House price prediction - Ames housing dataset

Naveenkumar Ramaraju
Indiana Univeristy, Bloomington
nramaraj@iu.edu

Saketh Pilli
Indiana Univeristy, Bloomington
sakpilli@indiana.edu

Keshav Sridhar
Indiana Univeristy, Bloomington
ksridhar@iu.edu

December 15, 2016
All the work herein is solely ours.

Abstract

The problem of house pricing presents an interesting opportunity for us. Everybody at some point in their lives would want to purchase or build their own house. And the people looking to purchase their dream house usually look for certain things like the amenities provided, the look and feel of the house, proximity to facilities etc. On the other hand, the real estate market is constantly looking to sell houses. The underlying opportunity that encompasses both these issues is the "value estimation of a house". So, the basic problem we address using data mining is "How much is a house worth?".

To tackle this challenge, we look at various underlying features and test the impact of feature engineering on the prediction of house prices using data mining techniques and also to try and look for certain interesting relationships within the housing data. The Ames housing dataset has around 80 features pertaining to houses from the Ames region in Iowa. We predict the household prices using a boosted tree based model and a multiple linear regression model with regularization techniques. Combining the results from both the techniques, we predict the price of a house.

1 Introduction

House price evaluation is an often cited scenario in statistics. It is an intriguing problem in the sense that the price of a house is dependant on so many factors some of which on first glance could be construed as mundane. This fact opens up the possibility for feature engineering by way of which we can hope to extract some interesting insights from the observed data and better predict the sale price.

One of the more frequently used techniques to evaluate house prices is multivariate linear regression. In this technique, we try to model the dependant variable(sale price of a house) as a function of several independant variables(predictors or features about the house). Once the model has learned the coefficients, we can then proceed with the predictions. For this type of model, we look to minimize the sum of squares error. We capture this as the `RootMeanSquaredError` between the logarithm of the observed price and the predicted price. As sometimes the features are multicollinear, and some features might render other features redundant, we perform our regression with both L-1(**Lasso**)[1] and L-2(**Ridge**)[2] regularization to see if the model performs better.

We also take a look at a different approach which utilises a boosted tree based model, of which some implementations are steadily gaining popularity for both classification and regression. The advantage of this technique is that it provides robustness of trees against the scaling of inputs, i.e we do not necessarily need to normalize the data and it can learn higher order interactions by using many trees. The model used in this effort is **XGBoost**[3] which is a variant of a gradient boosted tree.

2 Background

2.1 Data

The Ames housing dataset contains 80 features including the *id* feature and the target variable *SalePrice*. The summary of data types is as follows:

Datatype	# of features
Nominal	23
Ordinal	23
Discrete	14
Continuous	20

The majority of the continuous variables are the area columns particular to a house in square footage. The nominal attributes includes neighborhood, types of dwelling, garages and so on. The ordinal attributes detail the level of certain items such as PoolQuality, HeatingQC and so on.

2.2 Notation

Following are the notations used in the algorithms involved for this project effort:
• $f(X)$ implies that a function f is applied to a set of variables/features X
• MSE is the mean squared error
• $\hat{\beta}$ is the estimated value of our function/parameter β
• x^T refers to the transpose of the input vector/values x
• RSS is the residual sum of squares and $PRSS$ is the penalized residual sum of squares
• λ is the tuning parameter
• σ is the standard deviation and σ^2 is the variance

2.3 Algorithm

To predict the SalePrice of a given house in the dataset, we utilise three different models namely,

1. Regression with Ridge regularization
2. Regression with Least absolute shrinkage and selection operator(LASSO)
3. Gradient boosted tree

In the general technique of multivariate linear regression we represent our model as,

$$y = f(x) + \epsilon,\ \epsilon \in (0, \sigma^2)$$

with our regression function as $\hat{f}(x) = x^T\hat{\beta}$
and the error estimate as mean squared error MSE,

$$MSE(\hat{\beta}) = E||\hat{\beta} - \beta||^2$$

The problem is that even if our $\hat{f}(x)$ fits our data well, we are not sure that it is a correct representation of future data points, i.e there exists some kind of prediction error. As our model complexity grows, the coefficients estimates could suffer from high variance. By introducing some kind of bias to our estimates could result in a decrease in variance and by that our prediction error could go down as well.
One of the techniques to handle the high variance among coefficients is by imposing a **Ridge**[4] constraint,

$$min \sum_{i=1}^{n}(y_i - \beta^T x_i)^2 \text{ s.t, } \sum_{j=1}^{p}\beta_j^2 \leq t$$

The new penalized sum of squares can be written as,

$$PRSS(\beta)_{l_2} = \sum_{i=1}^{n}(y_i - x_i^T\beta)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

Here λ is the shrinkage parameter which controls the size of the coefficients, as $\lambda \to 0$, we move closer to the least squares solution and as $\lambda \to \infty$, we get $\hat{\beta}_{\lambda=\infty} = 0$.
In this way, the weights in ridge regression are penalised for growing too large.
In ridge regression model it continuously shrinks the coefficients and due to this, even though the model could become stable, it is much harder to interpret as we do not set any coefficients to 0.
To combat this, we use the **LASSO** model where it shrinks some coefficients just as in ridge but more importantly, it also sets other coefficients to 0.
The model can be represented as,

$$min(y - X\beta)^T(y - X\beta) \text{ s.t, } \sum_{j=1}^{p}|\beta_j| \leq t$$

We can rewrite the penalized least squares solution as,
$PRSS(\beta)_{l_1} = \sum_{i=1}^{n}(y_i - x_i^T\beta)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$
A large enough λ value will set some of the coefficients to 0.
For both **Ridge** and **Lasso** we assume that the data is standardised(mean 0 and unit variance) and centered.

Now, moving from one end of the spectrum to the other, we arrive at utilising a gradient boosting

machine[4](**GBT**) to predict our target variable. Gradient boosting is an ensemble technique which builds upon many weak classifiers, which in this case are decision trees, in which it optimizes a loss function and grows the trees in a stepwise manner. In a general form we can represent our gradient boosted model as,

$$\hat{y_i} = \sum_{k=1}^{K} f_k(x_i),\ f_k \in F$$

where F is the space of functions containing all the regression trees. The parameters of the model are the structure of each tree and the score achieved at each leaf.
Instead of learning weights for coefficients as in lasso or ridge regression, here we are learning functions or trees. To learn functions we need an objective function given as,

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y_i}) + \sum_{k=1}^{K} \omega(f_k)$$

where the first operand signifies the training loss(l) and the second operand(ω) signifies the complexity of the trees.
The generic gradient boosting algorithm is given as follows,

Algorithm 1 Gradient_boost

$F_0(x) = \text{argmin}_p \sum_{i=1}^{N} L(y_i, p)$
for $m \leftarrow 1..M$ **do**
  $\hat{y_i} = -[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}]_{F(x)=F_{m-1}(x)},\ i = 1, N$
  $a_m = \text{argmin}_{a,\beta} \sum_{i=1}^{N} [\hat{y_i} - \beta h(x_i; a)]^2$
  $\rho_m = \text{argmin}_{\rho} \sum_{i=1}^{N} L(\hat{y_i}, F_{m-1}(x_i) + \rho h(x_i; a_m))$
  $F_m(x) = F_{m-1}(x) + \rho_m h(x_i; a_m)$
end for

where the term $L(y, F)$ is the squared-error loss, $\hat{y_i}$ is the unconstrained negative gradient which gives the best steepest descent step direction $-g_m = \{-g_m(x_i)\}_1^N$ in the $N-$dimensional data space at $F_{m-1}(x)$, $h(x; a_m)$ is the member of the parameterised class that produces $h_m = \{h(x_i; a_m)\}_1^N$ most parallel to $-g_m \in R^N$. This constrained negative gradient $h(x; a_m)$ is used in place of unconstrained one $-g_m(x)$ for the steepest descent strategy. This algorithm is modified to handle regression trees.

2.4 Evaluation metric

The evaluation metric considered for this effort is the root mean squared error[5]. It represents the sample standard deviation of the predicted values and observed values. We call these as the residuals when using our model/training data and prediction errors when using actual test data. It is given as follows,

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n} (\hat{y_t} - y_t)^2}{n}}$$

`Example:`
$\hat{y_t} = [2.5, 0.0, 2, 8]$
$y_t = [3, -0.5, 2, 7]$
$RMSE = \sqrt{(0.5^2 + 0.5^2 + 0 + 1^2)/4}$
$RMSE \simeq 0.612$

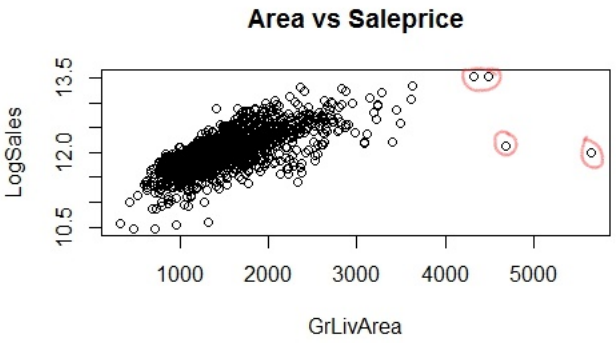

Figure 1: Scatter plot of GrLivArea against the log of SalesPrice to check for any outliers. Few outlier candidates highlighted in red.

3 Experiments

The following are the software components involved in this project effort:

1. Python v3.5.2
2. Python packages: numpy, scipy, sklearn, xgboost, pandas, matplotlib
3. R v3.3.0
4. OS specs: Windows 10 x64, RAM: 8GB
5. Code: `https://github.com/keshavsridhar/Data-mining/blob/master/Regression.py`

3.1 Data preprocessing

In the Ames housing dataset, there are a total of 80 features present. Initial data analysis was done in R to find correlations and to check for outliers with scatter plots and further investigation(Example `Figure 1`). Various stages of processing the data are given as follows:

1. Data analysis
2. Data cleaning
3. Missing/NA value imputation
4. Outlier removal
5. Feature engineering
6. Log transformation
7. Feature normalising and scaling
8. Encoding categorical features

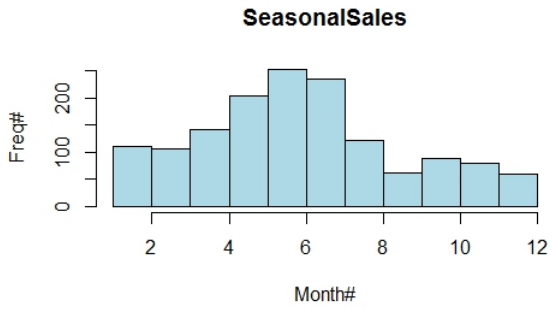

Figure 2: Plotting a histogram of the month sold feature to check for any pattern. Looks like there is some kind of trend which we can cash in on.

Most of the features which have quite a bit of missing/NA data are handled based on the closest data points particular to any house. For example, the LotFrontage represents the amount of frontage in square feet. For the lots that have missing lot frontages we check their neighborhood and assign based on the median of the neighborhood. Similarly for garage area feature we apply median based area if the garage area is missing but other garage fields are present.
Neighborhood feature is interesting because the sale price is quite correlated with some sets of neighborhood implying that some of them are ranked higher than the others. In a similar fashion, we impute missing values for the rest of the features.

Data cleaning was done to check whether the data present made sense logically. For example, a data point had "GarageYrBlt" feature as "2027", so we replaced it with valid data from its nearest neighbours.
All the features were normalised by using the mean and standard deviation. Skewed features and the target variable were log transformed.

3.2 Feature engineering

There are lots of simple features such as different kinds of areas and also some categorical features which we can utilise in several unique ways. One such feature is the $MoSold$ feature(`Figure 2`).

• Introduced a "Seasonal" feature based on the data from $MoSold$ feature.
• "New home" feature based on the categorical feature $MSSubClass = (20, 60, 120)$
• Various "Age" specific features built from $YearSold$,$YearBuilt$,$YearRemodeled$ features.
• "MasVnrScaling" feature. New feature based on the type of masonry house is built($MasVnrType$) and the $MasVnrArea$ features. • All ordinal features were also binned into simpler values as new features.
• Few categorical features which had few classes were converted into a binary Yes/No(1/0) feature.
• Most categorical features were converted as encoded values and also they were transformed into dummy encoded columns as new features(with feature names starting with _featurename).
• Polynomials of top 10 features as new attributes.
In total the final model has around 350 features which includes a majority of dummy encoded columns, binned features, polynomials of top 10 features, new features etc.

3.3 Feature Importance

Some features of the dataset contribute more to the prediction of the target variable. Knowing this, we can now test whether our new feature engineered variables play an important role or not.

As we see in both the lasso model and the ridge model in `Figure 3` & `Figure 4`, that our feature engineered attributes have a good say in the model, thus reinforcing our idea to go ahead with these new attributes as an added bonus to our model. It makes sense in a way that certain features such as Area based features have positive coefficients while other features such as "Age"(older houses),

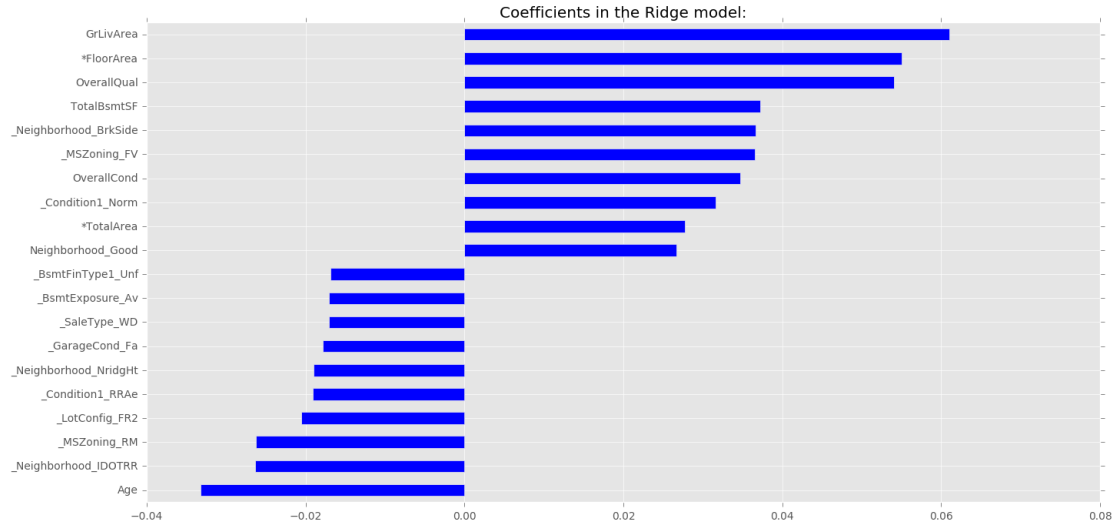

Figure 3: Top10 positive and negative coefficients picked by the Ridge model. The abscissa represents the coefficient weights and the ordinate represents the features of our model.

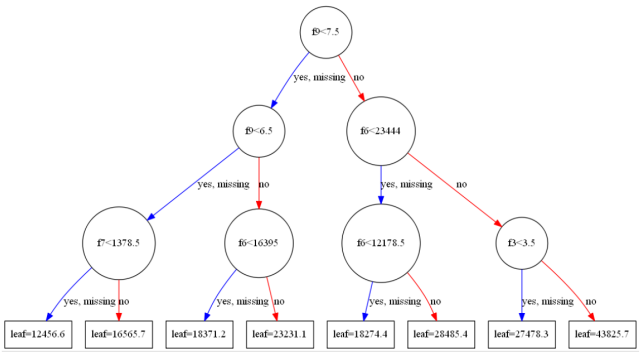

Figure 4: One of the trees built by xgboost with a subset of our features. XGBoost builds many such trees and optimizes the loss function over all of them. The final predicted value will be the sum over all the leaf values of a tree.

"_BsmtFinType1_Unf"(unfinished basement), "_Neighborhood_IDOTRR"(the neighborhood is close to a rail road) have a negative weight attributed to it.

The best result was the model with `XGBoost + LASSO + feature engineering(FE)` which achieved a rank of 165/2366(top 7%) on the leaderboard.
This reinforces the belief that feature engineering plays a crucial role and has a positive impact on the prediction of the sales price.

4 Results

The results are displayed in `Table 1`. The evaluation metric for multiple regressors was calculated on the final output of the blended model.

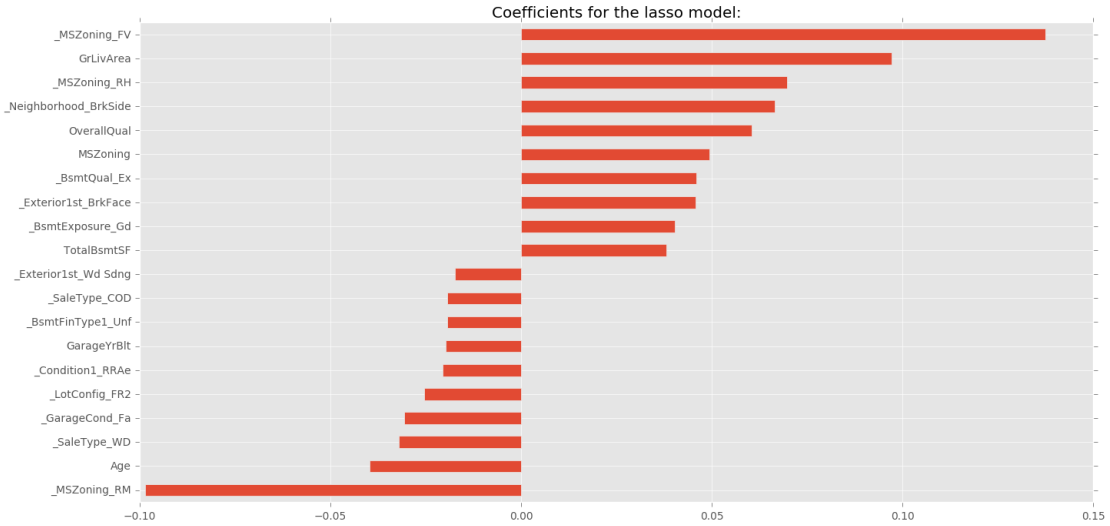

Figure 5: Top10 positive and negative coefficients picked by the LASSO model. The abscissa represents the coefficient weights and the ordinate represents the features of our model.

Table 1: Evaluation of models with different settings

Model	**RMSE**$_{train}$	**RMSE**$_{test}$
`XGBoost`	0.0398031	0.12529
`XGBoost + FE`	0.0354913	0.12332
`LASSO + FE`	0.0970031	0.11917
`Ridge + FE`	0.0971086	0.11967
`XGBoost + Ridge + FE`	**0.0634703**	0.12286
`XGBoost + LASSO + FE`	0.0633683	**0.11670**
`XGBoost + Ridge + LASSO + FE`	0.0742142	0.11871

Note: FE is Feature Engineering

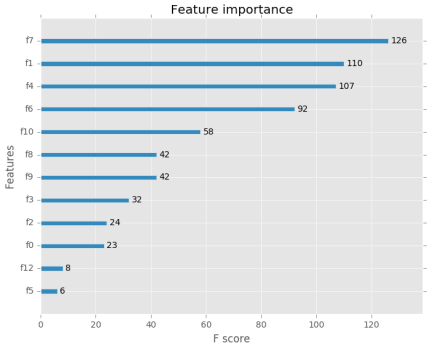

Figure 6: Feature importance by F-score for the Xgboost model. F-score is the harmonic mean of precision and recall given as, $(2 * precision * recall)/(precision + recall)$

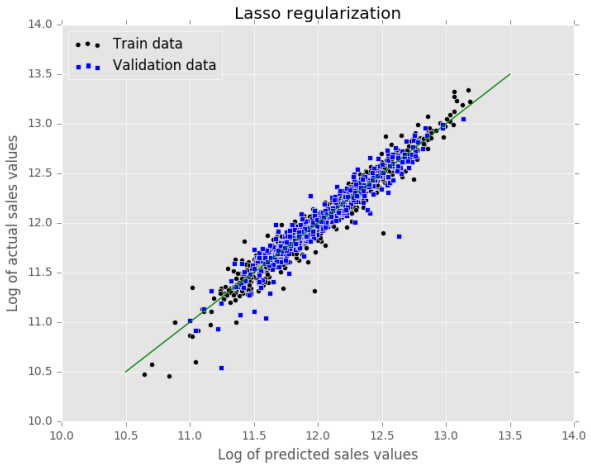

Figure 7: Scatter plot of predicted vs actual values by LASSO regularization model.

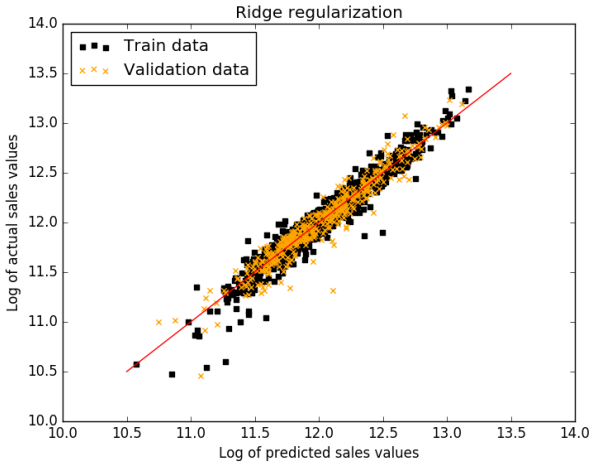

Figure 8: Scatter plot of predicted vs actual values by Ridge regularization model.

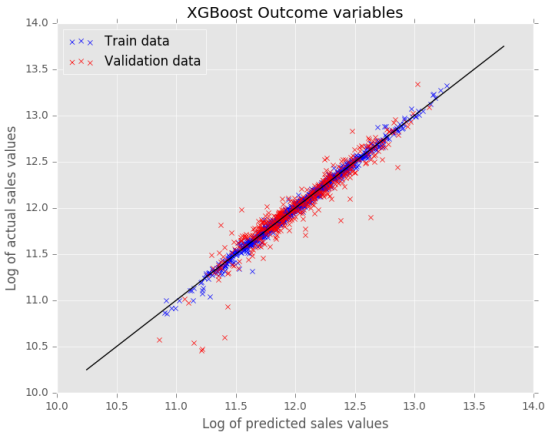

Figure 9: Scatter plot of predicted vs actual values by XGBoost model. Compared to the regularized linear models, it looks like the XGBoost model is outperforming them. The points along the line seems to be much closer.

5 Conclusion

The following are the conclusions extracted from this project effort:

• Introducing some kind of bias into the model via regularization does reduce variance which could result in lower prediction error. Thus regularization could reduce prediction error.

• Both Ridge and LASSO are effective regularizers but LASSO can produce sparse solutions by setting some of the coefficients 0.

• Parameter tuning is a highly essential component while building gradient boosted trees.

• Constructing new features based on top 10 features of a model is a viable method of feature engineering.

• Blending different models results in averaging the prediction error and hence is a valid way to get better prediction outcomes.

6 Future Work

In addition to what has been done, we feel we can improve upon our results by taking a few more steps as follows:

• Building a voting based ensemble approach with random forests, xgboost and LASSO regularization.

• Checking the errors from the validation set and using the LASSO path to find out where exactly the model is going wrong.

• Use bagging based modelling approach along with regularization to check if it decreases the variance.

• Try out the linear combination of L_1, L_2 regularizers, i.e Elastic net regularization.

7 Acknowledgements

We would like to thank Prof. Mehmet Dalkilic for giving us an idea about the quintessential aspects of data mining as we have found that to have a major impact on the outcome of the model. Also we would like to thank the team of associate instructors for taking time to guide us and clarify our questions over the course of the semester.

References

[1] Robert Tibshirani: Regression Shrinkage and Selection via the Lasso `http://statweb.stanford.edu/~tibs/lasso/lasso.pdf`

[2] Andrew Y. Ng: Feature selection, L_1 vs. L_2 regularization, and rotational invariance `http://www.machinelearning.org/proceedings/icml2004/papers/354.pdf`

[3] Tianqi Chen and Carlos Guestrin: XGBoost: A Scalable Tree Boosting System `http://dmlc.cs.washington.edu/data/pdf/XGBoostArxiv.pdf`

[4] Jerome H. Friedman: Greedy Function Approximation: A Gradient Boosting Machine `https://statweb.stanford.edu/~jhf/ftp/trebst.pdf`

[5] Wikipedia: Root Mean Squared Error https://en.wikipedia.org/wiki/Root-mean-square_deviations