we could. Thus, we looked at using GridSearchCV from Scikit-Learn and tuned the maximum depth, number of estimators, and maximum features (Koehrsen, 2018). Once the model was tuned to find the optimal combination of values for these parameters, we fit the final model using these values. This model used data from all of the predictors and all of the different dummy variables that were created previously. The best model came from a maximum depth of 8,800 estimators, and 9 features. This model got us an RMSE of $62,642.87. As can be seen, this is worse than both LASSO and Ridge Regression.

Based on some of our other testing, we saw that it might be beneficial to not include any of the categorial features and for us to remove all of the dummy variables that had been created, bringing us back down to just over 30 parameters to use to train our model. Thus, we went through the same process of using GridSearchCV from Scikit-Learn but on a version of the dataset that did not contain any of the categorical variables. Once the optimal model was fit, we achieved an RMSE of $37,592.51. This model used a maximum depth of 10, 700 estimators, and 4 features. As can be seen, these are clearly different from before and gave us a slightly better RMSE.

One final method we tried, once again based on other testing, was training a model on the data without the categorical data that did not utilize any hyperparameter tuning. Doing this gave us an RMSE of $35,755.64, which is better than either of the other two methods that we had tried, but we figured that this may have been because of overfitting. If not due to overfitting, it is likely because of the randomness associated with training a model. Multiple attempts at running this code gave varying RMSE estimates, some higher than the one given and some lower. Thus, a low RMSE is likely due to luck and would not perform well on the final set of predictions as the goal of this competition is to get the best overall predictions of final sale price on data that the model has not seen. If the model is overfitting on the data we have given it or our good results are due to luck, it is likely to perform poorly on the actual testing dataset used to judge us. Thus, we moved on from this unoptimized model and went on to try one final method.

Finally, we turned to using a deep neural network. Given that our Random Forests models took a significant time to run, we were hoping that the interconnectedness of the deep neural network algorithm would lead to lower run time, while also outperforming our previous models. The preprocessing steps we used for the neural network were the same as the first Random Forests model, with the categorical variables being One-Hot encoded. Additionally, we normalized the data that contained our training features, a necessary step so that all features were treated with equal importance. Normalization was also important as for when the neural network was implemented so that we didn't end up with an exploding gradient, which would make the neural network untrainable.

Once our data was preprocessed, we then needed to build the neural network. None of us had ever made a neural network model before, so we decided to utilize APIs; we chose to use industry standards Tensorflow and Keras. We started by creating a sequential model and adding two hidden layers. We chose to use two hidden layers as this reduces the number of nodes we need overall, and helps compensate for the imprecise nature of node selection. As just stated, there is no standard way to determine how many nodes per hidden layer to use, however too many nodes and the model will overfit the data, and too few and the model will underfit the data. Stathakis (2009) provides formulas from previous research for the number of nodes to have at each hidden layer, but also states that the most accurate network will have fewer nodes than determined by the formulas. So we decided to use these formulas as a starting off point, with the equation for the first hidden layer being $\sqrt{(m + 2)N} + 2\sqrt{N/(m + 2)}$ and $m\sqrt{N/(m + 2)}$ for the second layer, where m is the number of outputs and N is the number of samples (Stathakis 2009). This gave us 84 nodes for our first hidden layer and 17 nodes for our second hidden layer. We decided that our activation function for these hidden layers would be Rectified Linear Units (ReLU), as compared to other activation functions, ReLU does not have the issue of vanishing gradients.

Once the neural network was built, all that was left was to compile the model. We chose our optimizer to be Adam, which implements stochastic gradient descent, and adaptively updates the weights of each connection between neurons. Given that we are determining which model we use based on RMSE, we chose our loss function to be MSE. We then fit our training data to our model and got an RMSE of $38,701.20. We then wanted to try using fewer nodes to see how this affected our model. We arbitrarily decided to take the original number of nodes of 84 and 17 for hidden layer one and two, respectively, and round them down to the nearest power of two. We decided to do this as it appears to be convention to have the number of nodes be a power of two (albeit not a concern of ours, is most likely for CPU reasons), which gave us 64 nodes for the first hidden layer and 16 nodes for the second hidden layer. When fitting this updated model we got an RMSE of $38,139.82, which is very similar to the previous model. We were pleased to see our model with fewer nodes performed similarly, as this means we eliminated potentially unnecessary nodes from our model, but decided to end our experimentation here as decreasing our number of nodes too much could lead to underfitting.

The neural network met our original goal of a less time intensive model to execute than the Random Forest models, as fitting the model for the neural network took less than 30 seconds compared to around 45 minutes for the random forests. Unfortunately, our neural network models did not outperform the random forest model in terms of RMSE. One final thing we want to note about the RMSE related to the neural network is that, just like the RMSEs of the Random Forest models, it is variable to change depending on the specific data used for training and testing. Overall, we found that it was rare to get an RMSE that was consistently below the RMSE of our best models.

Once we had trained and tested these four models, we had four different RMSE estimates to work with that would help us to determine the best model and the one that we would use for the Kaggle submission. For reference, in terms of RMSE, the LASSO achieved $30,011.74, Ridge Regression $49,426.45, Random Forest $35,755.64, and Neural Network $38,139.82. To gain more insight into how well these models predicted the home sales price, we must look back to the standard deviation of the sale price found within the training set to give us a baseline. From the metric discussion earlier, we had found that the standard deviation was nearly $80,000, meaning that if we could achieve an RMSE of below $80,000 – hopefully *well* below $80,000 – we would be happy. As can be seen from the results above, we were able to achieve this with all of our models. We, however, need the *best* model, so we ultimately chose the LASSO model to use for our final predictions as it achieved the lowest RMSE.

References

Koehrsen, W. (2018, January 9). *Hyperparameter Tuning the Random Forest Classifier*. Towards Data Science. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

Martins, G. (2023). *House Prices Prediction using TFDF*. Kaggle. https://www.kaggle.com/code/gusthema/house-prices-prediction-using-tfdf/notebook

Montoya, A. (2016). *House Prices - Advanced Regression Techniques*. Kaggle. https://kaggle.com/competitions/house-prices-advanced-regression-techniques

pandas (n.d.) *pandas.get_dummies*. https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html

Scikit-learn (n.d.) *sklearn.ensemble.RandomForestClassifier*. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Stathkis, D. (2009, April 30). *How many hidden layers and nodes?*. International Journal of Remote Sensing. https://www.tandfonline.com/doi/full/10.1080/01431160802549278