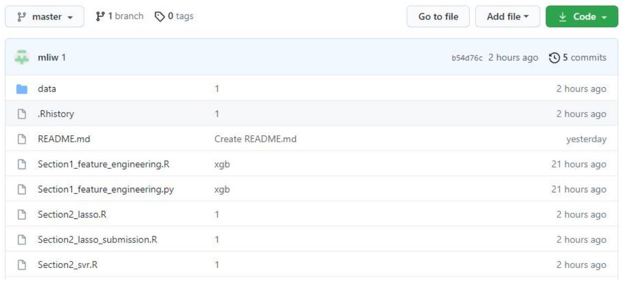
ECO395M STAT Final Project-Kaggle House Price Prediction[*]

Mingwei Li, Xinyu Leng, Hongjin Long, shuheng Huang

Abstract

House Price prediction based on different house properties has become more and more important in real-world business. A precise prediction enables clients to bid or ask reasonable prices, and thus they can save more money or make more profit. This project aims to solve a real-world house price prediction problem on kaggle. In our project, 3 single models including Xgboost, SVM-Regressor and Lasso Regression are adopted. Repeated cross-validation is used to tune parameters. After the 3 single models with the best parameters are obtained, We stack them with different weights. Finally, the rmse on leader board is 0.13357, ranks top 3623/9923 = 36.5%.

The code of our project is available at github.



mliw 1	b54d76c 2 hours ago	5 commits
data	1	2 hours ago
.Rhistory	1	2 hours ago
README.md	Create README.md	yesterday
Section1_feature_engineering.R	xgb	21 hours ago
Section1_feature_engineering.py	xgb	21 hours ago
Section2_lasso.R	1	2 hours ago
Section2_lasso_submission.R	1	2 hours ago
Section2_svr.R	1	2 hours ago

[*]Mingwei Li, Xinyu Leng, Hongjin Long and shuheng Huang are master students of economics, The University of Texas at Austin

Contents

1 Introduction

The problem we try to answer is **kaggle house price prediction problem**.

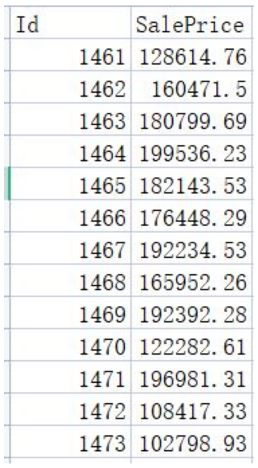
In real-world business, there are many situations where clients have to make a precise evaluation on house price. What they can observe are some house properties like area and quality. Some properties are listed in table 1.

Table 1: Part of House Properties of Kaggle Competition

Properties	Descriptions
MSSubClass	The building class
MSZoning	The general zoning classification
LotFrontage	Linear feet of street connected to property
LotArea	Lot size in square feet
...	...
SaleType	Type of sale
MiscVal	$Value of miscellaneous feature

Our ultimate goal is to build a model which can predict house price based on these properties. Such model is very important, as it can provide reasonable price to potential clients and help them promote their business.

As for the kaggle competition, we have train data whose dimension is 1460×81. 1460 houses with 81 different properties are included in our data. We would (1)Impute missing values. (2)Design new features. (3)Conduct cross-validation based on train data. After the best model is obtained, we fit it on the whole train data set. Such model can be used to predict on test data whose dimension is 1459×80. Then we get predicted prices of 1459 houses. Finally, we submit our prediction to kaggle and get our score. One advantage of kaggle is such competition enables us to compare our performance with other teams. Part of our final submission is like figure 1.



Id	SalePrice
1461	128614.76
1462	160471.5
1463	180799.69
1464	199536.23
1465	182143.53
1466	176448.29
1467	192234.53
1468	165952.26
1469	192392.28
1470	122282.61
1471	196981.31
1472	108417.33
1473	102798.93

Figure 1: Part of our final submission

2 Methods

2.1 Data Preprocessing

A part of original data is listed in table 2. The dimension of original data is 1460×81. We can see there is a lot of missing values(NA) and factor features. Our prediction target is log1p_SalePrice.

Table 2: Part of Original data

Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	...	log1p_SalePrice
1	60	RL	65	8450	Pave	NA	...	12.24769912
2	20	RL	80	9600	Pave	NA	...	12.10901644
3	60	RL	68	11250	Pave	NA	...	12.31717117
4	70	RL	60	9550	Pave	NA	...	11.84940484
5	60	RL	84	14260	Pave	NA	...	12.4292202
...	...	...	...	...	...	...	...	...
1459	20	RL	68	9717	Pave	NA	...	11.86446927
1460	20	RL	75	9937	Pave	NA	...	11.90159023

The procedure of data preprocessing can be summarized as follows:

(1) Imputing missing value with median value or mode value.

(2) Design new features.

(3) Delete outliers.

After we impute all missing values and design new features, R-package *dummies* is adopted to encode factor features into one-hot numerical variables. Finally, the dimension of train data becomes 1454×308, which means we have 1454 observations and 308 features.

2.2 Single Models

Three single models include Xgboost, SVM-Regression and Lasso Regression would be used to fit on the training data. We use repeated cross-validation instead of one-time cross-validation to find the best model parameter. Details of repeated cross-validation are listed below, we conduct cross-validation 20 times in order to make a precise evaluation of parameter performance.

Algorithm 1 Repeated Cross-Validation

```
1: Input $(param_1, param_2, ..., param_n)$
2: for $i = 1$ to $n$ do
3:     Use $param_i$ as model parameters
4:     Define $CV_i = 0$
5:     for $j = 1$ to 20 do
6:         Set the $random\_seed = j$
7:         Shuffle the data according to $random\_seed$
8:         Calculate the corresponding$CV\_error_j$
9:         $CV_i = CV_i + CV\_error_j$
10:    end for
11: end for
12: $param_i$ with the lowest $CV_i$ is the best parameter
```

Then we discuss the 3 single models:

1 Xgboost: According to Xgboost Package Introduction, Extreme Gradient Boosting, which is an efficient implementation of the gradient boosting framework from Chen & Guestrin (2016). This package is its R interface. The package includes efficient linear model solver and tree learning algorithms. The package can automatically do parallel computation on a single machine which could be more than 10 times faster than existing gradient boosting packages. It supports various objective functions, including regression, classification and ranking. The package is made to be extensible, so that users are also allowed to define their own objectives easily.

For simplicity, we can regard XGBoost as a tree model which tries to use a linear combination of indicative functions to approximate the relationship between X and Y.Based on real-analysis knowledge, linear combination of indicative functions can approximate all Borel-measurable functions. Therefore, Xgboost is very powerful in fitting data.

The details of Xgboost in our project is as follows.

Algorithm 2 Xgboost

```
1: Input $train\_data, test\_data, (param_1, param_2, ..., param_n)$
2: Use Xgboost fit on $train\_data$ to get the top 25 features
   Such features would be used for following repeated cross-validation
3: Use repeated cross-validation to find $param_{best}$ for Xgboost
4: Xgboost with $param_{best}$ would fit on $train\_data$
5: Fitted Xgboost would be used to make prediction on $test\_data$
```

Code of finding *param_best* is at here; Code of making prediction is at here.

2 SVM-Regression: According to e1071 Package Introduction, function SVM is used to train a support vector machine. It can be used to carry out general regression and classification (of nu and epsilon-type), as well as density-estimation. A formula interface is provided.

SVM firstly maps the original data to a different space through a kernel function. Such kernel function can be linear, polynomial, radial basis and sigmoid. Linear kernel performs the best in our project. Then, a hyperplane is adopted to divide the data into different groups with different mean values. The details of SVM in our project is as follows.

Algorithm 3 SVM

```
1: Input $train\_data, test\_data, (param_1, param_2, ..., param_n)$
2: Use Xgboost fit on $train\_data$ to get the top 25 features
   Such features would be used for following repeated cross-validation
3: Use repeated cross-validation to find $param_{best}$ for SVM
4: SVM with $param_{best}$ would fit on $train\_data$
5: Fitted SVM would be used to make prediction on $test\_data$
```

Code of finding *param_best* is at here; Code of making prediction is at here.

3 Lasso Regression: Unlike classical linear regression, lasso's optimization target is as follows. A L1 penalty term is involved.

$$\sum_{i=1}^{n}(y_i - \sum_{j} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

According to Glmnet Introduction, Glmnet is a package that fits generalized linear and similar models via penalized maximum likelihood. The regularization path is computed for the lasso or elastic net penalty at a grid of values (on the log scale) for the regularization parameter lambda. The algorithm is extremely fast, and can exploit sparsity in the input matrix X.

The details of Lasso in our project is as follows. We don't conduct feature selection this time, as Lasso can select features automatically.

Algorithm 4 Lasso

```
1: Input $train\_data, test\_data, (param_1, param_2, ..., param_n)$
2: Use repeated cross-validation to find $param_{best}$ for Lasso
3: Lasso with $param_{best}$ would fit on $train\_data$
4: Fitted Lasso would be used to make prediction on $test\_data$
```

Code of finding *param_best* is at here; Code of making prediction is at here.

2.3 Stacking

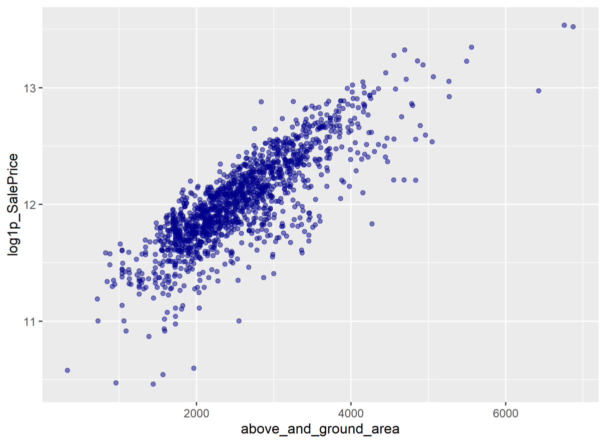
At this stage, we have 3 predictions from 3 different single models. Then we can mix these predictions with different weights to achieve a higher score(lower RMSE) in public leaderboard.

3 Results

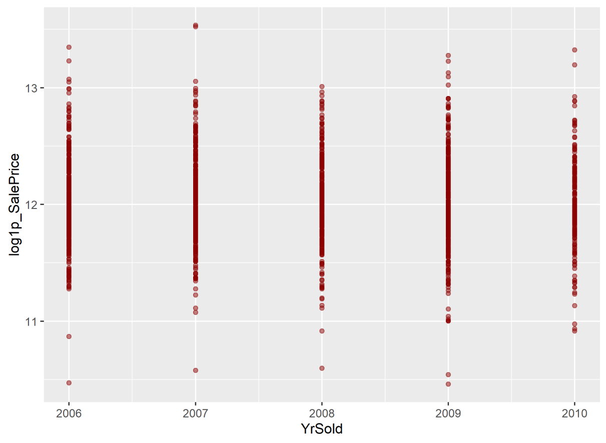
3.1 Feature Importance

Feature engineering determines the upper bound of our final prediction performance, and our model determines how close we can achieve this upper bound. If we can design a feature which has the same value as the prediction target log1p_SalePrice, then a single linear model can help us get a high score in public leaderboard.

Figure 2 demonstrates the comparison between best Feature and worst Feature. We can see the predictive power of above_and_ground_area.



(a) above_and_ground_area(best_feature)



(b) YrSold(worst_feature)

Figure 2: Comparison between best Feature and worst Feature

3.2 Xgboost

The results of repeated cross-validation of Xgboost are listed in table 3. The best parameter combination is $gamma\_value = 0, min\_child\_weight\_value = 0$.

Table 3: Results of repeated cross-validation of Xgboost

gamma_value	min_child_weight_value	cross-validation error
0	0	0.121804009
0	0.3	0.121804009
0	0.6	0.121804009
0	1	0.121804009
0.3	0	0.13818625
0.3	0.3	0.13818625
0.3	0.6	0.13818625
0.3	1	0.13818625
0.6	0	0.146007412
0.6	0.3	0.146007412
0.6	0.6	0.146007412
0.6	1	0.146007412
1	0	0.153047792
1	0.3	0.153047792
1	0.6	0.153047792
1	1	0.153047792

Xgboost with the best parameters is adopted to fit on the train data and make prediction. The RMSE on public leaderboard is 0.14181.

3.3 SVM-Regression

Different kernel functions are tried, and linear kernel is much better than others. The results of repeated cross-validation of SVM-Regression are listed in table 4. The best parameter combination is $eps\_value = 0.1, kernel\_type = linear$.

Table 4: Results of repeated cross-validation of SVM-Regression

eps_value	kernel_type	cross-validation error
0	linear	0.116743011
0.1	linear	0.116404929
0.2	linear	0.116454238
0.3	linear	0.116948547
0.4	linear	0.118046238
0.5	linear	0.120444927
0.6	linear	0.124259672
0.7	linear	0.128512672
0.8	linear	0.134190005
0.9	linear	0.14726593
1	linear	0.160303307

SVM with the best parameters is adopted to fit on the train data and make prediction. The RMSE on public leaderboard is 0.14150.

3.4 Lasso-Regression

The results of repeated cross-validation of Lasso-Regression are listed in table 5. $alpha = 0, lambda = 0.2$ are the best parameters.

Table 5: Results of repeated cross-validation of Lasso-Regression

alpha	lambda	RMSE
0	0	0.117741538
0	0.2	0.1158448
0	0.4	0.119725798
0	0.6	0.124441736
0	0.8	0.129306086
0	1	0.134140604
0.2	0	0.122860809
0.2	0.2	0.155784292
0.2	0.4	0.201218861
0.2	0.6	0.244992069
0.2	0.8	0.287642877

Lasso-Regression with the best parameters is adopted to fit on the train data and make prediction. The RMSE on public leaderboard is 0.13955.

3.5 Stacking Model

The public leaderboard RMSEs of 3 single models are listed in 6. We can see (1)The performances of these 3 models are very similar. (2)Lasso-Regression performs the best among the 3 models.

Table 6: Public Leaderboard RMSEs of 3 single Models

Model	Public RMSE
Xgboost	0.14181
SVM-Regression	0.1415
Lasso-Regression	0.13955

We notice that SVM-Regression with linear kernel is very similar to linear regression. Therefore, we try 2 model combinations.

Combination1:Lasso-Regression+Xgboost(table 7);Combination2:SVM-Regression+Xgboost(table 8).

Table 7: Combination1:Lasso-Regression+Xgboost

Lasso_weight	Xgboost_weight	Public RMSE
0.8	0.2	0.13546
0.6	0.4	0.13357
0.4	0.6	0.13398
0.2	0.8	0.13673

Table 8: Combination2:SVM-Regression+Xgboost

svr_weight	Xgboost_weight	Public RMSE
0.8	0.2	0.1391
0.6	0.4	0.13791
0.4	0.6	0.13795
0.2	0.8	0.13926

We notice weight combination (lasso_weight=0.6, Xgboost_weight=0.4) can achieve the lowest public RMSE of 0.13357, which ranks top 36.5%.

4 Conclusion

Due to time limitation, there are still many works which can be done to improve final score. The following are some places which can be improved:

1 Change a prediction target: According to figure 2, $above\_and\_ground\_area$ has a very strong predictive power on $log1p\_SalePrice$. We notice the fact that $above\_and\_ground\_area \times unit\_price = log1p\_SalePrice$. Therefore, we can change the prediction target to $unit\_price$ to make full use of the information contained in data.

2 Explore more features: Feature engineering determines the upper bound of our final prediction performance, and our model determines how close we can achieve this upper bound. If we can design a feature which has the same value as the prediction target $log1p\_SalePrice$, then a single linear model can help us get a high score in public leaderboard.

3 Conduct feature Selection: We have 301 features in this project. There must exist a subset of these 301 features which can minimize cross-validation score. Genetic Algorithm and Forward Algorithm can be used to solve this problem. In fact, we used to achieve top 10% in this house price competition with the help of python package deap.

4 Tune more parameters: In our project, we only tune a part of model parameters due to time limitation. Grid-Search would consume a lot of time and thus it's not suitable for this task. Bayesian optimization is a powerful tool to solve this problem. However, we don't find appropriate R package. We used to use Python package hyperopt to tune model parameters.