A House Price Predictor

Capstone Project

Yuanhang Ren
December 30th, 2016

I. Definition

Project Overview

How do home features add up to its price tag? Well, this question maybe difficult to answer.With **79** explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, my mission is to **predict the sale price for each house** with these variables.The more accurate my model predicts, the more buyers and real estate agents will get benefits.It will be easier for Buyers to buy houses more suitable for them and for real estate agents to sell more houses,it sounds great for both buyers and real estate agents.But,how should we do ?

The key technique used here is called *machinge learning*,it takes a set of known inputs which is related to problems you want to solve and predict an output on what you are interested in.

To summarize,the problem domain here is *machine learning* while the origin of this project is that real estate agents want to do something good for both themselves and buyers. As a competition of Kaggle, all related data is provided here,which should be part of *Ames Housing Dataset*

With the power of machine learning, we can do something really amazing,here we go !

Problem Statement

To be specific, the problem is a *supervised regression*[1] problem which is a branch of machine learning and target here is `SalePrice` with **79** input features,which means you should use `MSSubClass` `MSZoning` `LotFrontage` and other variables in `data_description.txt` to **predict** `SalePrice`.

In order to solve this problem,first I will explore the dataset to see if there are missing values and what kind of data in each variable,is it a text or number etc.Second I'm going to visualize some properties such as distribution of variables and statistical values in variables.Third I'll come up with some models like Lasso,RandomForest,Bagging,Adaboost,GradientBoosting to fit the data.Finally,models will be tuned and blended to get the best performance to create the submission file.

Only the `Id` and `SalePrice` should contained in submission file,so `SalePrice` on `test.csv` is what I'm going to output in the end.

Metrics

Different machine learning problems always have very different metrics,for classification problems *AUC* is often used, since our problem is a regresison problem, *mean absolute error* and *mean squared error* can be used.But,notice that this is a Kaggle competition, the metric for different problems in Kaggle are often different.In this problem,the metric here is Root-Mean-Squared-Error.

If $\hat{y}_i$ is the predicted value of the ith sample ,and y_i is the corresponding true value, then the mean squared error (MSE) estimated over $n_{samples}$ is defined as

$$MSE(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2$$

then

$$RMSE(y, \hat{y}) = \sqrt{MSE(y, \hat{y})}$$

Note:here y is the logarithm of the observed sales price while $\hat{y}$ is the logarithm of the predicted value.(Taking logs means that errors in predicting expensive houses and cheap houses will affect the result equally.)

II. Analysis

Data Exploration

The dataset can be accquired by competition link I provided above, `train.csv` will be used as train and validation purpose, `test.csv` is the file where we predict.

First,we use *pandas* to load `train.csv` data into *dataframe*,`dataframe.head()` will give us a direct info on this.

> MSSubClass MSZoning LotFrontage LotArea Street Alley LotShape \
> Id
> 1 60 RL 65.0 8450 Pave NaN Reg
> 2 20 RL 80.0 9600 Pave NaN Reg
> 3 60 RL 68.0 11250 Pave NaN IR1
> 4 70 RL 60.0 9550 Pave NaN IR1
> 5 60 RL 84.0 14260 Pave NaN IR1
>
> LandContour Utilities LotConfig ... PoolArea PoolQC Fence \
> Id ...
> 1 Lvl AllPub Inside ... 0 NaN NaN
> 2 Lvl AllPub FR2 ... 0 NaN NaN
> 3 Lvl AllPub Inside ... 0 NaN NaN
> 4 Lvl AllPub Corner ... 0 NaN NaN
> 5 Lvl AllPub FR2 ... 0 NaN NaN
>
> MiscFeature MiscVal MoSold YrSold SaleType SaleCondition SalePrice
> Id
> 1 NaN 0 2 2008 WD Normal 208500
> 2 NaN 0 5 2007 WD Normal 181500
> 3 NaN 0 9 2008 WD Normal 223500

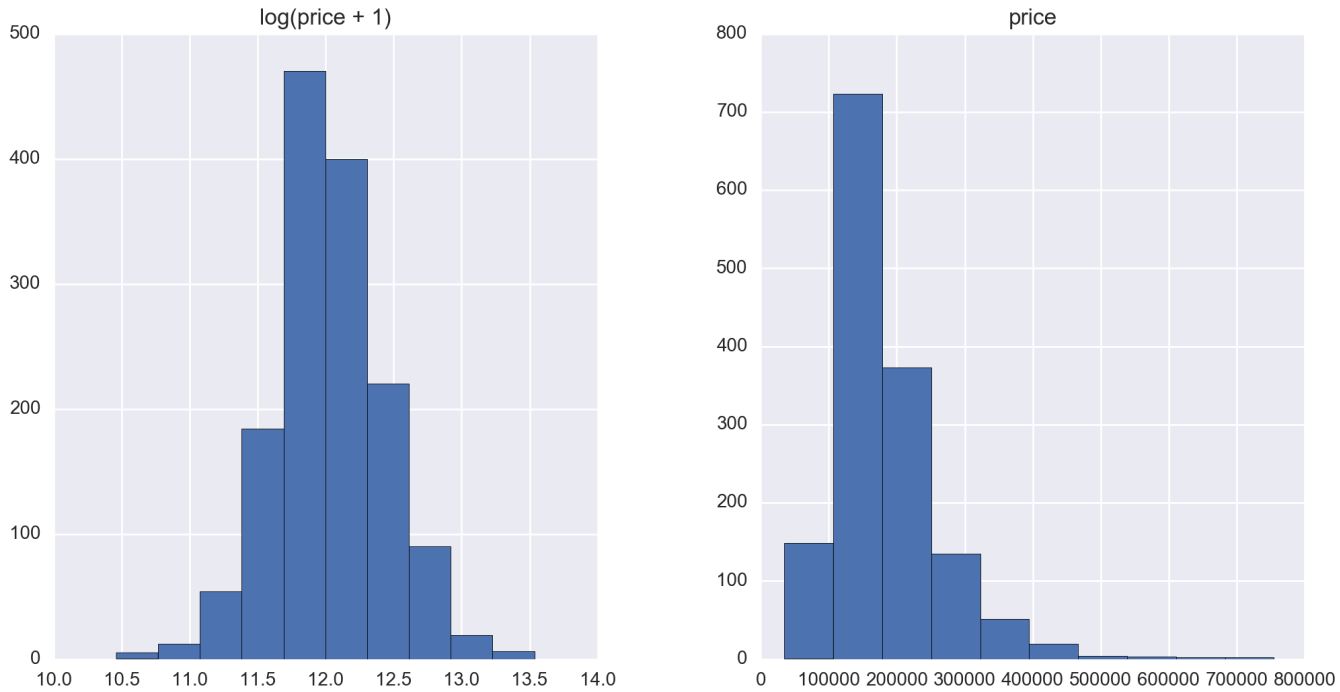
> 4 NaN 0 2 2006 WD Abnorml 140000
> 5 NaN 0 12 2008 WD Normal 250000
>
> [5 rows x 80 columns]

As you can see, there are **79** features and target, which is `SalePrice` is also contained. There are two kinds of features in training data,one is numerical values like `LotFrontage`-linear feet of street connected to property while the other is categorical values like `MSZoning` which identifies the general zoning classification of the sale.They all provided in `data_description.txt`. Another tricky thing we found is there are lots of missing values in the *dataframe*, denoted as *NaN*.

Different type of features corresponds to different preprocessing steps and we also need to fill in missing values with some reasonable values during preprocessing

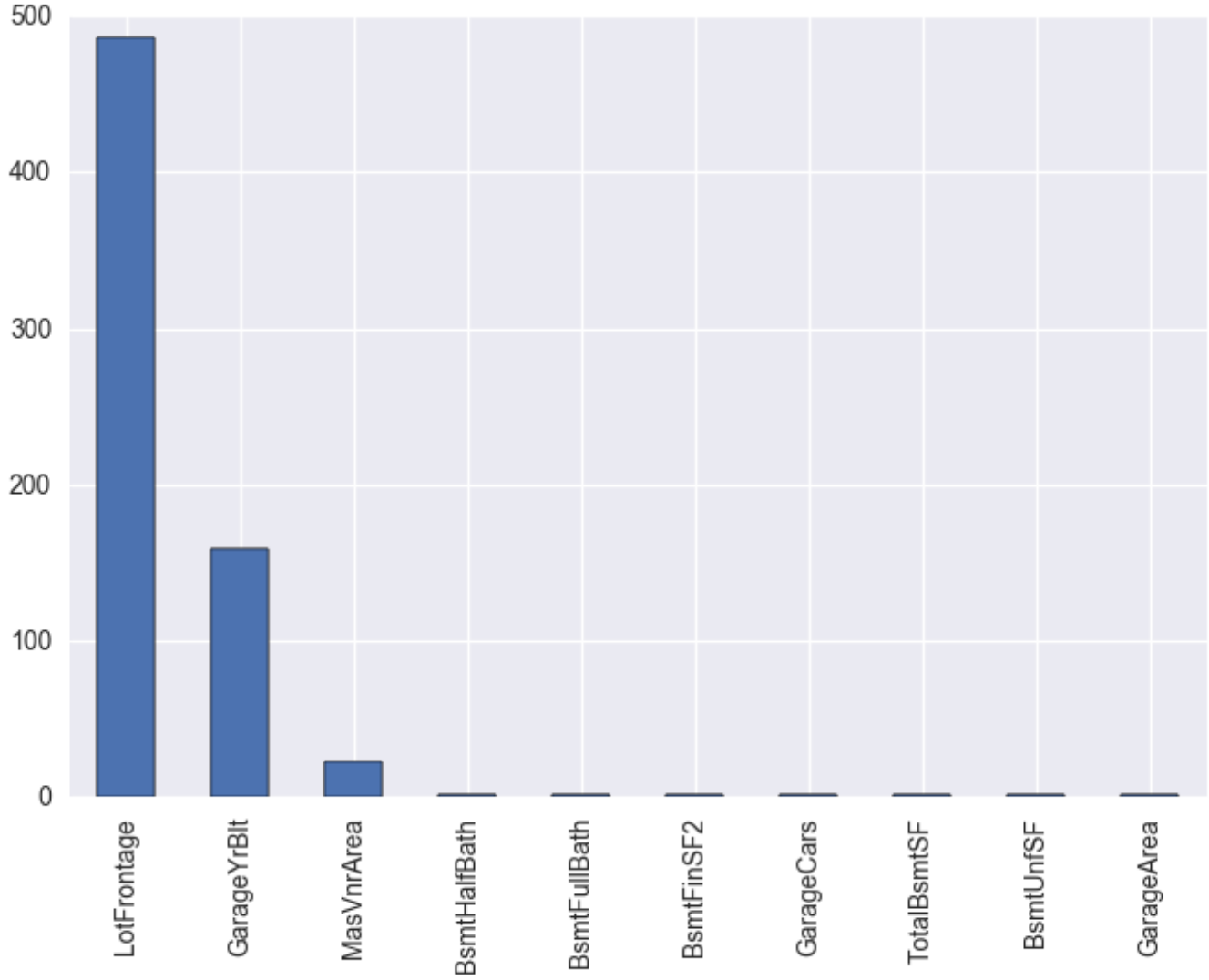
Exploratory Visualization

Let's take a look at the target distribution.



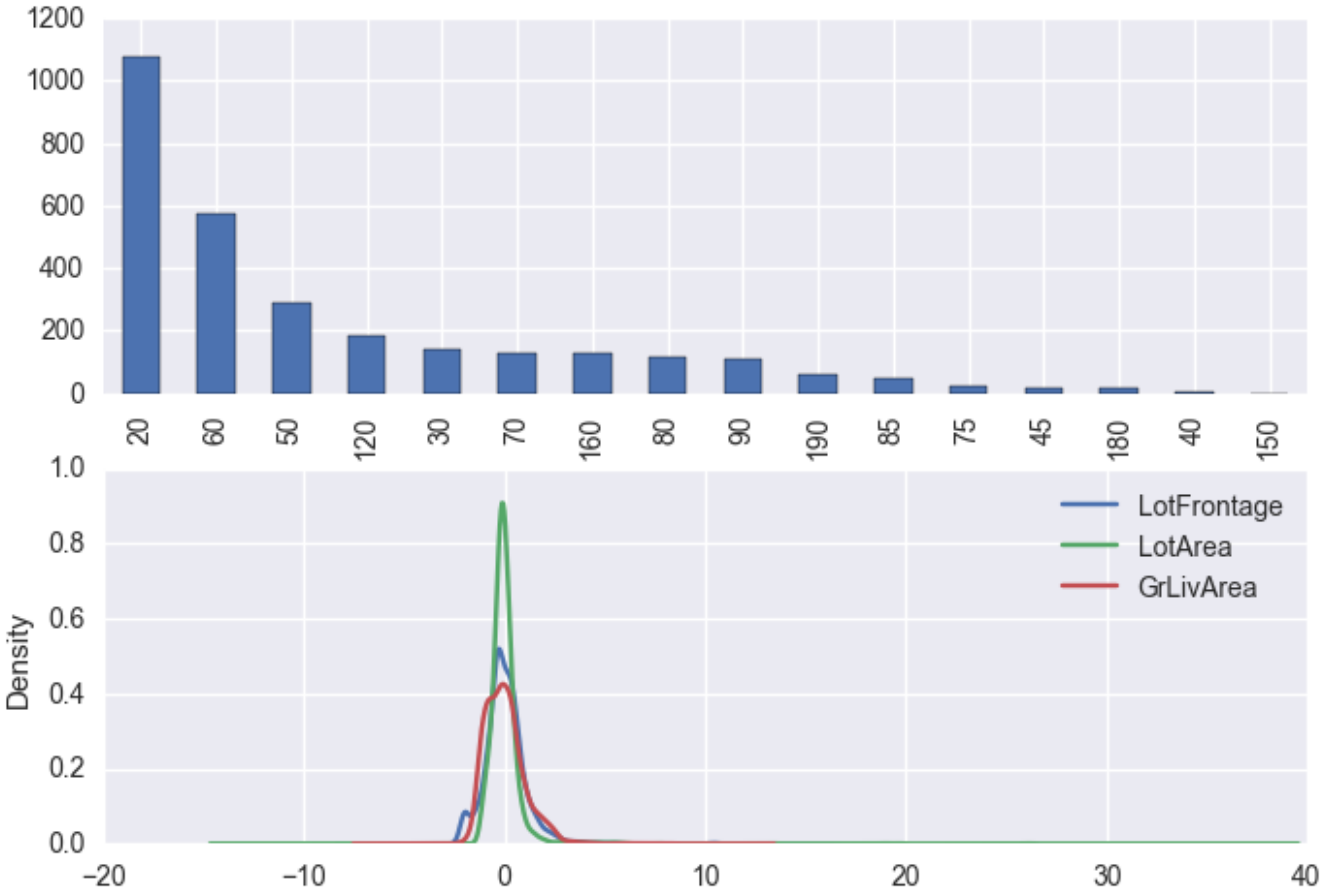
Hmm...the original price is skew, in order to get a better model, we need to transform it, a well known transformation for skew data is *log transform*[2], so we log price and plot on the left, then the distribution is fairly normal.

Notice that numerical columns have a lot of missing values, let's count the missing values in numerical columns and plot a bar chart for the top 10 features.



Well, result shows the `LotFrotage` feature is the column who has the most missing values in all features.How to handle the missing values?Just read the later section :)

What about the featuers themselves, are categorical features or numerical features skew?Let's pick up some features and plot them.



The upper plot is the distribution of `MSSubClass` in train and test csv files while the bottom plot is the density of `LotFrontage` `LotArea` `GrLivArea` in train and test files. I standardized them in order to plot them in one picture and compare.

From the above plot, we can draw conclusions as follows:

- A lot of features maybe skewed(including `MSSubClass`)
- `LotArea` has a higher density than others

Although there maybe other important info I didn't draw here, we can move on to do some interesting modeling work.

Algorithms and Techniques

I will first preprocess the data,do some feature engineering and then I'll use *LinearRegression*[3], *Lasso*[4], *RandomForest*[5], *Bagging*[6], *AdaBoost*[7], *GradientBoosting*[8] to model the data, the characteritics are as following:

- Linear Regression
  Pros:easy to use,almost no need to tune parameters
  Cons:performance is bad for most cases
- Lasso
  Pros:As a regularized linear model,quick,easy to use,feature selection
  Cons:linear model may hard to fit non-linear data
- RandomForest
  Pros:tree based model,hard to overfit
  Cons:performance is not so good if data contains redundancy
- Bagging
  Pros:ensemble method,hard to overfit
  Cons:slow,performance improvement is limited
- Adaboost
  Pros:ensemble method,hard to overfit,slightly better than bagging
  Cons:hard to parallelize,even slow than bagging
- GradientBoosting
  Pros:ensemble method,faster and better than adaboost if use XGBoost
  Cons:too many parameters to tune

I used *scikit-learn*[9] to do modeling except *GradientBoosting*,which used *XGBoost*[10].Data preprocessing needs *pandas*[11] while ploting needs *seaborn*[12] and *matplotlib*[13].

Benchmark

My benchmark is *LinearRegression*.My other models should beat the benchmark on the metric above.Meanwhile,wish I could be one of competitors in top 30% for my first Kaggle competition :)

III. Methodology

Data Preprocessing

Since our target `SalePrice` is skewed,I loged the target in order to change the distribution,then pop it out from train data,named `y_train`.

We concat the train and test csv data in order to preprocess conveniently,named `all_df`

According to the `data_discription.txt`,`MSSubClass` is category but its *dtype* is **int64**,which means it needs to convert to **object**.So we convert it using `astype()`

Then,encode all the category variables,which used *one-hot* here.With the great power of *pandas*,`get_dummies` will do the work for us.

Next step is for numerical variables,recall there are lots of missing values.I used **mean** value for each missing column to fill in because no indication for missing values in discription file.

Different features has different scale,we must *standardize*[14] featuers if we want our algorithm works.Formula is as following

$$(X-\mu)/\sigma$$

μ is mean value for X and σ is standard deviation for X

Finally, we need to recover train and test,it is a easy slicing for *pandas*, `.loc[train.index]` would be fine.

Thus, we get `X_train`,`y_train`,`X_test`,dimension for X should be **303**

Implementation

I first tried benchmark using **5** folds *cross-validation*[15] on `X_train`,`y_train`.Its mean metric score is **1899763840.03**.

I then tried *Lasso*,*RandomForest*,*Bagging with Lasso*,*Adaboost with Lasso*,*GradientBoosting*,with **default parameters** the same way.Their mean metric

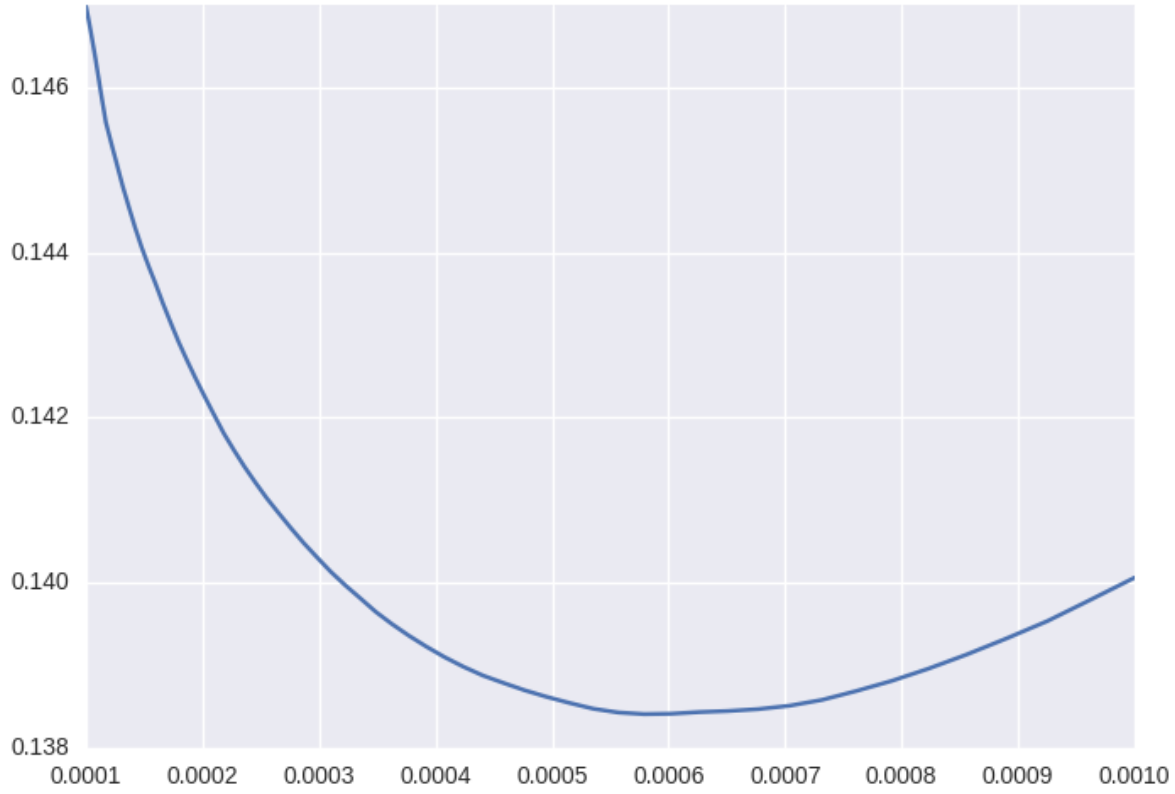
scores are **0.3992**,**0.1495**,**0.3992**,**0.3949**,**0.1309**.

They all beat the benchmark heavily, *GradientBoosting* which is implemented by `XGBRegressor` seems to be the most potential model.

In the next section, I will tune the models and use a black magic called *blending* to enhance the performance.

Refinement

As we all know, `alpha` parameter for *Lasso* is siginificant,so I tuned `alpha` first. I use `np.logspace(-4,-3,60)` to be candidates for `alpha`,which means `alpha` change from 10^{-4} to 10^{-3} and sampled **60** pieces.Then `GridSearchCV` is used to find the best `alpha` on **5** folds of train data using *cross-validation*.

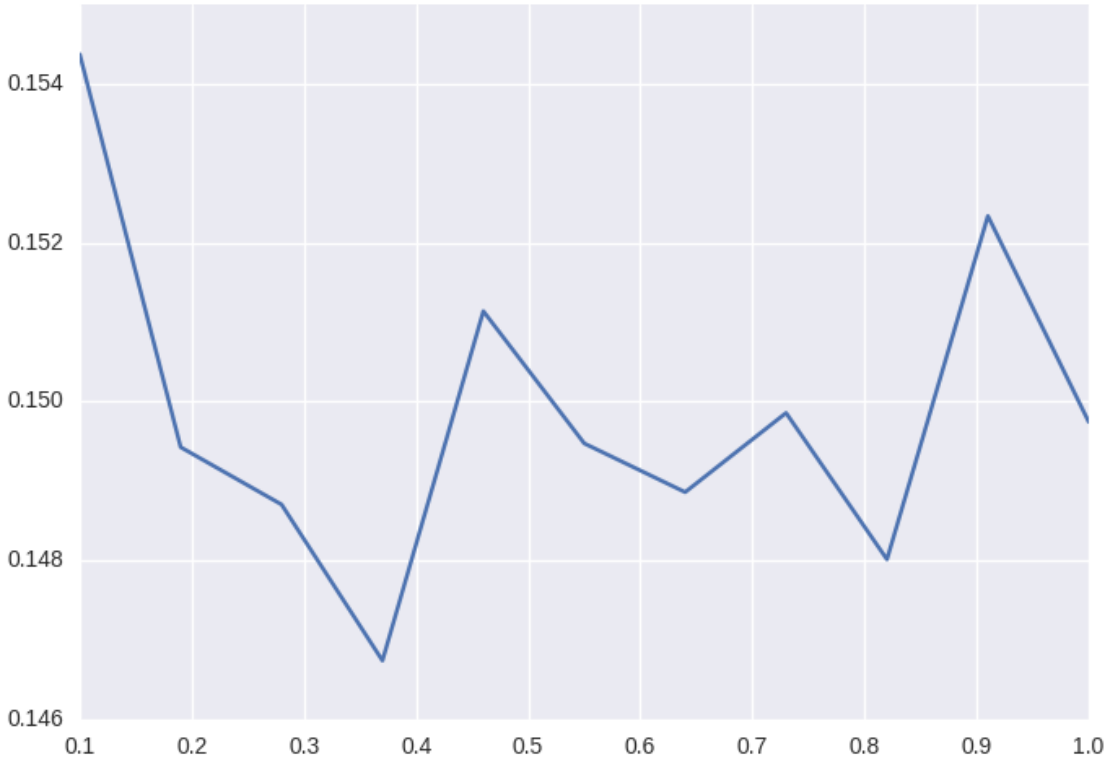


It turns out **0.000579** is the best `alpha` in the period.I also tuned the `max_iter` and `tol` of *Lasso* later,since the metric score didn't increase much so they remained default.

Note:all the following models will use `GridSearchCV` with 5 folds on train data to find the best parameters,so I'll not say it again in the later part.

Next tunned model is *RandomForest*.This model has a lot of parameters to tune.In order to speed up the `GridSearchCV`,*greedy algorithm* is used to tune all the parameters.

- I begin with `max_featuers`,it varies from **0.1** to **1** and sampled **11** pieces using `np.linspace(.1,1,11)`.



  As you can see, **0.37** is the best `max_features`
- Then, `max_depth` and `n_estimators` tuned jointly, since *matplotlib* is hard to draw 3d pictures so I use `grid.best_params_` to get best `max_depth` and `n_estimotors` directly, which are **14** and **700**
- Finally, we can get validation score on *randomforest* model, using **5** folds of train data

Two parameters need to be tuned for *bagging with Lasso*, `n_estimators` and `max_features`.Since ensemble method takes a huge amount of cpu resources, so I

tuned them seperately. The best `n_estimators` and `max_features` are **369** and **0.8**.Due to limited space, I'm not going to show the pictures,cause they are similar with above ones.

I also tuned *Adaboost on Lasso* with `n_estimators` and `learning_rate` jointly. The metric score is highest with **10** `n_estimators` and **1.0e-05** `learning_rate`

Finally comes to the *GradientBoosting*, I have to say, *XGBoost* is amazing , not only its high speed but also its great model performance.But its tuning process is painful for so many parameters need to be tuned.I tuned **12** parameters total with *greedy algorithm*[16].

Let's checkout our performace for the models now, all mean scores are cross-validated on **5** folds of train data.

	La	RF	Bag	Ada	GB
before	0.3992	0.1495	0.3992	0.3949	0.1309
after	0.1351	0.1370	0.1349	0.1346	0.1265

From the tables, we can draw some conclusions

1. Obviously, all models beat the bechmark,whose score is **1899763840**.
2. Thanks for *regularization*, *Lasso* is much better than *LR*
3. From all the ensemble methods, *Gradient Boosted Decision Tree* is the best,thanks for *XGBoost*.
4. *Boosting* is slightly better than *Bagging*
5. *RandomForest* seems not so useful here

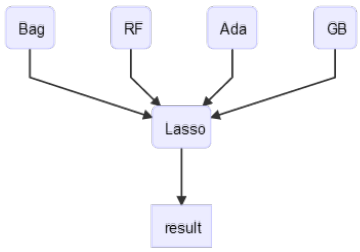
So,what's our final model? *GB*? No,all the models!

Here comes *blending*, the key idea is to use machine learning to assign weights for each base model,which are *RF*,*Bag*,*Ada*,*GB* here.I choose *Lasso* for *blending* model,using `GridSearchCV` to tune `alpha` for it.As expected,its mean score is **0.1249** on **5** folds of `blend_train` data, better than any other models.

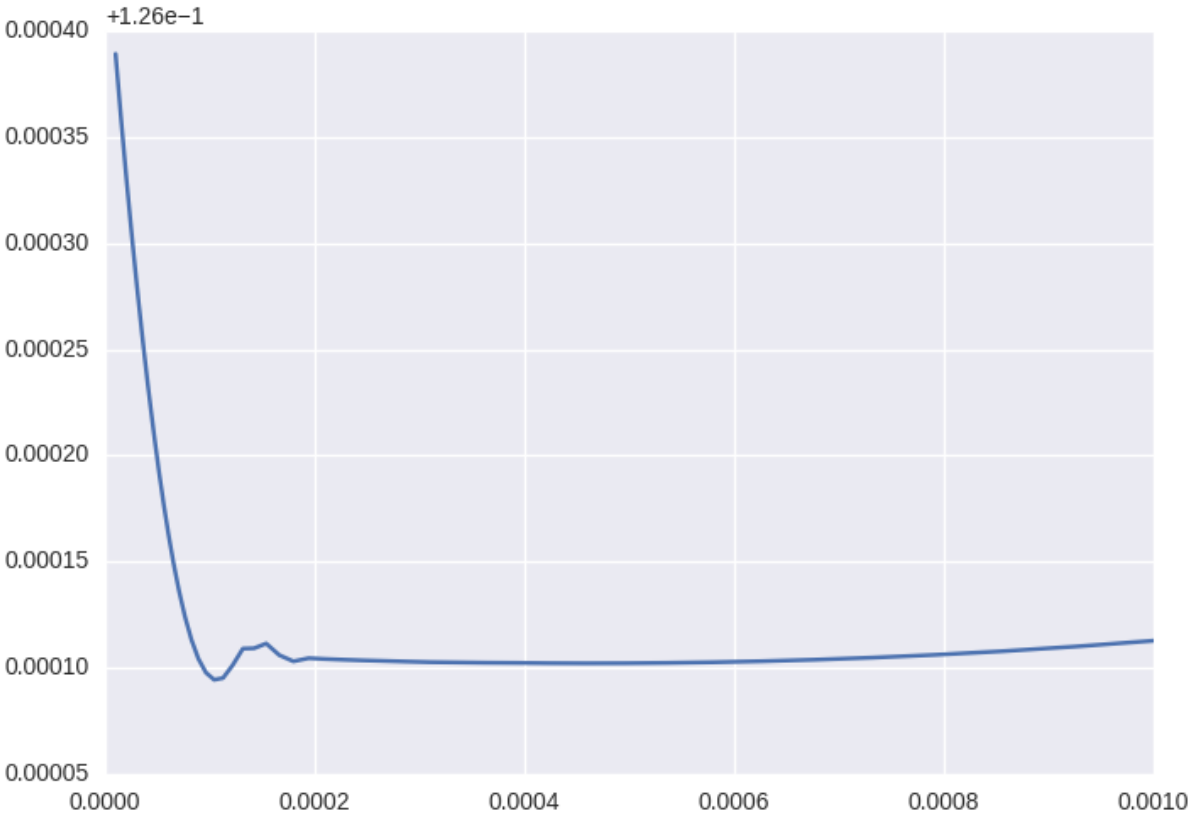
Now,it's time to submmit result on `test.csv` !

IV. Results

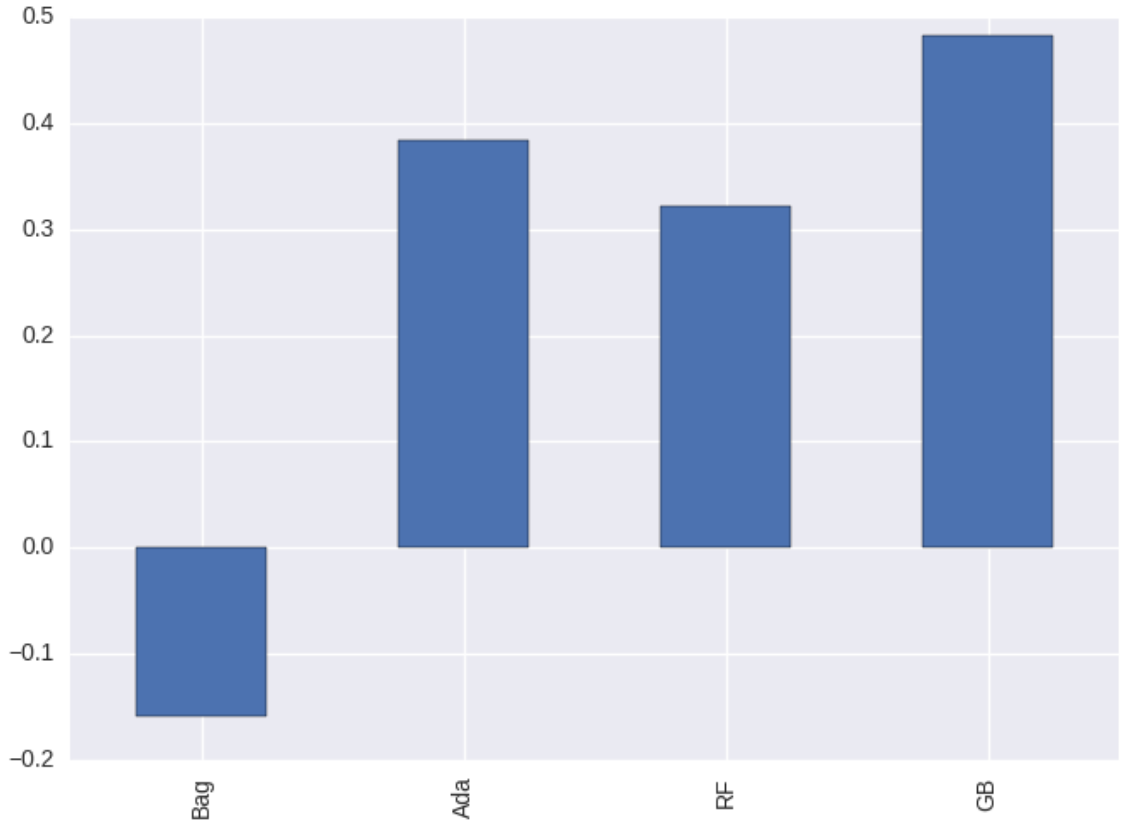
Model Evaluation and Validation



This is our final model,using *Lasso* to blend **4** base models metioned above.



As you can see different `alpha` controls different final performance,**0.0001** is chosen for the final `alpha`.**Note that horizontal axis is changing `alpha` while vertical axis is metric score**. What about the weights of different models?



Kind of surprised, I think *RF* should have the lowest weights, but it turns out, *bagging* has the lowest score.As expected, *XGBoost* contributes most.

I make predictions with tuned *Lasso* on `blend_test` and submit the result csv file.My public score is **0.12266** ranked Top 29% over 3000 teams until 30th December 2016. Perfect!

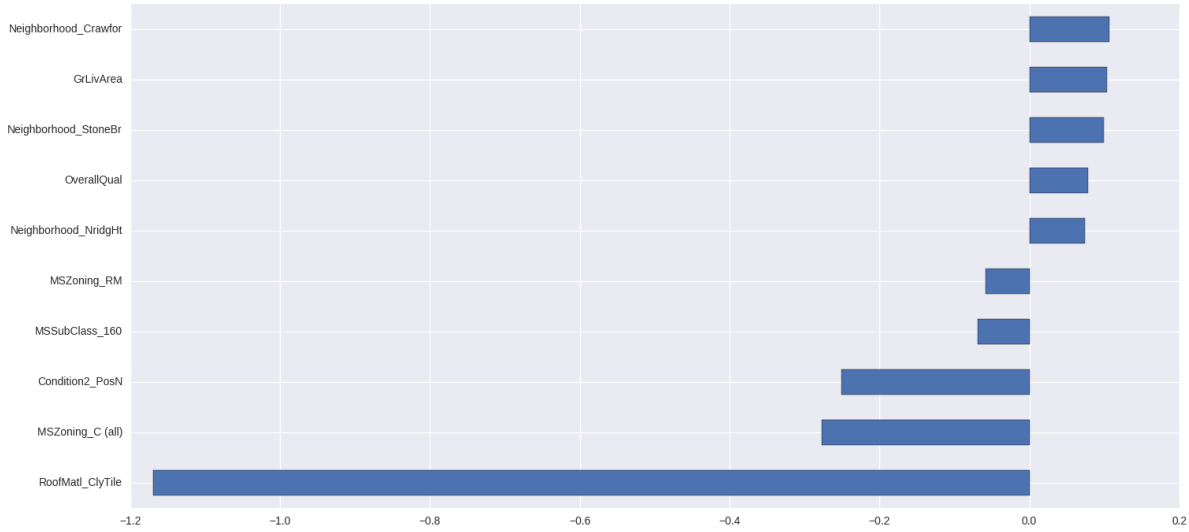
Justification

Let's compare our result with benchmark, **0.1249** is much much better than **1899763840**.The metric score has proved everything, although the best public score is smaller than **0.1**,I think my solution is siginicant enough to solve the problem,and I will continue work on it to enhance the performance until the dead line of this competition.

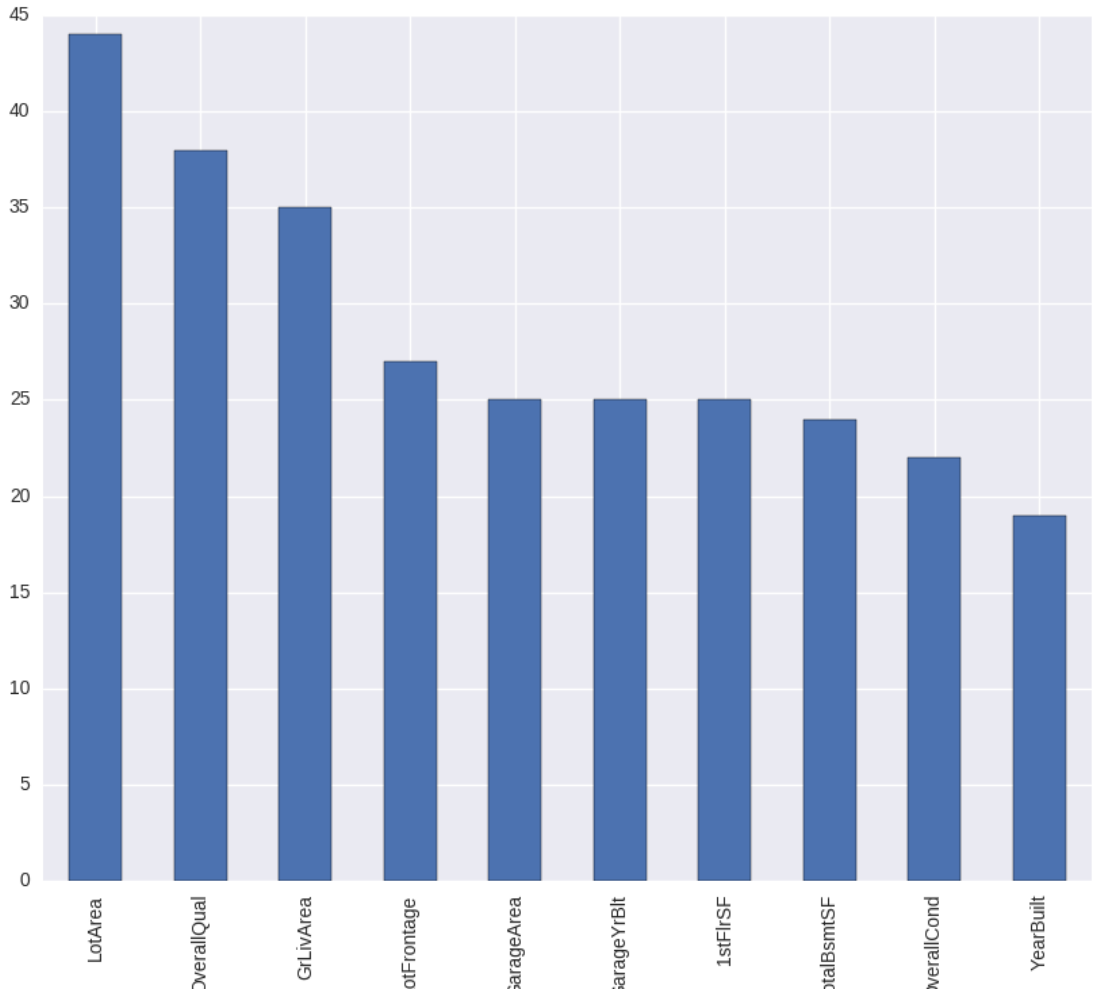
V. Conclusion

Free-Form Visualization

Let's use `lasso.coef_` to list the importance of the top **5** and tail **5** features,see if there are some intresting things.



Hmm...it seems `Crawford` this neighborhood is more intresting than others.There is no doubt that `GrLivArea` -the above ground area by area square feet is also an important positive feature,the more area house has, the more expensive it will be.Then a few other location and quality features contributed positively. Some of the negative features also make sense and would be worth looking into more - it seems like they might come from unbalanced categorical variables.



This is feature importance created by *XGBoost* with top **10** features,x label is feature name while y label is *F-score*[17].As expected, area and quality of the house has a strong influence on the final price.

Reflection

This project is to solve a *regression problem*.I first download train and test data,then did some visulization,preprocessed the data,applied *linear regresison* ,*lasso*,*randomfroest*,*bagging*,*adaboost*,and *GradientBoosting* models to the data and tuned parameters for models ,found that *blending* method is amazing and the performance of *blending* model outperformed the others.

I think the interesting aspects of this project is that your model can predict unknown just from the past experiences,it's awesome,especially when you look at the public

board of Kaggle,noticed that you beat lots of people,this kind of feeling is great.

Meanwhile, this project is also hard on some aspects,you should konw how to preprocess different kinds of data and what pros and cons for different models,if you want to become the top **10**,you should explore data thoroughly and feature engineering carefully,step by step ,you'll become an experienced data scientist

I think my final model and solution fit my expectations for this problem,of course, I will keep working on it.Some of kinds of techniques can be used in a general setting to solve these types of problems like *standardization* and so on.

Improvement

In this project,I should do some feature selection, casue **300**+ features maybe insufficient for **1460** train and validation data.

Furthermore, I should dive into some features,explore their meaning and try to find more useful infomation on predicting the `SalePrice`.

---

[1] Regression: https://en.wikipedia.org/wiki/Regression ↩

[2] Log transform: https://en.wikipedia.org/wiki/Log-normal_distribution ↩

[3] Linear Regression: https://en.wikipedia.org/wiki/Linear_regression ↩

[4] Lasso: http://scikit-learn.org/stable/modules/linear_model.html#lasso ↩

[5] RandomForest: http://scikit-learn.org/stable/modules/ensemble.html#forest ↩

[6] Bagging: http://scikit-learn.org/stable/modules/ensemble.html#bagging ↩

[7] Adaboost: http://scikit-learn.org/stable/modules/ensemble.html#adaboost ↩

[8] GradientBoosting: http://xgboost.readthedocs.io/en/latest/model.html ↩

[9] Scikit-learn: http://scikit-learn.org/stable/index.html ↩

[10] XGBoost: http://xgboost.readthedocs.io/en/latest/ ↩

[11] Pandas: http://pandas.pydata.org/ ↩

[12] Seaborn: http://seaborn.pydata.org/ ↩

[13] Matplotlib: http://matplotlib.org/ ↩

[14] Standardize: https://en.wikipedia.org/wiki/Standard_score ↩

[15] Cross Validation: https://en.wikipedia.org/wiki/Cross-validation_(statistics) ↩

[16] Greedy Algorithm: https://en.wikipedia.org/wiki/Greedy_algorithm ↩

[17] F-score: https://en.wikipedia.org/wiki/F1_score ↩