University of Rochester

Semester Project

Data Mining
Team Name: **KamiKakushi**

Predicting House Prices: Advanced Regression Techniques

Author:
Shouman Das
Adrian Eldridge

Supervisor:
Dr. Ji Liu

May 15, 2017

Abstract

In this project, we participated in a data science competition organized by Kaggle. We used different kinds of feature engineering and apply lasso and XGBoost to make our prediction. We also used stacking, averaging numerous models and self-training. But our final model is a weighted average of predictions from different models, which gives us the best rmse in the public leaderboard.

1 Overview

Our main data is taken from the kaggle website which gives us 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa. This competition challenges to predict the final price of each home. In our analysis, we start by inspecting the data thoroughly, then did some exploratory analysis to detect some outliers. We also made some cleaning and preprocessing on the dataset. Finally, we applied different models on our dataset to make our prediction.

2 Dataset

Our data files contain two csv files: train.csv and test.csv. Each data file has information about various features (79 in total) about residential houses in Ames, Iowa. The last column in the training file is the sale price of the house, but the testing file does not have this column. After inspecting the data set, we find that there are numerical as well as categorical features. For example, some of the features and their description are given below.

- LotFrontage: Linear feet of street connected to property
- Street: Type of road access
- LandContour: Flatness of the property
- Utilities: Type of utilities available
- GrLivArea: Above grade (ground) living area square feet
- Bedroom: Number of bedrooms above basement level

- Kitchen: Number of kitchens
- KitchenQual: Kitchen quality
- GarageArea: Size of garage in square feet
- PoolArea: Pool area in square feet
- YrSold: Year Sold

Some of the features contain a lot of missing data(NAs) which we will discuss in the preprocessing section.

3 Exploratory Analysis

We will do some basic exploratory analysis on our dataset. First of all, lets look at the histogram of the sale prices of the training set.

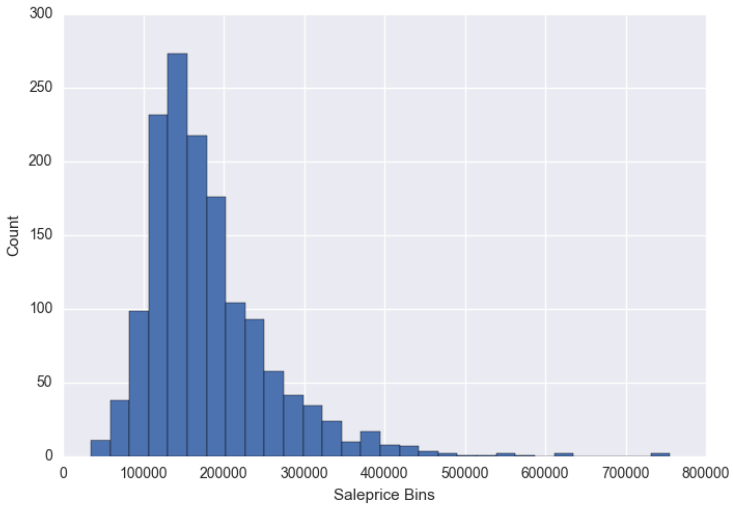

Figure 1: Histogram of Sale prices of houses.

If we look closely, we can see that there is some kind of skewness in the sale price. So while modeling our prediction algorithm, it might be a good idea to use the log(1+feature) transformation. After making this transformation, this distribution looks like more of a normal distribution.

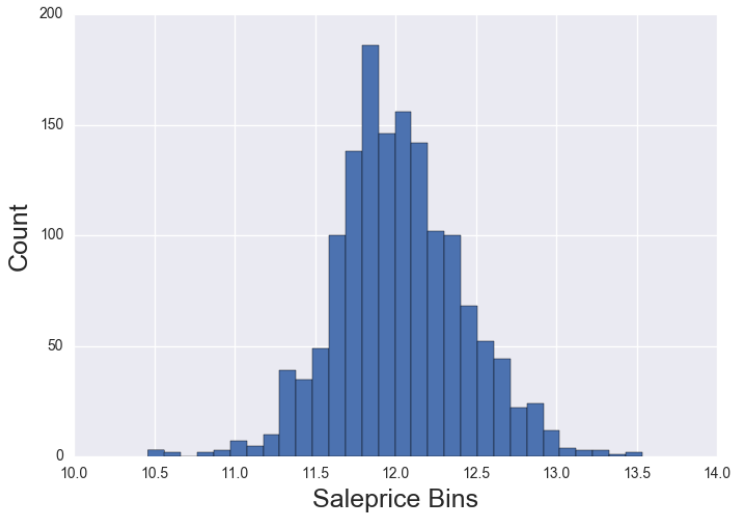

Figure 2: log(1+Sale price) Histogram.

Next we try to detect some outliers from the data and will remove it from our modeling. After inspecting the data closely, it is reasonable to think that living area of a house is an significant predictor for the price. So we look at the scatter plot of 'SalePrice' vs 'GrLivArea'.

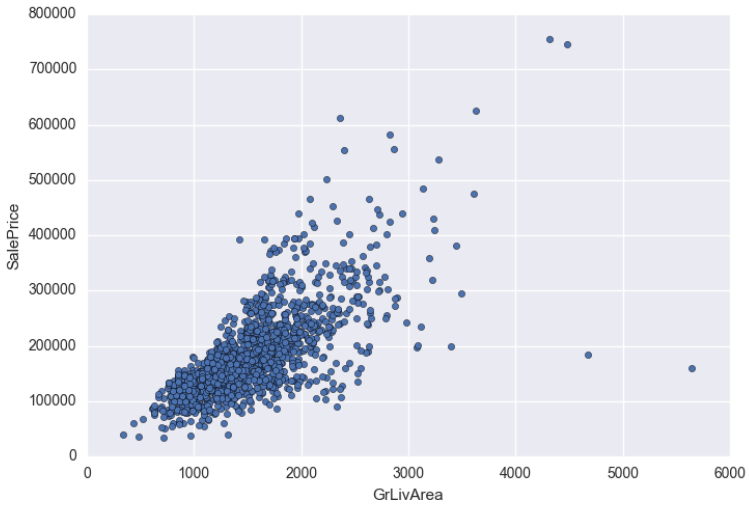

Figure 3: SalePrice vs GrLivArea .

As expected we see that price has positive correlation with the living area. But in the lower right corner, we can detect two points where the price is ambiguously low even though the living area is huge. We assume that these might be some kind of under developed area or agricultural area where the residential facilities are limited. So we treat them as outliers and remove it from the prediction model.

Now lets also investigate the sale price vs some feature. For example, we will consider the 'OverallQual' column which represents the overall quality of the house.

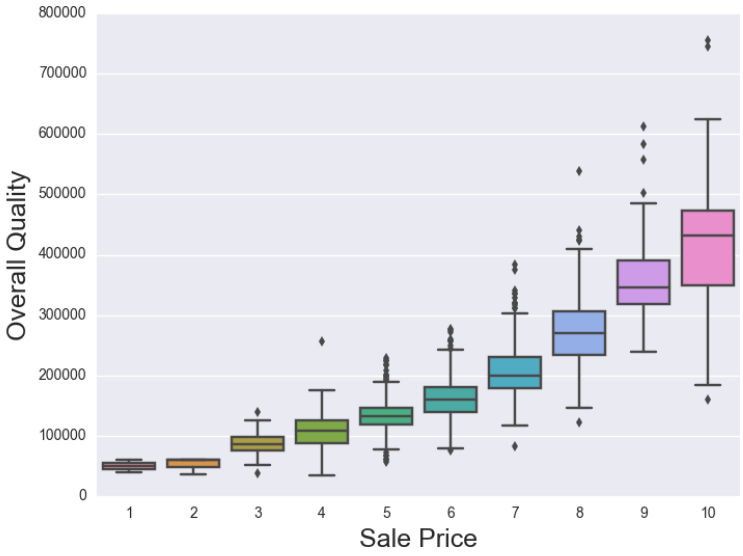

Figure 4: SalePrice vs OverallQual Boxplot.

Clearly it is noticable that as the quality increases the median of the sale price also increases.

4 Data Cleaning & Preprocessing

In this section we will discuss various methods we have used to clean and preprocess the data. We have several major issues to work with the data set.

1. Filling up NAs.
2. categorical values which needs to be numeralized.
3. Data transformation

4.1 Filling up Missing Values

If we look at out data matrix, we find that a lot of columns contains plethora of NAs. Some columns have more than half. But we need to be careful here to notice that some NAs are not missing values. They just represent that

particular feature doesn't exist for that particular house or that feature is not applicable. Lets have a look at some of the features which contains a higher number of NAs.

Feature Name	Total	Percentage
PoolQC	1452	0.995885
MiscFeature	1404	0.962963
Alley	1367	0.937586
Fence	1177	0.807270
FireplaceQu	690	0.473251
LotFrontage	259	0.177641
GarageCond	81	0.055556
GarageType	81	0.055556
GarageYrBlt	81	0.055556
GarageFinish	81	0.055556
GarageQual	81	0.055556
BsmtExposure	38	0.026063
BsmtFinType2	38	0.026063
BsmtFinType1	37	0.025377
BsmtCond	37	0.025377
BsmtQual	37	0.025377
MasVnrArea	8	0.005487
MasVnrType	8	0.005487
Electrical	1	0.000686

Some of the features have a very high percentage of missing values. For example, PoolQC, Alley, Fence etc have NAs which count more than 80 %. But in the data description file we can see that these are not real NAs. For instance, PoolQC NA doesn't mean that the data is missing, rather it implies that the house does not have any pool. So there is no need to fill up any missing values here. The same idea applies for Alley, MiscFeature, Fence etc.

Some features contains some real NAs. We use different methods to fill up there values. For example, if the feature is numerical we choose mean or median, if categorical we choose mode. But while choosing this values, we used one important feature of any house: the neighborhood the house is in. In our data, there is a column which gives us the neighborhood of a house which is one of 25 neighborhoods in Ames, Iowa area. We first make grouped data based on the neighborhood. Then for a house which contains

any feature with NA, we fill up that value with the mean/median of that house's neghborhood. Lets describe this method with an example.

Suppose we have a house which has NA in its GarageType feature. We first detect the neighborhood of that house(say neighborhood NoRidge). Then take the mode of the GarageType of NoRidge neighborhood.

4.2 Encoding the Data Matrix

After we have the data matrix with all the values filled up by methods described above, we will concentrate on the encoding. We have to encode the categorical values. Note that, there are two types of categorical column.

- Ordinal Categorical Values: To encode this type of column, we use a corresponding set of numerical values which also have the same ordinal relations. For example, suppose we want to encode some quality related features. We use the mapping, quality = {None: 0, "Po": 1, "Fa": 2, "TA": 3, "Gd": 4, "Ex": 5} . Here, we can easily see that the quality features maintain their ordinal relations after being encoded to numbers.

- Nominal Categorical Values: Some categorical columns contain only nominal values which does not have any internal ordinal relations among them. For this kind of columns we use 'One Hot Encoding'. One hot encoding transforms categorical features to a format that works better with classification and regression algorithms.

4.3 Data Transformation

We use various techniques to transform the data, add more columns to the data.

4.3.1 Adding Columns

From the given data matrix, we create some more columns add it to the data matrix. For example, we consider the Neighborhood column and depending on their median price of house, we make a new column 'NeighborhoodBin' indicating how rich or poor the neighborhood is.

Another kind of transformation we use is the simplification of some quality related columns. For example, OverallQual feature has 10 levels. We change

it to be of 3 levels by using the following map, {Very Excellent: 3, Excellent: 3, Very Good: 3, Good: 3, Above Average: 2 , Average: 2 , Below Average: 2 , Fair: 1 , Poor: 1 , Very Poor: 1 }

Also, we added some binarized data, for example "Has2ndFloor", "HasShed", "HasOpenPorch" etc.

4.3.2 Scaling and Normalization

This is only for the numerical columns. We did the use usual normalization procedure by subtracting the mean and dividing by the standard deviation.

4.3.3 $\mathbf{Log}(1+x)$ Transformation

To avoid the skewness of the data, we use this transformation. We selected .75 as the cutoff threshold of the skewness. This means that if the skewness of any column is more .75, we apply this log transformation.

Now that we have a cleaned, preprocessed data matrix we will start building our prediction models.

5 Prediction Models

After applying our preprocessing and data transformation, we get a data matrix which contains around 400 columns. We tried different regression techniques such as random forest, lasso, ridge regression, XGBoost etc. Among this a linear combination of lasso and xgboost works best for our leaderboard score.

5.1 Random Forest and Overfitting

First we tried to apply the random forest regressor from scikit learn. We used some manual parameter tuning and we found that random forest overfits the data. We deduced this conclusion by looking at the rmse on training and testing set.
Training RMSE: 0.052884
Testing RMSE : 0.14857

5.2 Ridge Regression

We will try the RidgeCV method from SciKitLearn. The objective function for ridge regression is

$$\frac{1}{2n}\|y - Xw\|^2 + \lambda\|w\|^2$$

By default, RidgeCV() method in SKLearn performs Generalized Cross-Validation, which is a form of efficient Leave-One-Out cross-validation. We got a reasonabale improvement in our rmse.

Training RMSE: 0.09259
Testing RMSE: 0.11926

5.3 LASSO

In lasso, the model itself can choose the columns which are needed for prediction. We used Python's SciKitLearn package. and the formulation about the objective cost function is

$$\frac{1}{2n}\|y - Xw\|^2 + \alpha\|w\|_1$$

After our preprocessing we get an data matrix which contains 392 columns. Then we did some cross validation by grid search to choose the optimized parameter. After training our model we get the following rmse for our training and testing set.

Training RMSE: 0.097599
Testing RMSE: 0.11896

5.4 XGBoost

The main difficulty about applying this method is the huge number of parameters the model involves. We tried to do some manual parameter tuning by looking at the rmse and after some trial and error we get some good result.

Training RMSE: 0.07909
Testin Rmse: 0.11979

5.5 Combination of Lasso and XGBoost

We applied this method only for our testing results. We made a weighted average from the lasso and XGBoost predictions. We gave lasso prediction a bit more weight than the xgboost prediction. After this our final leaderboard score improved to **0.11696**.

5.6 Polynomial Feature with lasso and xgboost

We tried adding polynomial combinations of features in an attempt to discover hidden interactions in the data set that could improve the performance of our regressors. We used sklearn.preprocessing.PolynomialFeatures to create all combinations of one or two features up to polynomial degree two. This yielded almost 70,000 new features. We then added to the data set only those new features which had a correlation of greater than 0.75 with the sales price. This expanded data set did not yield any improvement results than the original data set with the XGB and Lasso models.

Training error: 0.11274

Testing error: 0.12869

5.7 Self-Training (Semi-Supervised Learning)

In this method we made two rounds of predictions. In the first round, we used XGBoost, Lasso, NeuralNetwork, RandomForest, and ElasticNet to create five different predictions of the test set. Houses in the test set which had very confident predictions (variance of the predictions < 0.003) where then added back into the training set with the mean of the first round predictions as their stated house price. Then a second round of predictions was applied using the augmented training set. Due to the time consuming nature of this method we did not do any cross-fold validation testing and therefore only have the testing error. This method did not yield better results.

Testing error: 0.13507

5.8 Stacking

We tried many different variations on this method in which a second round of regression is applied to the predictions of a variety of first round regressors. Using a Lasso model to combine the XGBoost and Lasso models gave the following performance:

Training error: 0.09127
Testing error: 0.1215

Combinations of other regressors tended to perform worse. Using XGBoost, Lasso, NeuralNet, RandomForest, and ElasticNet with a secondary Lasso model to blend the predictions yielded the following errors:

Training error: 0.12923
Testing error: 0.15216

These numbers shows that there might be some underfitting issue since the training error is too high comparing to other models.

5.9 Combination of Several Models

In this attempt, we combine those models which have performed better and make a weighted average of our prediction on the testing set. We give more weight to those models which have performed better. We manually tried to tune the weight on the predictions from different models and used the following weight :

$$y = .25 * ridge + .35 * lasso + .35 * xgb + .03 * poly + .02 * stack$$

This attempt gives us a RMSE of **0.11676** on our testing data. And this is our best accuracy.

6 Conclusion

In this project, we have got our hands on a practical real world dataset and made prediction models using various machine learning techniques. Our analysis on this data implies that no single machine learning model works

best for the data. To achieve a better accuracy, we have used a combination of results from different models. In our case, the combination of five different models with better performance works best. There might be some other inherent combination among the other models which could produce more improved results.

7 Acknowledgment

We would like to thank out professor Dr. Ji liu for helping us with various insights and expertise. Our TAs for guiding us during various points of our course work.

References

[1] Comprehensive data exploration with Python.
`https://www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-python`

[2] Lasso and XGBoost models.
`https://www.kaggle.com/humananalog/xgboost-lasso`