Project Report

Introduction

Ames Real Estate Co. (AREC) sponsored a contest to see how accurately Data Scientists could predict house prices based on historical data. Furthermore, their new CTO Randy Cornell also expects us to present our findings as well as include recommendations on next steps.

The data itself comes from two csv files, “train” and “test,” which include about 3000 rows of data and 80 variables, ranging from square footage of the garage, number of rooms, neighborhood of the house, and even the month it was sold. Per the contest, the column pertaining to the price of homes “SalePrice” was withheld from the test dataset as, upon submission, it will be used to score my model. There was also a text file which described all the features and how it was coded as well as a published paper which talked about the dataset. In it, the author of the Ames dataset Dean De Cock mentioned that certain data points were outliers and should be discarded.

Data Preparation

Section 3.1 is where I started initial EDA and data cleaning. I saved and dropped the “ID” columns from both training and testing datasets because I’ll use it again when I submit my final data. Per DeCock’s article, I deleted data points where the feature ‘GrLivArea’ was greater than 4000 because he said they were true outliers. This also increased my Pearson R metric by 2% and narrowed the confidence interval around the regression line.

Section 3.2 is where I started exploring the data. I saved the shape of the training and testing dataset before concatenating them together. I called this combined dataset “all” and it

composed of 2915 rows and 79 columns. I also saved the values of SalePrice so I could use it later for training and modeling my data. After, I took a cursory look at my data and identified the feature names and amounts of categorical features vs numerical ones (46 and 32, respectively). I checked the amount of unique values per column because that also helps to identify which ones were truly continuous and which ones were categorical (even if they were listed as numeric). Last, I looked at a table of summary statistics of the continuous variables. Other than giving me another way to glance at my overall data, I used this to identify erroneous values by looking at the min and max values. For example, for the feature 'GarageYrBlt,' which denotes the year the garage was built, the max value is "2207." This is obviously an error.

Section 3.3 is where I explored and imputed my missing values. Six features had at least 15% of the data missing, three of which were over 90% missing. For the most part, I imputed the missing values using simple metrics and logic. For categorical values, missing data typically meant "None" so I imputed that value with the string "None.". For continuous features, missing values were imputed using "0" to denote none. I had to ensure that there were no "None"-types or np.Nan's as some of my models will not accept them. Another technique was imputing via the mode as some categorical features had an overwhelming majority in one class compared to the rest. My most advanced imputation was for 'LotFrontage' which meant the, "Linear feet of the street connected to the property." I imputed using the median of the 'Neighborhood' it belonged to, as it should be similar. I also dropped the feature 'Utilities' as it held no predictive power. It was overwhelmingly in favor of one class, and the test dataset only had one class itself.

Section 3.4 is where I checked for duplicate rows. Thankfully, there were none.

Exploratory Data Analysis

In this section, I explored the correlations between 'SalePrice' and the independent variables. I also performed feature engineering and ultimately prepared my data for modeling.

I started with a Heatmap in section 4.1. I saw examples of multicollinearity such as between ‘GarageArea’ and ‘GarageCars.’ This made sense as the size of the garage increases, so should the number of cars that can fit inside it. After, I narrowed the Heatmap to just features that were highly correlated (>0.50) to SalePrice. This identified ten features. The most correlated was ‘OverallQual’ as the higher the quality of the home, the higher the SalePrice. There were other such variables including the year it was built, square footage of certain sections of the home, the number of rooms, and the number of cars.

I used these highly correlated features in Section 4.2.1, where I created new polynomial transformations of them. The reason for doing this is acknowledgement that often data cannot be best “explained” using a straight line of best fit. Sometimes, the best line is nonlinear. Using the top ten features, I created square, cubic, and square-root polynomials as these are possible explanations of the data.

The following subsections leading up to Section 4.3, are closer inspections and subsequent preparation for modeling for each of the remaining original features. I categorized them logically by type and analyzed most features by at least three different visuals. When analyzing a categorical feature, I looked at it using a boxplot, strip plot, and a barplot. The boxplot gave me an idea of the distribution of the data, the stripplot the volume of data in each class, and the bar plot the average with the size of the error. When analyzing a numeric feature, I used a scatter plot to visualize the interaction between SalePrice and the variable, as well as boxplots or barplots to see how the feature varied when related to other features. For example, how does ‘BsmtFinSF1’ (Type 1 finished square footage) vary by ‘Neighborhood?’

Subsection 4.2.2 analyzed the “interior” features of the home. This included variables such as garage size, number of rooms, kitchen quality, etc. For the most part, the rule of thumb “more is better” reigned true. Square footage for all features including ‘TotalBsmtSF’ (total basement), ‘1stFlrSF’ (1st floor), and ‘GrLivArea’ (above grade ground living area) had a positive correlation with SalePrice. Furthermore, categorical features also tended to follow a

positive correlation with SalePrice. As the quality of the feature, an example being ‘FireplaceQU’ (fireplace quality) increases, the price of the home increases. There were some exceptions but they were usually due to unusual or rare features. An example of this included the number of kitchens as most homes did not have more than one kitchen so it did not correlate with SalePrice. For continuous features, if the data points were highly skewed to one side, I tended to cut them into equal size bins. A few features had the majority of the data lie in one value (numeric) or class (categorical). An example of this is ‘LowQualFinSF’ (low quality finished square feet) where the vast majority of data was zero. I flagged and binned it as a binary variable. Last, I took all 4 variables related to the number of bathrooms and summed them together into one feature for total bathrooms.

Subsection 4.2.3 explored the Architectural and Structural aspects of the homes. ‘OverallQual’ (ratings for the overall material and finish of the house) had a very high correlation with SalePrice. I manually coded this ordinal feature. Others such as ‘HouseStyle’ (style of dwelling) did not show a positive correlation. For this and other features like it, I just created dummy variables. Features related to temporal aspects of the home, such as the year it was built (YearBuilt) tended to show a positive correlation with SalePrice. However, I wonder how much of it was due to inflation but I do not have this data. In the end, I binned the feature..

Subsection 4.2.4 explored features related to the Exterior of the home. Generally, these categorical features did not show a positive correlation with SalePrice. For the most part, I just created dummy variables. I dropped ‘MasVnrArea’ (masonry veneer area) because it was highly correlated to ‘MasVnrType’ (masonry veneer type). This is because ‘MasVnrType’ cannot be “None” in order for ‘MasVnrArea’ to have data. There were a few ordinal features such as ‘GarageQual’ (garage quality) that should have had a positive correlation with SalePrice. Interestingly, this really was not the case as most of the homes belonged to the “TA” class which signified “Typical/Average” quality. I created dummy variables in these instances. Similar to what I did for bathrooms in the “Interior” Subsection, I summed all four variables of porches together.

This made sense as they were only different due to style. I created a binary variable from ‘PoolArea’ because very few homes actually had pools, and then dropped the original variable. Subsequently, I also dropped ‘PoolQC’ (pool quality) as it also had almost zero data points and was highly dependent on ‘PoolArea.’

Subsection 4.2.5 explored the features related to the “location” of the homes, and there were only four of them. ‘MSZoning’ (zoning classification the sale) did not have a positive correlation with SalePrice. I just created dummy variables. I did the same with ‘Neighborhood.’ ‘Condition1’ and ‘Condition2’ were interesting features because they denoted the home’s proximity to various conditions such as railroads. ‘Condition2’ existed only if the home was located near more than one condition. There were 9 classes, and I clustered them logically. For example, all types of railroad classes became one railroad class, and the same was done for the different types of streets. Furthermore, I dropped ‘Condition2’ after creating a binary variable if it equalled ‘Condition1’ for a home.

Subsection 4.2.6. explored features related to the “land” around the home. For the most part, all the continuous features such as ‘LotArea’ had a positive correlation with SalePrice. Also, because the area of the house lots were clustered around 10,000 square feet, I did a quantile cut to evenly divide the data points into 8 bins. Opposite the continuous features, the categorical ones did not show a positive correlation with SalePrice. I turned those into dummy variables.

Subsection 4.2.7 only had three features and they were related to access to the property. The ‘Street’ variable only had two classes, “Gravel” and “Paved.” Unfortunately, there were only six data points in the “Gravel” class so I dropped the entire redundant feature. The ‘Alley’ feature only had two classes and I created dummy variables from them. Last, I created dummy variables from the ‘PavedDrive’ (Paved Driveway) feature as there were no discernible patterns from the three classes.

Subsection 4.2.8 explored the utilities of the home. The ‘Heating’ feature did not have a discernible pattern and was just turned into dummy variables. ‘HeatingQC’ was ordinal as the price of homes increased as the quality of the heat increased. ‘CentralAir’ was binary and ordinal as homes with AC, on average, sold higher than homes without it. I manually coded these last two ordinal features. Last, the ‘Electrical’ feature denoted the type of electrical system in the home. Overwhelmingly, most homes had standard circuit breakers, and I clustered the three different fuse classes into one “fuse” class. Then, I created dummy variables.

Subsection 4.2.9 contains the miscellaneous features. ‘MiscFeature’ pertained to homes that may have miscellaneous properties such as an elevator, a shed, or a tennis court. I ultimately dropped this feature because there was not that much information, and the little it had, belonged mostly to the “Shed” class. ‘MoSold’ and ‘YrSold’ features signified the month and year the home was sold. I created dummy variables as there was no discernible pattern. This was the same for ‘SaleType’ and ‘SaleCondition.’

Now that my EDA and feature engineering is complete for my independent variables, I’m going to analyze SalePrice. Because I’m using linear models, I want my data to be as normally distributed as possible. Section 4.3 showed the distribution of SalePrice superimposed by a normal distribution. Currently, my data is skewed with a long tail to the right. I used np.log1p to transform the data into one that more closely matched a normal distribution. Furthermore, I wanted all my data to be as normally distributed as possible. I performed a boxcox transformation for 288 features that had a skewness greater than 0.5. Last, because my data was fully prepared for modeling, I returned the “all” dataset back into their respective train and test sets.

Modeling

Similar to my strategy with EDA, my model exploration strategy was to go from broad to narrow. I started with DummyRegressor to establish a baseline model from which I could judge all my prior models. The scoring metric I used to judge my models was the RMSE, both because it is how the contest was scored, and also because it is an industry standard. Furthermore, I used a five-fold cross-validation strategy to make the results as robust as possible. DummyRegressor's test score was 0.4188. After, I performed LinearRegression and Decision Tree Regressor as they are the simple and base versions of the advanced models I would use later. Interestingly, the decision tree had a 50% reduction (0.2077) in the test RMSE and the linear regression almost 70% reduction (0.1290).

After, I explored Advanced Linear Models without optimization. Ridge gave me a test RMSE (0.1219) similar to LinearRegression. Lasso did not perform as well (0.1553) and ElasticNet the worst with a score of 0.1747. Ridge gave me hope because its untuned score performed really well. On the other hand, ElasticNet's score did not surprise me as it's algorithm was more complex than the other two models, and needed more careful optimization.

Section 5.1.4 explored Ensemble Tree Methods which are advanced algorithms built upon simple Decision Trees. There are a lot more hyperparameters so my scores were not as good as I would have liked, but they were still better than using a simple Decision Tree alone. Most notably, GradientBoost had the lowest test RMSE (0.1314) and XGBoost right after with 0.1327. Last, XGBoost is a very powerful, popular, and flexible algorithm. The most popular regression version is compatible with SciKit Learn, but I wanted to try the model using LinearRegression as its base model (XGBoost Regressor is built upon decision trees by default). This meant I had to use XGBoost's own native API, and its score was 0.2735. Moving forward, I would only use the default version of XGBoost as its score did not warrant further exploration.

In section 5.2.1, I explored feature selection via modeling. I know that there are many ways to perform feature selection such as domain knowledge or choosing features that are

highly correlated with my dependent variable. However, I chose to engineer many variables (for example, adding the polynomials) and having a powerful model pick the best features. One reason for performing feature selection, at all, is that simplicity is preferred over complexity, and the other is that reducing the “noise” or features that do not help (or worse harm) in predictive modeling should be removed.

I used XGBoost because of its powerful predictive capabilities, and it isolated 44 features for modeling. However, I did not truly know if modeling on these reduced features would give me a lower test RMSE. To test this, I took eight algorithms from my prior section and performed modeling only on the 44 features. I took the average of the scores and compared it to the average of the same models on the 315 variable dataset, and found that the reduced features outscored the full features by 2% on average. Because it reduced complexity and resulted in a better score, I used the reduced dataset as my final training dataset.

Section 5.3 had to do with model optimization or tuning the hyperparameters to get the best score. Because of the involved process, each of my chosen eight models has its own subsection. Furthermore, I kept my broad-to-narrow approach to optimization. I started with a RandomSearch because I had no intuition of the best hyperparameters for each model. I often tried to let it run more iterations to give it a better chance of finding exceptionally good hyperparameters. After, to verify or even find better hyperparameters, I ran a GridSearch. Now that RandomSearch gave me an idea of the best hyperparameters, I used GridSearch to run every possible combination of tuning, in a range, around RandomSearch’s chosen hyperparameter. This approach was not always successful, but it gave me confidence that I optimized my models as best as possible.

The first models to optimize were linear-based models. First, Linear Regression did not really have any hyperparameters so it kept its score of 0.1194 from the prior section. Ridge had a negligible increase in performance with a final RMSE of 0.1201, but then again its untuned version performed quite well. Lasso, on the other hand, had a remarkable 25% increase in

performance. It's test RMSE was 0.1200. ElasticNet, which is a hybrid of both Ridge and Lasso algorithms, had an even better 32% increase score of 0.1215. Once tuned, all my linear models scored about the same range of about 0.1200. What's interesting is how my simple Linear Regression still had the lowest RMSE of 0.1194.

The next set of models to optimize were my tree-based ensemble models. What's interesting is that tuning my Random Forest algorithm actually gave me a worse score. In that case, I stuck with my base model. Optimizing Adaboost increased my test score (0.1703) by less than 2%. This was a similar situation to GradientBoost, where optimizing my model only gave me less than 1% increase performance (0.1261). Tuning XGBoost gave me an overall increase of 3% over the untuned version earlier. It is interesting to note that this is the only model to have a lower RMSE (0.1189) than Linear Regression (0.1194).

What is interesting is that it seemed my linear models, on average, scored better than my tree-based models. This could be due to linear models, in general, being more appropriate in describing the overall trend and pattern in the dataset. Correcting skewness (Section 4.4) and transforming the data into more normal distribution could have been why my linear models performed so well. On the other hand, XGBoost did have a lower RMSE than Linear Regression (although by barely half a percent). Tree-based ensemble algorithms are also more complex, and their hyperparameters reflect that. I'm inclined to think that maybe I did not tune them correctly, but I'm also confident in my due diligence. Furthermore, using all my tree-based models and linear models together did ultimately decrease the RMSE as shown below.

Very similar to how tree-based ensemble methods use decision trees as base learners, "Stacking" uses models as base learners and outputs a prediction from them. Each of the eight prior algorithms are different with their own unique strengths and weaknesses. By inputting them into a meta-stacking algorithm, I was able to harness all their predictive prowess while covering their shortcomings.

I was able to achieve this via SciKit Learn's new "Stacking Regressor" algorithm. I never used it so I tried a few different versions of the model to see what would give me the lowest RMSE. My first iteration had a test RMSE of 0.1180 which barely performed better than XGBoost (0.1189). Still, this score was without hyperparameter tuning of the meta-algorithm. All I did was input my base models and let the algorithm run, and the baseline prediction still beat my best single model. This gave me hope that, with proper tuning, I could further decrease the RMSE. The second iteration did not include AdaBoost or Random Forest because they were my worst performing models, and my theory was that removing them would give me a better score. Interestingly, it output a higher RMSE than my prior meta-model which bolstered the theory that the meta-algorithm covers the weaknesses of the base learners.

By default, Stacking Regressor uses a version of Ridge as its final estimator, but it's hyperparameters are untuned. For my third iteration, I used all eight of my models and used my optimized Ridge as the final estimator. The test RMSE was not lower than my first meta-model. Upon my next iteration, I changed the final estimator to my optimized XGBoost. My hypothesis was that because it had the lowest RMSE of any of my models, it might perform better as the final estimator. Unfortunately, the RMSE was higher than all my prior meta-models.

Up to this point, Stacking Regressor used only my eight models as input for prediction. However, it also has an option to allow input data from both my models and the original training set to be used together. This gave me my lowest test RMSE, which resulted in a 2% increase score (0.1164) over my best single model (XGBoost: 0.1189). This became my final model.

Conclusion

This was an interesting exploration. I embarked on a journey to predict the SalePrice of homes by using 80 independent variables. I explored and cleaned the data using domain

knowledge and statistical principles bolstered by the power of computers. By the end of Feature Engineering, I had a total of 315 columns and the data was fully prepared for modeling.

I established a baseline model and found that basic Linear Regression decreased the RMSE by over 70%. I tested other untuned linear-based and tree-based models. I then compared the average score of 8 of my models against the average score of the same models, except trained on only 44 features identified by XGBoost as "most important." Because the latter models had a 2% increase in its score, I decided to model with the reduced features.

I optimized my 8 models using both Random Search and Grid Search. My best performing model XGBoost had an increase score of less than 0.5% over Linear Regression. Then, I took all my tuned models and placed them in a stacking meta-model algorithm. After final exploration, my best model had an increase of 2% over XGBoost when comparing the test set's RMSE. This equated to an overall increased score of 72% over my baseline model.

Regarding insights, my analysis confirmed what most people already know about what influences the price of homes: more is better and quality is better. The price of homes tended to be higher when the home had more rooms, bathrooms, square footage, etc. Price of homes also increased as the quality of the different aspects of the home increased. However, my model did take into account the subtle nuances of the data, and its predictions were less than $0.13 off, on average, for every $1.00 of SalePrice.

Recommendations

To fully take advantage of my work, these are the next actionable steps I would recommend:

1. Engineer data infrastructure. This would allow us to productionize the model for use at scale, increase the amount of data coming in, allow data flexibility, and access to everyone.

2. Create a data-driven culture. Teach what data is and is not so that everyone can be on the same page. Have the data team learn domain knowledge and insights from the realtors, and have the realtors be open to newly discovered patterns, trends, and factors.
3. Identify goals and teams. What are the most important questions or problems that your data team can investigate? Are your teams cross-functional or separate?

The goal is to increase revenue through the use of data. By setting-up the proper infrastructure and instilling a data culture, we are investing in a future where any and all data can be used to quickly capitalize on revenue-increasing opportunities. Furthermore, identifying goals and creating the proper team organization will help smooth this transition. In a world where the use of data is becoming widespread and adopted, AREC can position themselves at the head of the curve.

Areas for Further Exploration

Here are some recommendations for further areas of improvement and exploration:

1. Compare Time Series Analysis with inflation data. There seemed to be a positive correlation of the price of homes against the year it was built. However, the data was not adjusted for inflation so that is something for further exploration.
2. Feature Selection via Ridge or other models. I used XGBoost because of its predictive prowess, but maybe Ridge or other models could identify more important features that would ultimately increase accuracy.

3. When performing Feature Engineering, I often binned continuous features which replaced the original variable. Instead, I would use quantile or adaptive binning which would evenly distribute the data into evenly sized bins. Furthermore, I would not replace the original feature with these bins but add them as their own separate features.
4. Experiment with Bayesian Hyperparameter Tuning. GridSearch is computationally expensive and RandomSearch is a game of chance. Bayesian Optimization is essentially a “smart” RandomSearch that can identify areas that can improve the predictive accuracy.
5. Stacking Regression on all the features. The score increased when I enabled the algorithm to use both my base estimators and my training data as input. Maybe the test RMSE would be lower if I used all the features instead of the reduced dataset.
6. Auto Modeling and Tuning. In Section 6.4, I showed my attempt at using the TPOT automated machine learning tool. The meta-model it chose was very convoluted and had its own API. I would like to explore this more and see how well it ultimately scores.
7. Hypothesis Testing for Linear Regression vs Final Stacking Model. Stacking Regressor was my best model. However, it is more complex and computationally expensive than Linear Regression. I’d like to perform a formal Hypothesis Test to see if its average RMSE is statistically significant than Linear Regression because if it is not, then the latter is cheaper and faster to move into production.