Mirnes Salkic
AMOD 5410H
April 2018

Machine Learning on Ames Housing Data set

Abstract

This project is an exploration of the Ames Housing data set compiled by Dean De Cock. It heavily focuses on preprocessing which is a vital step in producing models that achieve high accuracy. Besides exploring different preprocessing techniques, the project compares the performance of regression techniques available in a statistician's toolkit such as Multiple Linear regression, Lasso regression, and Ridge regression as well as advanced ensemble techniques more commonly used by data scientists namely Random Forest and XGBoost Regressors. The goal was to produce a model with good accuracy compared to top Kaggle participants –the top 18 score achieves a RMSE (root mean squared error) of 0.11 which translates to an r-squared of 0.92. The goal has been achieved as the best model - Ridge Regression - has a RMSE of 0.113 and an r-squared of 0.910 on the test set. Lastly, identifying the most important features for predicting the Sale Price of the house was an important goal. The most important features detected by Ridge Regression include: Total Area, Sale Type, Overall Quality, Neighborhood, and Sale Condition among others. I highly recommend taking a close look at the iPython notebook as it reflects my efforts to a great extent.

Data set source:
https://www.kaggle.com/c/house-prices-advanced-regression-techniques#description

Mirnes Salkic
AMOD 5410H
April 2018

1. Introduction

The data set used in this project describes the sale of individual residential property in Ames, Iowa from 2006 to 2010. It has 2919 samples and 79 explanatory variables. The target variable is the SalePrice and the aim of the project is to be able to predict the SalePrice on data the model has not been trained on. The data is beforehand split into train and test sets. However, the test set supplied does not have the SalePrice, thus we cannot check how accurate our model is unless the results were submitted to Kaggle. For the purposes of this project I decided to further split the train set into another train and test set as it would allow me to easily validate my models without having to continuously submit to Kaggle. Another reason for this action is that there is a limit on the number of submissions per day for Kaggle. In principle, the accuracy obtained from my models would be even higher had I validated my models by uploading the results to Kaggle as I would be able to use more data for training my models.

The 79 features in the data set focus on the quality and quantity of many physical attributes of a property. Many of the features are the information a typical home buyer considers when buying a property such as: When was it built? How big is the lot? How many bathrooms are there? There are about 20 continuous features referring to different areas of the property (e.g. pool area, basement area, porch area etc.,). The 14 discrete variables present counts of different items in the house such as number of bathrooms, bedrooms, kitchens etc. In terms of categorical features there are 23 nominal (e.g. neighborhood, type of dwelling) and 23 ordinal features (e.g. overall quality of the house).

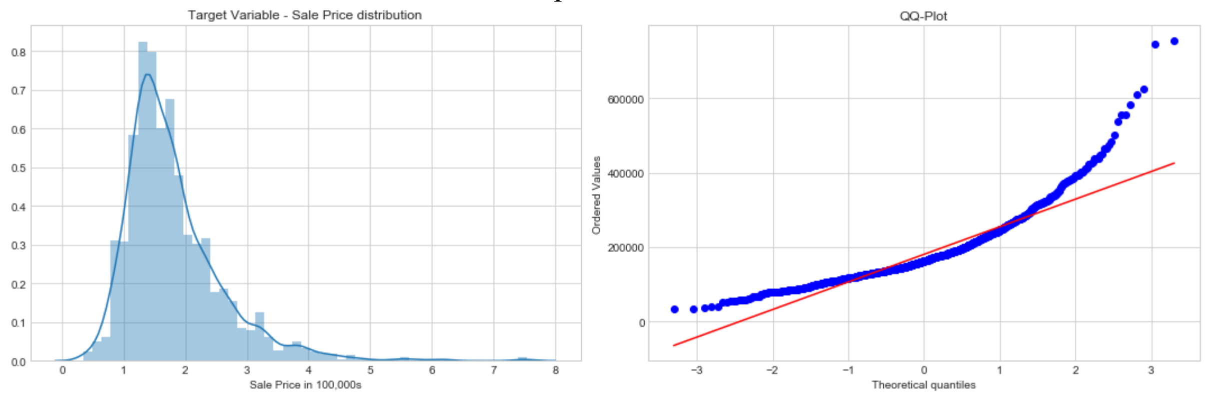
Most of the time working on this project, roughly 85%, went into understanding the various features, thinking of different feature engineering approaches, filling in missing values, removing highly correlated features to avoid multicolinearity, and removing outliers. The remaining 15% of the time was spent on model building and analysis.

The remaining part of this paper will describe my thought process and efforts in data preparation, model building, and discussion of the results. About 95% of the code was written by myself, with the exception of a few graphs that analyze the residuals of the models and the importance of the coefficients. The ideas and partial code were taken from Sebastian Rashka's *Python Machine Learning book.*

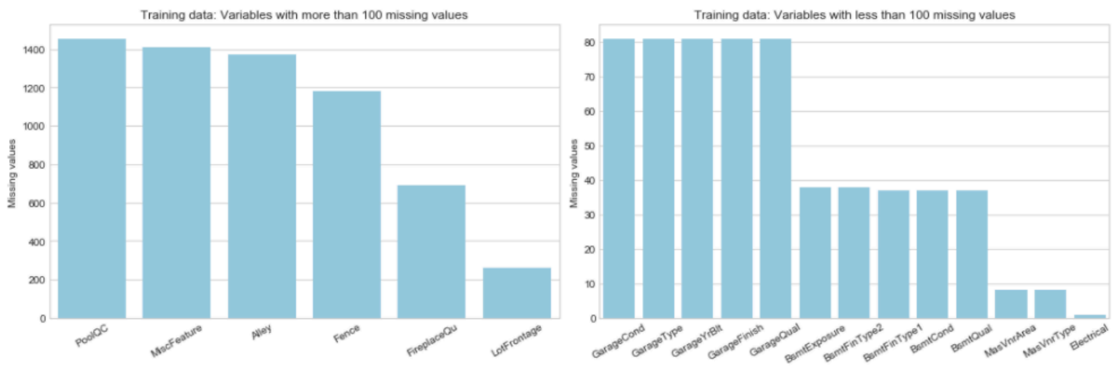
2. Preprocessing

2.1. The Target Variable

The target variable is right skewed, which in principle should not matter for linear regression models, however it is recommended in Kaggle's evaluation section, to log transform this variable so that "errors in predicting expensive houses and cheap prices will affect the result equally". Thus, I logged transformed this variable at the end.



2.2. Missing Values/NaNs



As it can be seen from the graphs above there are many variables in the train set that have missing values. Furthermore, a similar graph in the my Jupyter notebook (section 2.6.2) reveals that the test data has even more features that have missing values (i.e.33 features). It was important for me to understand why the values are missing and discover the best ways to fill them in.

After careful study of the data set instructions provided for the competition, I have discovered that many of the NaN values are not missing but in many cases represent the value of 0 in case of numeric variables and “None” in case of categorical variables. For example, the missing values in pool quality variable means “No pool”.

On the other hand, continuous ratio variables like LotFrontage (linear feet of street connected to property) has missing values where NaN does not represent 0. Thus, I had to think of a way to impute those! Using the mean was not a good option as the mean is highly affected by outliers; thus, I decided to use the median which is a more robust statistic. Furthermore, I realized that the LotFrontage might be dependent on the neighborhood of a house, and hence after visually exploring the relationship between the two I decided to impute the missing values of LotFrontage conditionally upon the Neighborhood (for the graph see section 2.6.2. of the Jupyter notebook).

During this process of inspecting each of the 80 variables in the train and test set I noticed that some levels in some categorical variables were present in the train set but not in the test set. Hence, it was important to remove those variables before model building in order to avoid overfitting. I will explain this point in greater detail later.

2.3. Converting between different types of variables

During the process of data exploration, I noted that some of the variables that are nominal in nature were encoded as numerical. For example, MSSubClass, a variable that describes the type of dwelling involved in the sale, was encoded with numbers 20, 30, 40, 45, etc., where each number represents a type od dwelling. After reading the data description, which was of invaluable help, I identified and encoded those numeric variables as nominal. Surely, this was not the only variable converted – more examples can be seen in the iPython notebook section 2.7.

In addition, the variable MoSold which was encoded as ordinal was casted to a nominal variable. Here I reasoned that February is not more important than January and thus does not deserve a higher value, instead treating it as a category which stresses distinctness rather than both distinctness and order would make more sense. Indeed, I built the models treating the MoSold variable in both ways and concluded that when used as a nominal type it gave better results.

On the other hand, many of the variables were encoded as nominal instead of ordinal. For example, LandSlope (slope of the property) has three levels: Severe slope, Moderate slope, and Severe slope were label encoded as 1, 2, and 3 respectively. More than 20 such features were identified as such and were casted into their appropriate data type.

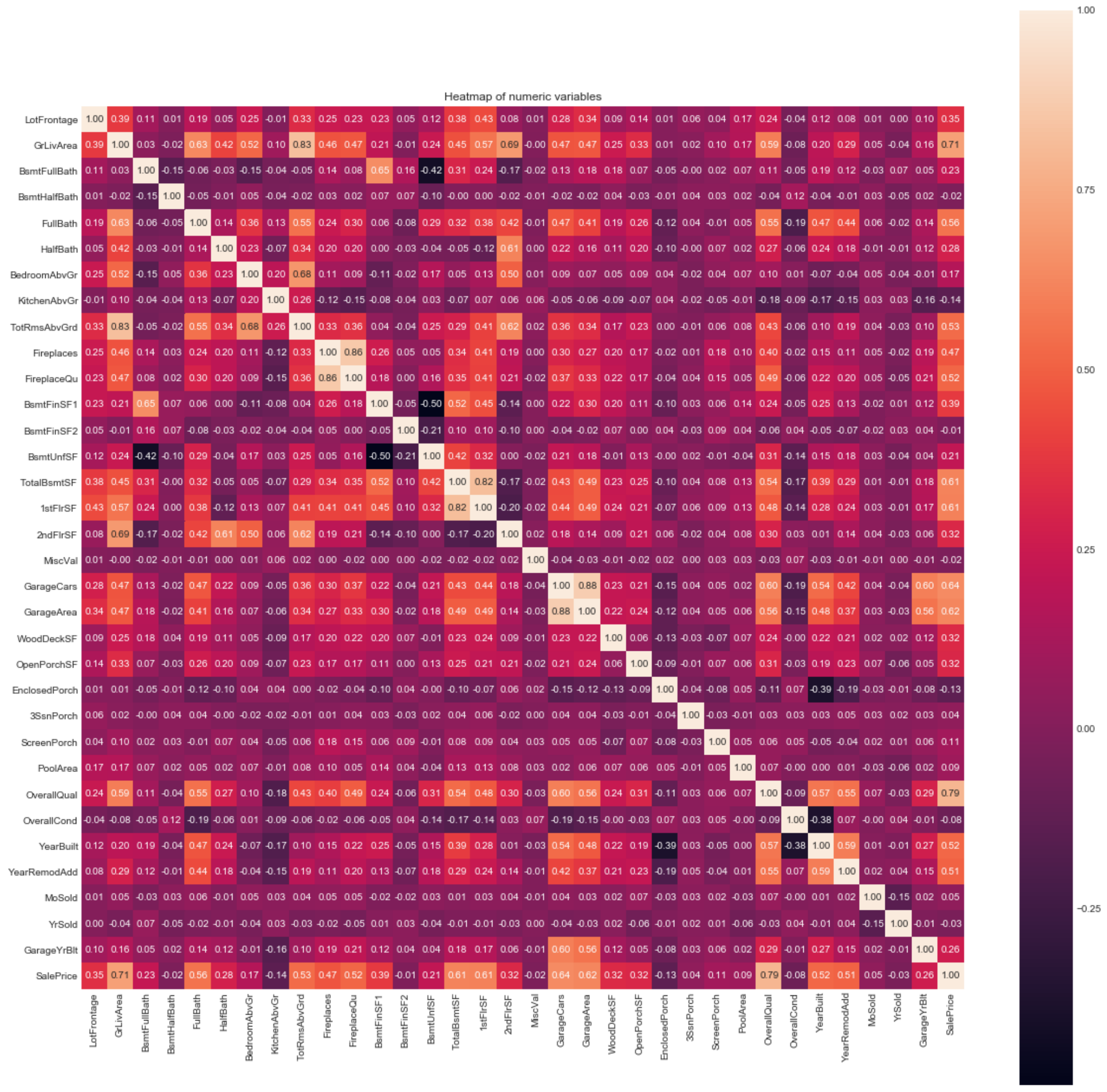
2.4. Multicolinearity/Relationship with the Target

One important assumption in linear regression is that residuals need to be normally distributed. This assumption could be broken if there is a high correlation between predictor variables. If two predictor variables are highly correlated it implies that they carry similar information, and hence one of them should be dropped. There is no better way to inspect correlation between variables than using a heatmap.

Given the large number of attributes it is not surprising that the heatmap is big. In fact, not all numeric variables are even shown in the heatmap below. Because of this, I had to create a few smaller heatmaps grouping variables that I suspect to have some sort of multicolinearity together on the same graph. For example, there were 11 basement related variables and I would add a separate plot for those as I expect some of them to be correlated. Thus, as it can be seen from the notebook, heatmaps were created multiple times for the initial diagnostics and observations and later again after the feature engineering was completed.

In the heatmap shown below it can easily be noted that some variables are relatively strongly correlated with each other:
GarageArea and GarageCars - 0.88.
Fireplace and FireplaceQu - 0.86.
GrLivArea and TotRmsAbvGrd - 0.83.
1stFlSF and TotalBsmtSF - 0.82.
GrLivArea and 2ndFlSF - 0.69.

Mirnes Salkic
AMOD 5410H
April 2018



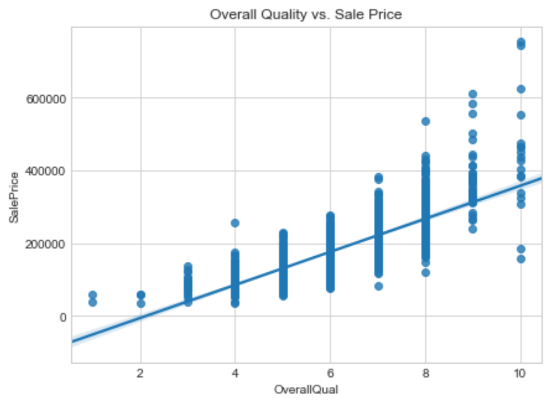
The correlations make sense intuitively:

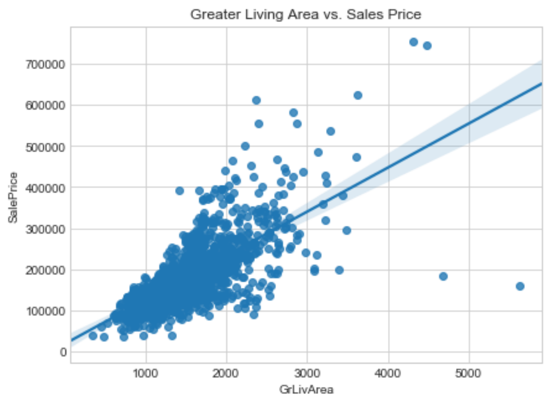
- If more cars can fit into a garage it means that the area is bigger.
- If there are more total rooms above the ground floor it means that the living area is larger.
- The greater the first floor area the greater the total basement area.
- The greater the second floor area the greater the living area.

Once those highly correlated variables were identified the one which had a smaller correlation in respect with the SalePrice variable was dropped. For instance, Garage Cars has a correlation of 0.64 and Garage Area has a correlation of 0.62 in respect with the SalePrice. Thus, Garage Area was dropped. Surely, this is just to illustrate the logic used and is explained for one particular example. For a complete and detailed exploration of multicolinearity please see the notebook which also includes brief discussion notes.

2.5. Features highly correlated with SalePrice

As part of the preprocessing I was interested in learning what features are highly correlated with the SalePrice – the response variable. I would expect the highly correlated features to be good predictors of the SalePrice and would expect them to have high coefficients once the models get built.





Hence, I found that overall quality of a house is the best predictor of the SalePrice with a correlation coefficient of 0.79. Furthermore, greater living area (above ground living area in square feet) has the second highest correlation coefficient of 0.71 with respect to SalePrice.

We can notice easily from the graphs the the relationships are relatively linear. For example, a greater living area generally leads to a greater SalePrice. However, in the case of greater living area, there seem to be outliers. Most notably, there are four data points in which the area is above 4000 square feet that strike one's attention. Those four data points represent houses that are priced both unusually low and unusually high given the size of the area. It is wise to remove those outliers as they may cause overfitting and might even be a potential cause for non-normally distributed residuals. Those were removed.

Other numeric variables that are highly correlated with the SalePrice include exterior quality (0.68), kitchen quality (0.66), garage cars – number of cars that fit into a garage (0.64). We would expect those variables have among the highest coefficients once the models get trained.

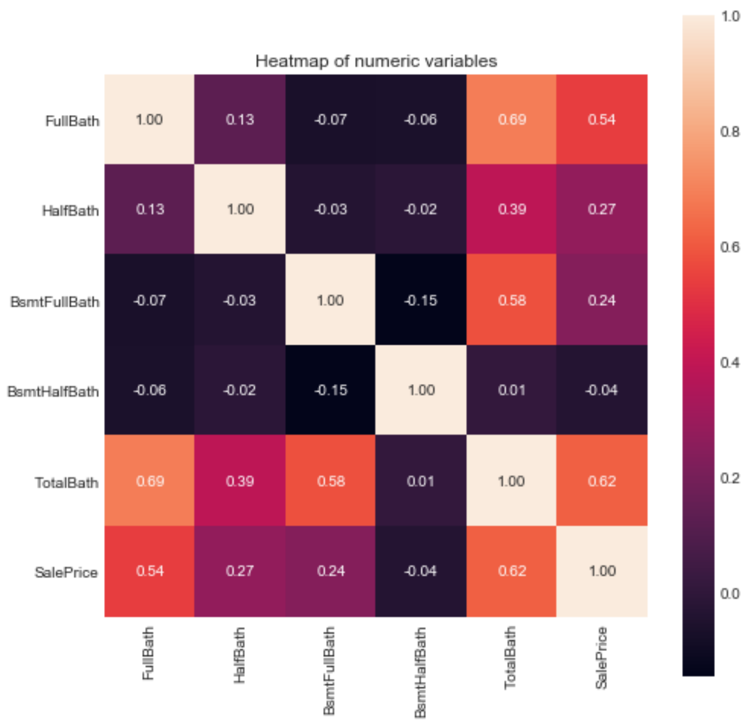
There are also nominal variables that appear to be correlated with the SalePrice, after visual inspection of every single graph (please see the python notebook section 2.7. Visualizing Nominal Variables), I concluded that neighborhood has a strong correlation with price. Additionally, the SalePrice tends to be higher for houses that have central air as opposed to those that do not.

All of the 79 variables are plotted in the jupyter notebook for closer inspection.

2.6. Feature Engineering

The visual examination of all the variables was a very important step as it allowed me to think about potential feature engineering techniques. The pair plot, made me realize that there were too many “Area” attributes. For example, there is a variable- greater living area and another - total basement area which share complementary information. Intuitively, both of them are important, but my assumption was that a buyer typically would think in terms of the total area rather than on individual ones. Thus, combining these two features into one called totalSF was my choice. After doing this step, I checked for multicolinearity again using a heat map, but more importantly I was interested in knowing how this newly created feature correlates with the SalePrice. It turns out that the newly created feature “totalSF” has a higher correlation with SalePrice (0.80) than greater living area (0.70) and total basement area (0.63) individually. This is how I checked whether my feature engineering was justified. Surely, I tried various combinations for combining various features and left only those that seem to meet the aforementioned criteria.

There were a total of four variables that quantify the number of bathrooms in a house. Namely, basement full bath, basement half bath, full bath, and half bath. It was natural to combine those four features into one which tells us the total number of bathrooms in a house. To get a combined number of bathrooms I multiplied the the fullbathroom variables by 1 and half bathroom variables by 0.5 and added them together. This newly combined feature again had a higher correlation with the SalePrice (0.54) compared to the components. See heatmap below.



Similarly, four porch related variables, all describing the areas of the porch given different criteria (e.g. open porch, enclosed porch etc.) were combined into a single variable – total porch SF.

There were a few Age related variables – e.g. year the house was built, sold. They have a larger range compared to most other variables. I thought that one way to reduce the range, which is done to avoid one feature to dominate over the other in the feature space, was subtract each year from 2010 (this is when the data was collected). For more on feature preprocessing the interested reader may take a closer look at the jupyter notebook section 3.

2.7. Binning and Dropping Irrelevant Variables

The preprocessing of the data allowed me to see the distributions of all the variables. Some of nominal variables such as Neighborhood and MoSold had many levels. I realized that after one-hot-encoding each of these levels would become a new variable which can be problematic as too many features may lead to the curse of dimensionality. My aim was to see if some of the nominal variables with many levels could be binned. In the case of the neighborhood variable I realized that some of the neighborhoods had houses with similar SalePrices and thus I grouped the neighborhood variable into five levels based on the SalePrice. Hence instead of 25 new variables we would get only 5.
To validate my technique used, I created models with and without binning to see if there was a significant difference in the R^2 and indeed there was. The binned variables gave better results. Similarly, the MoSold variable was binned into seasons.

After the necessary preprocessing was done there were variables that were no longer needed. For example, variables that were identified to have a high correlation with other predictor variables were removed based on the logic already discussed. Similarly, after feature engineering (e.g. combining multiple feature into one) the individual variables that comprised the new aggregated variable were dropped. The preprocessing step reduced the number of variables to about 60.

2.8. One-hot- encoding

For machine learning purposes it is important to have all the variables in a numeric form. Thus, some machine learning algorithms do not work with non-numerical attributes and thus an important part of preprocessing is to create a separate feature for each of the levels in a nominal attribute where a 1 indicates the presence of the level and 0 the absence. This certainly adds many new features and in my case I ended up with 206 variables in the train set and 192 variables in the test set. Since all the preprocessing steps were done in parallel on both the train and test set one might wonder why is there a difference in the number of variables? The has to do with the number of levels in present in the nominal features. I have mentioned earlier that some levels of a given nominal variable were present in the train set, but not in the test set. This was recognized thanks to the thorough examination of each variable in the train and test set.

A case in points is the -type of heating - variable, which has the following levels in the train set: (“Floor”, “GasA”, “GasW”, Grav”, “OthW”, “Wall”), but only these levels in the test set: (“GasA”, “GasW”, Grav”, “Wall”). Thus, it is easy to see that after one-hot-encoding the train set will have 6 new variables while the test set only 4. After thinking about this issue, I realized that I should drop the variables that do not exist in the test set, because it may cause overfitting.

3. Splitting the Data in Train and Test

Even though the data was already split into train and test, I decided to split my train data again into train and test. I did this because it was much easier for me to validate my models, since the given test set did not have actual SalePrices and for validation purposes I would have had to submit my scores to Kaggle each time. Moreover, the daily limit for score submission for Kaggle was 5 and I could only submit predictions based on one model. My goal was to compare different regression techniques and machine learning techniques and thus needed to know how they compared to each other. I believe that this logic justifies my action.

4. Scaling numeric variables and Creating a Scoring function

Regression and generally machine learning algorithms require numerical variables to be scaled. This is necessary to avoid the variable with a higher range to dominate over the variable with a smaller range. Thus, I had identified all the numeric – in this case- continuous ratio variables and scaled them.

Additionally, the results on the data set on Kaggle are evaluated based on the Root-Mean-Squared-Error (RMSE) between the logarithm of the predicted value and the logarithm of the observed price. Since sklearn does not have this function, I had to design a custom one. This was relatively easy (Section 6.1).

5. Modeling

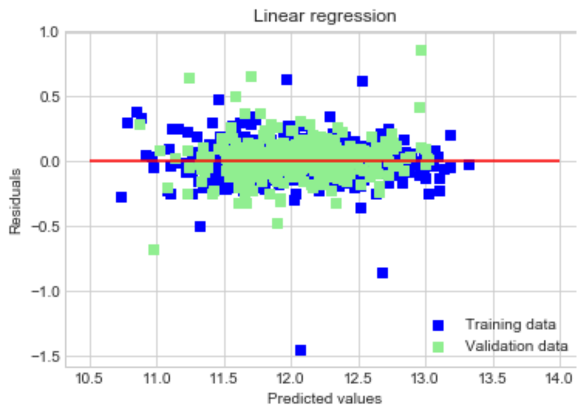
5.1. Linear Regression

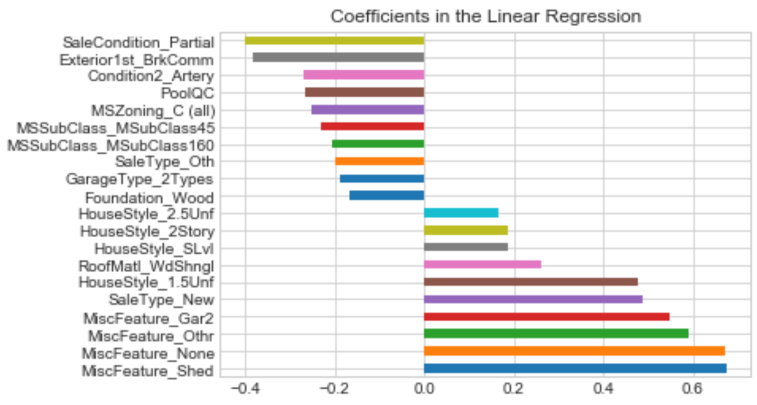
The first and simplest models of all I have run is the linear regression model. The scores are summarized below:

Linear regression results	
RMSE on Train	0.1097
RMSE on Test	0.1332
R^2 on Train	0.9265
R^2 on Test	0.8803

The R^2 test indicates how well the predictor variables are able to explain the variance of the SalePrice (response) variable. In the case of linear regression, we can see that the predictor variables can explain about 88% of the variance in the test data. It is a good result; however, we can see that the results on train data are better than on test data which could imply that the model is overfitting. Using some regularization might improve the results as we will see shortly.

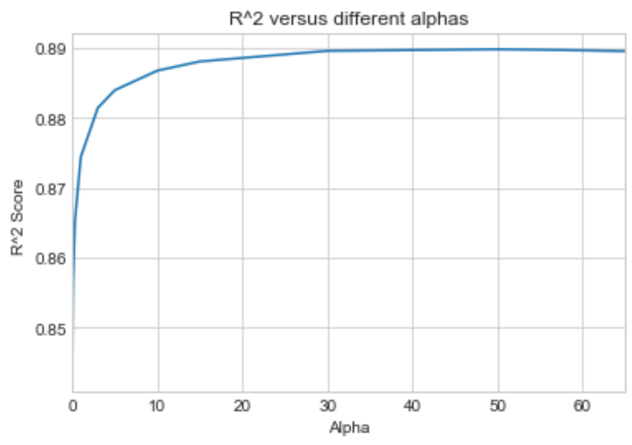
The residual plot versus the predicted shows that the residuals are normally distributed. It seems like that there are a few points left that might be outliers, but generally it looks good. The most important variables are summarized in the Coefficients in the Linear Regression plot. The most important coefficients identified by the Linear Regression surprised me as they do not include those I identified during the preprocessing (e.g. those that highly correlate with the SalePrice variable).





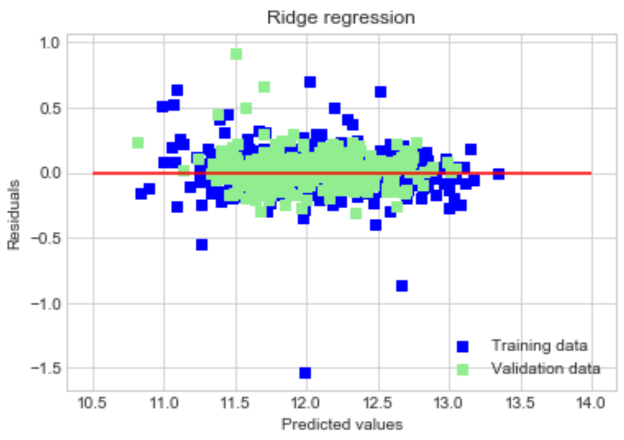
5.2. Ridge Regression

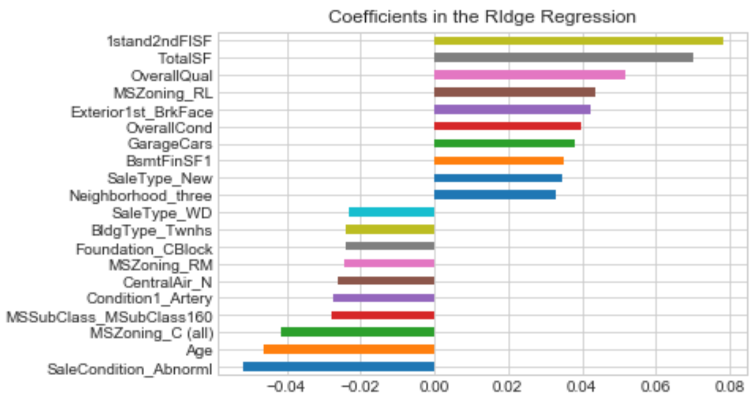
Since it appears that our initial Multiple Linear Regression model was overfitting, using some regularization that would penalize the model coefficients to prevent overfitting might be a path to success. In Ridge Regression, the coefficients are not only chosen to predict well on the training data well but also to fit an additional constraint. It ensures that the weights or the coefficients remain as small as possible while still predicting well. In other words, each feature has as little effect on the SalePrice variable as possible (i.e. small slope/coefficient) while still the model as a whole is able to predict the the SalePrice well. This is known as L2 regularization and is often used to prevent overfitting. In theory, Ridge Regression makes a trade off between the simplicity of a model (e.g. keeping the coefficient as low as possible, near zero) and the performance on the training set. This trade off is quantified in a parameter called alpha (Muller et. all). It was necessary to tune this parameter so that we achieve the best results. The graph below summarizes the tuning process of this parameter.



The model with the highest R^2 had an alpha of 33. After running the model on train and test set with the tuned parameter we obtain the following results:

Ridge regression results	
RMSE on Train	0.1186
RMSE on Test	0.1150
R^2 on Train	0.9140
R^2 on Test	0.9108

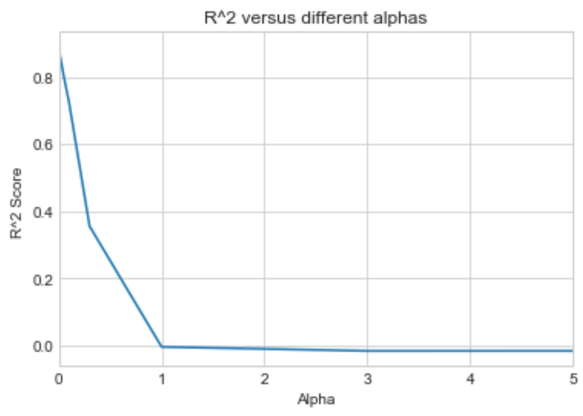




The residuals seem to be normally distributed with a few potential outliers. It is interesting that the most important feature is 1stand2ndFlSF which is a new feature created in the Feature Engineering as a combination of two other features. Other features that have high importance include total area, overall quality, MSZoning_RL (which is a residential low density housing). The R^2 on the test data is higher than in the linear regression making it a better model.

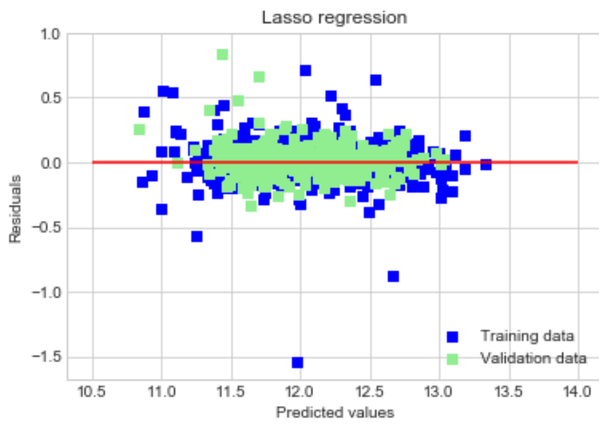
5.3. Lasso Regression

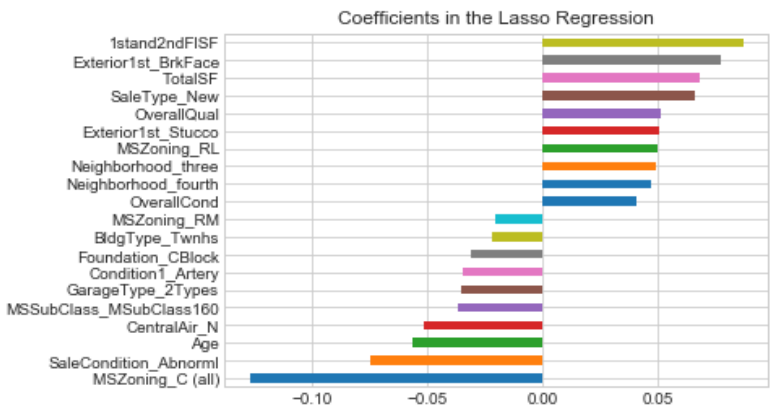
Lasso is an alternative to Ridge regression when it comes to regularization. Lasso is similar in a way to Ridge regression as it also penalizes high model coefficients, but it does so in a different way. Lasso uses the so called "L1" regularization and as a consequence sets some coefficients to exactly 0. Lasso is also often used as a model for feature selection. (Mueller *et al.)*As in Ridge regression I had to fine tune to get the best alpha.



The best alpha was at 0.007.

Lasso regression results	
RMSE on Train	0.1182
RMSE on Test	0.1136
R^2 on Train	0.9146
R^2 on Test	0.9129



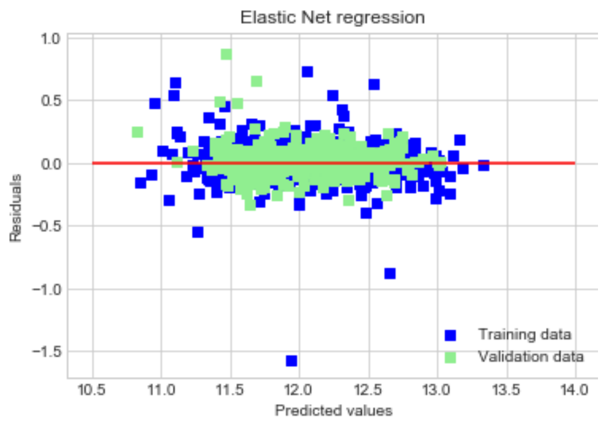


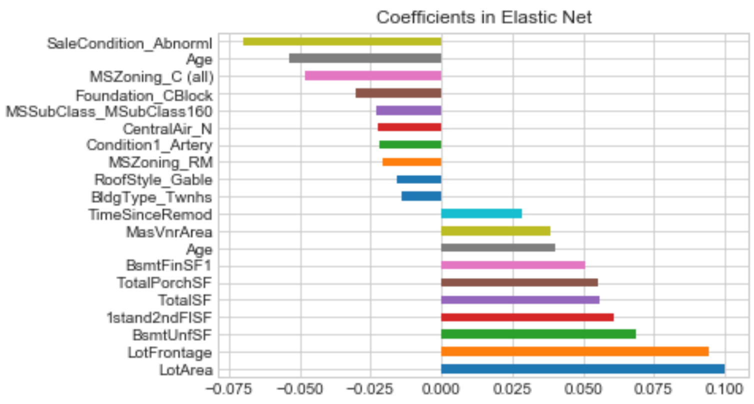
The results are almost identical to those obtained from Ridge Regression.

5.4. Elastic Net Regression

Elastic Net Regression is the middle ground between Lasso and Ridge Regression. That is, the regularization term is a simple mix of both Ridge and Lasso's regularization terms, and we can fine tune the model to give us the best mix ratio, namely r. When r = 0, Elastic Net is equivalent to Ridge regression, and when r=1, it is equivalent to Lasso (Mueller *et al*). The best tuned parameters are: r = 0.1 and alpha = 0.009. The results are slightly better on the test set which might mean that the tuned Elastic Net is slightly underfitting the data. This is the second best model.

Elastic Net regression results	
RMSE on Train	0.1205
RMSE on Test	0.1145
R^2 on Train	0.9112
R^2 on Test	0.9114

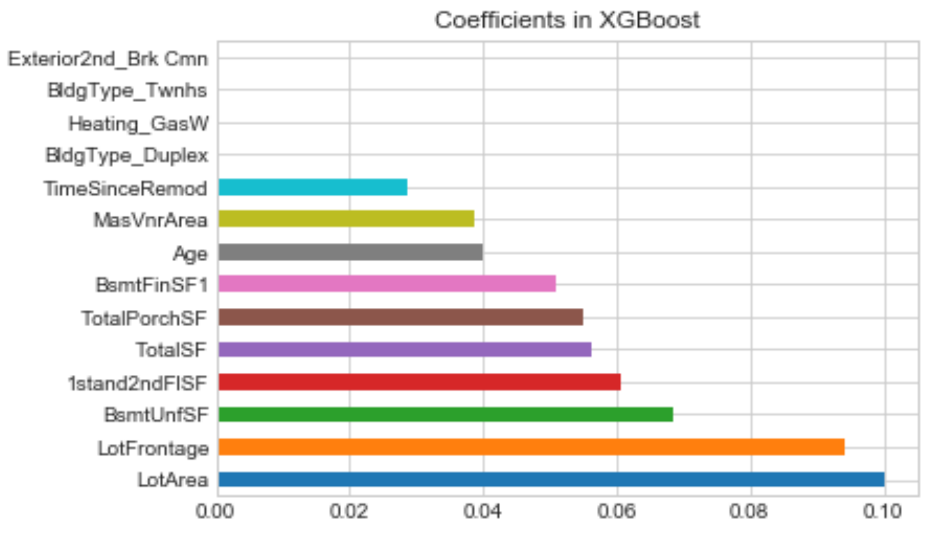




5.5. XG Boost Regressor

XG Boost is the most advanced algorithm I have used. Boosting, which is an ensemble method combines the results of many "weak" classifiers. In particular, the "weak" classifiers are decision tree regressors. In principle, the models are added on top of each other where the errors of the previous model are corrected by the next model. This process continues until the data in the train set is accurately predicted. In gradient boosting, as opposed to non-gradient boosting, different weights are not assigned after each iteration to each classifier, but rather the method fits the new model to new residuals of the previous prediction and then minimizes the loss. That is the, the model is being updated using gradient descent. [KDNuggets].

XGBoost Regressor results	
RMSE on Train	0.0026
RMSE on Test	0.1233
R^2 on Train	0.9999
R^2 on Test	0.8974

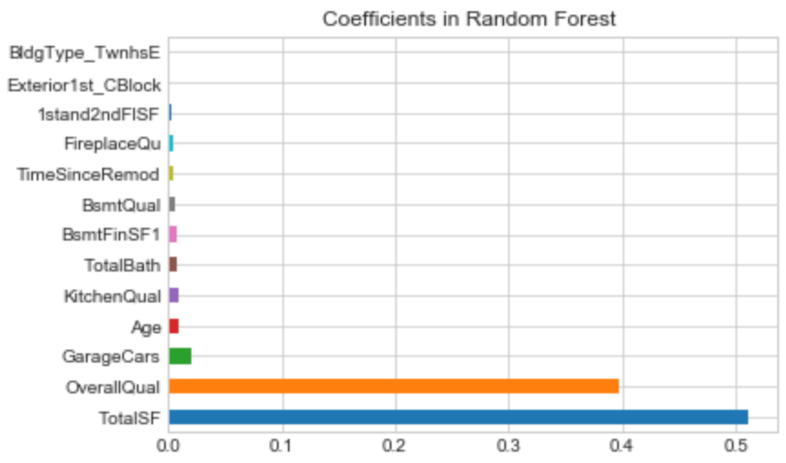


My fine tuned XGBoost Regressor is clearly overfitting. It does extremely well on the train set, but does not generalize well on the test set. It still achieves a better score than a simple Linear Regression model. However, I would need to further explore ways to reduce the complexity of the model. I am assured that the prediction error on test set can be improved with more cautious tuning.

5.6. Random Forest

Random Forest is a nice contrast to the XG Boost algorithm, it is based on bagging (averaging) rather than boosting. In simple terms, it trains many decision tree regressors on random subsets of the features and then averages the predictions. It is another ensemble learning algorithm.

Random Forest Regressor results	
RMSE on Train	0.1674
RMSE on Test	0.1628
R^2 on Train	0.8287
R^2 on Test	0.8212



The very close results on the train and test data seem to indicate that the model may be underfitting. I tried tuning tuning the parameters is a few different ways but I could not improve the result significantly. Random Forest performed the worst relative to other classifiers. The important coefficients identified by the classifier are interesting nevertheless, they all seem to be the attributes a typical person would think about when buying a house – no surprises!

6. Conclusion

Overall, this project has been a long and worthwhile journey. Many of the preprocessing techniques were utilized including imputation of missing values, feature engineering, feature transformation, binning, outlier detection, scaling. This part absorbed most of the time allocated for this project. For more thorough details, visualizations, and logic used for this project the interested reader should look at the iPython notebook. Among the many different simple and advanced regression models used in addition to some advanced ensemble methods from machine learning I conclude that Ridge Regression gave the best results with an R^2 of 0.9108 and RMSE of 0.1150 on the test set. This results puts me in the top 18 submission on Kaggle. My future goal is to continue working on this project and make a submission to Kaggle as I suspect that better fine tuning of the XG boost repressor may give even better results. Finally, we have learned that the most important predictors of the SalePrice of a house, according to our best model (Ridge Regression), are total area, overall quality of the house, whether the house is situated in residential low density zone, age of the house among others.

-word count 4392-

7. References:

KDNuggts, accessed April 2018.
https://www.kdnuggets.com/2017/10/xgboost-top-machine-learning-method-kaggle-explained.html

Muller, Andreas and Guido Sarah. *Introduction to Machine Learning with Python: A Guide for Data Scientists.* First edition. 2016.