House Prices: Advanced Regression Techniques

Author: Jialiang Cui
Module: Advanced Machine Learning
Data: 22/4/2017

Summary

This project is an existing Kaggle competition. [1] The goal of the project is to apply advanced machine learning techniques to predict the housing price in Ames, Iowa. The project can be divided into three processes:

1. The first process is data cleaning and preprocessing. Through EDA (exploratory data analysis) and data cleaning (missing values, outliers, categorical values), data are encoded and better prepared for model training.
2. The second process is model construction. Feature engineering is implemented to better select the features and reduce noises. Multi ML approaches are applied including Random Forests, Lasso/Ridge Regression, Elastic Net Regularization, Gradient Boosting, Support Vector Machine. 5-fold Cross Validation is taken to measure the local performance. Lasso performs the best on cross validation. Ensemble models such as stacking is also applied. The final model is the ensemble model which takes the mean of 5 base models' predictions.
3. The last process is improving the model and feature re-engineering. Final performance can surely be improved by taking deeper researches on feature engineering and parameter tuning. Different ensemble methods are available for further improving the prediction.

The Python packages used in this project are Pandas, Numpy, Sklearn, Matplotlib and Seaborn. (Last two are mainly for data visualization)

[1] https://www.kaggle.com/c/house-prices-advanced-regression-techniques

Table of Contents

Summary **2**

Introduction **4**

Technical Content **4**

Before the model **4**

Data Cleaning & Preprocessing 4

Insights from Data 8

Model Construction **8**

Feature Engineering 8

Random Forests Decision Trees 10

Regularization Models 10

Support Vector Machines Regression 11

Ensemble Learning 12

Stacking 13

Ensemble- Simple Mean 14

Training and Performance **14**

Cross Validation Performance 14

Kaggle Leaderboard (Test Data) 15

Possible Improvements 15

Conclusion **16**

Reference **17**

Appendices **18**

Introduction

This Kaggle competition uses Ames Housing Dataset. Participants are given two datasets, “Train” and “Test”, both in csv format. “Train” contains 1460 observations and 79 variables; “Test” contains 1459 observations. The target of prediction is house price.

Errors are measured in RMSLE, which is calculated as

$$\epsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2}$$

where p is the prediction and a is the actual observation of sale price.

The original dataset requires many data manipulations before any learning algorithms can be applied. In addition, feature engineering such as feature extraction and construction is critical to the final predictive performance.

Several ML models are applied separately to compare the performance. Those with good cross validation scores and less correlation will be formed to an ensemble model. It can be the simple average of different base models or more complex aggregations such as stacking.

Technical Content

Before the model

Data Cleaning & Preprocessing

Data cleaning is essential for the whole analysis. The following cleanings are implemented in this dataset.

i. Remove outliers
ii. Convert variable if necessary
iii. Fill missing values
iv. Encode categorical variables
v. Log transforming/ Standardization

Exploratory Data Analysis

EDA is a good approach to help data cleaning and gain some insight for further feature engineering. It benefits the whole data cleaning process, especially in finding outliers, constructing features and transforming/standardizing. The main Python libraries for EDA purpose are Seaborn and matplotlib.

Outliers

Data cleaning here stars with finding outliers. Intuitively, housing price is affected by location and size. A plot on living area (ft^2) vs. price will be helpful to see any unusual data points. As shown in Figure 1, SalePrice and GrLivArea (ground living area square feet) mostly follow a linear pattern. However, there are two points having huge living area but cheap price. According to De Cock (2011, p.4), the author of the dataset, it is "suggested to remove any houses with more than 4000 square feet from the data set."[2]

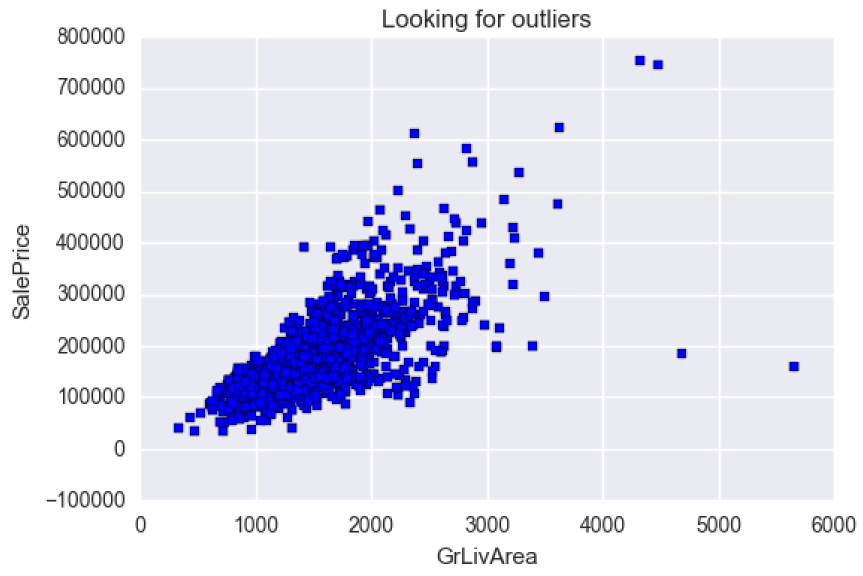


FIGURE 1 PRICE VS. AREA

Removing outliers is dealing with the observations of the dataset, or in other words, rows of the data frame. The next step is to take look at the columns of the data frame, or the features of dataset.

Change Feature Types:

Check feature types is very essential in data cleaning because many numerical features are indeed categorical. For instance, 'MSSubClass' (the building class) is displayed in number but it should be categorical. Under 'MSSubClass', 20 means '1-STORY 1946 & NEWER ALL STYLES' and 190 means '2 FAMILY CONVERSION - ALL STYLES AND AGES.' Failing to change the data type will cause errors in regression. Other examples are date variables such as

[2] De Cock, D., 2011. Ames, Iowa: Alternative to the Boston housing data as an end of semester regression project. *Journal of Statistics Education*, *19*(3).

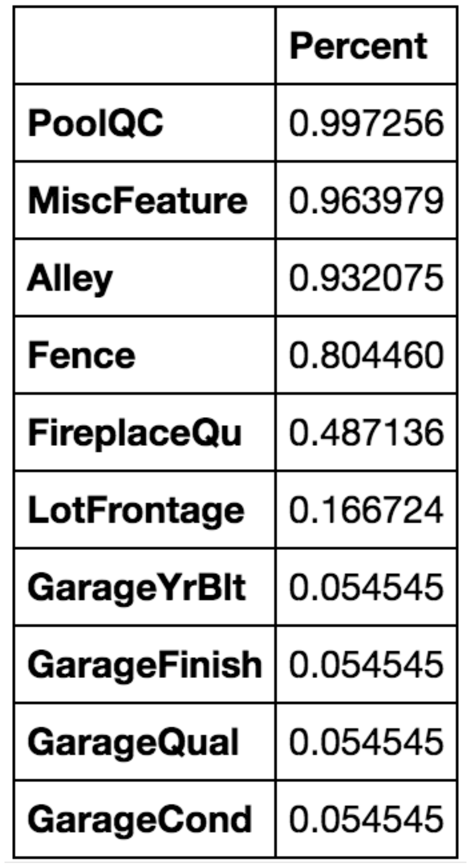
‘YearBuilt’ and ‘YrSold.’ A frequency table is helpful for this type of variables. It is found that there are lots of unique value under ‘YearBuilt’ and ‘YrSold.’ If they are changed to categorical variables, there will be huge number of dummies being created in further processing. The approach taken here is to take the difference from 2017 to transform them to measurable numeric features. The age of a house will affect its price; therefore, a numerical feature reflecting the house age would be appropriate.

Some other features containing many unique values need special construction. For instance, ‘MoSold’ contains 12 unique values since it stands for Month of Sold. It’s frequency table shows that most of its records are in summer time (May – Sep). Instead of having 12 unique values, this feature can be transformed to have only two values, summer or not. This operation can be easily achieved through list comprehension in Python.

Missing Values

To have a big picture of missing values usually comes before filling the hole. Pandas has the function to check missing values in data frame; df.isnull().sum() will show the sum of NaN in each column. The percentages of missing values for each feature are calculated here.

Figure 2 Percentage of missing values



	Percent
PoolQC	0.997256
MiscFeature	0.963979
Alley	0.932075
Fence	0.804460
FireplaceQu	0.487136
LotFrontage	0.166724
GarageYrBlt	0.054545
GarageFinish	0.054545
GarageQual	0.054545
GarageCond	0.054545

As shown in Figure 2, features like Pool Quality have more than 90% missing rate. Those features will only make noise to the regression.

Features with more than 20% missing rates are removed from the data. Then it is guaranteed that each column/feature only contains small number of missing values.

The next step is to fill the hole. Imputer function from sklearn.preprocessing is used here. Both numerical and categorical features have missing values; therefore, different approached are required.

1. Numerical missing values are replaced with the median of the exiting records of the column. The reason for using median is because some features are skewed, and taking the mean will not accurately represent the overall situation.

2. Categorical missing values are replaced with the most frequent value of the existing records.

Encode categorical variables

Categorical variables cannot be used in regression directly. They must be encoded to dummy variables. One-hot encoding is used here to create dummy variables. After encoding, 228 more features are created. Considering the sample size of 1456 (1460 in total but 4 outliers are removed), the concern of overfitting arises. One thing to state here is that the first dummy is not dropped here because the further test shows keeping all dummies will get better prediction. So, 10 dummies will be created if the feature has 10 unique values.

Log transformation and Standardization

Visualization of SalePrice and some numerical features show the importance of log transformation. Figue 3 and 4 shows obviously that SalePrice is right skewed. To make it follow normal distribution, log1p transformation is taken. Basically, x -> log(x + 1)

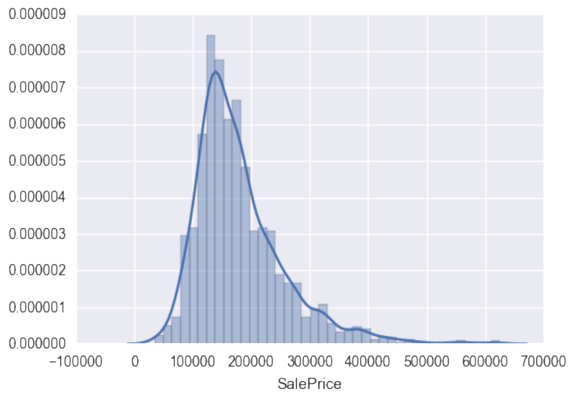


FIGURE 3 DISTRIBUTION OF SALE PRICE

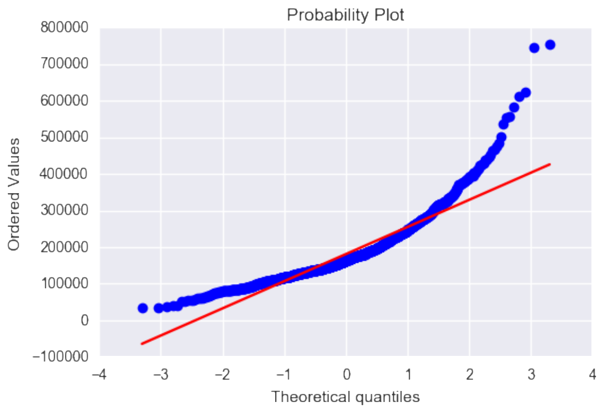


FIGURE 4 QQ PLOT OF SALE PRICE

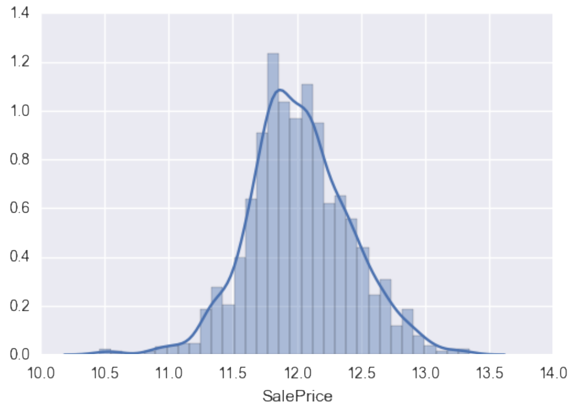


FIGURE 5 SALE PRICE AFTER TRANSFORM AREA

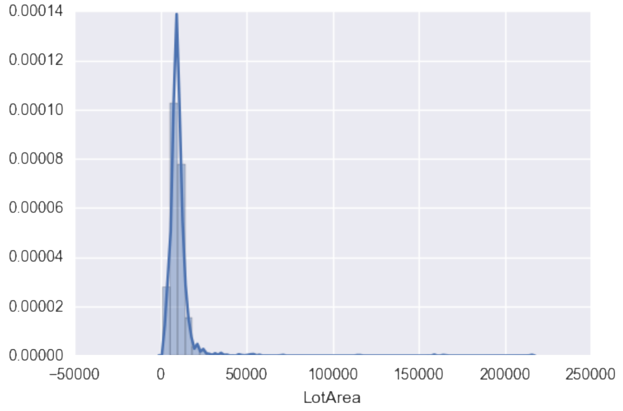


FIGURE 6 DISTRIBUTION OF LOT

Features such as 'LotArea' are skewed as well. The rule here is if the skewness is bigger than 0.75, log1p transformation is required.

If a dataset contains features in different scales, then standardization is necessary. In this case, many features are in different scales, for example, the age of the house and the number of bathrooms are not in the same scale level. After Z score standardization, all numerical features follow the distribution such that mean = 0 and s.d = 1.

Insights from Data

Although some routine cleanings are done, the data set is still not clean enough. Many related features can be combined. For example, 1st floor square feet, 2nd floor square feet and basement square feet can be added together to make a new feature for measuring the total square feet; the number of bathrooms above ground and in basement can be blend into just one feature for bathroom. Such manipulations can make the dataset more interpretative and reduce the dimensionality.

As discussed at the beginning of the report, there are commonly two important factors affecting the house price, location and size. Size of the house are reflected in many different aspects in the data set; however, it is tricky to quantify the location. 'Neighborhood' measures the location directly with 25 categorical values but it intuitively doesn't seem to be very effective. It is possible that some extra manipulations and constructions can be applied to location-relevant features.

Also, some features are ordinal. For instance, 'OverallCond' (Overall Condition) is measured in integer number from 1 to 10. Keep it as a numerical feature can possibly avoid losing information as well as reduce dimensionality. Here it is hot encoded into 10 dummy variables. ML models should be test to see which approach is better.

Model Construction

Feature Engineering

Feature selection is necessary in this dataset. With 1456 observations on hand, more than 200 features are likely to cause variability and make overfitting happen. Eliminating unnecessary features can lead to low variance at the tiny cost of increasing bias. (Hastie, 2013, p224)[3]

Basically, there are three ways to reduce the dimensionality of features:

1. Subset features
2. Regularization
3. Dimension reduction techniques such as PCA

Considering that the purpose is to predict house price, PCA is not implemented here because it will lose the interpretability. After PCA, it is difficult to get any insights on how specific feature from original dataset may affect the predictor. Therefore, subset selection of features and regularization are considered here.

[3] James, G., Witten, D., Hastie, T. and Tibshirani, R., 2013. *An introduction to statistical learning* (Vol. 6). New York: springer.pp.224

Correlation analysis will be helpful for studying the features. A heat of correlations is intuitive and informative. According to figure 7, several features share with very high correlations.

1. GarageYrBlt and YearBuilt are highly correlated
2. GarageCars and GarageArea are highly correlated
3. TotRmsAbvGrd and GrLivArea are highly correlated
4. TotalSF and GrLivArea are highly correlated
5. LotArea and LofFrontage are highly correlated

These findings are not surprising. Garage is usually built with the house; garage size determines how many cars can be park in; TotalRmsAbvGrd, TotalSF and GrLivArea are all measuring the size of the house. Due to the potential multicollinearity, these variables should not be included in the same model.

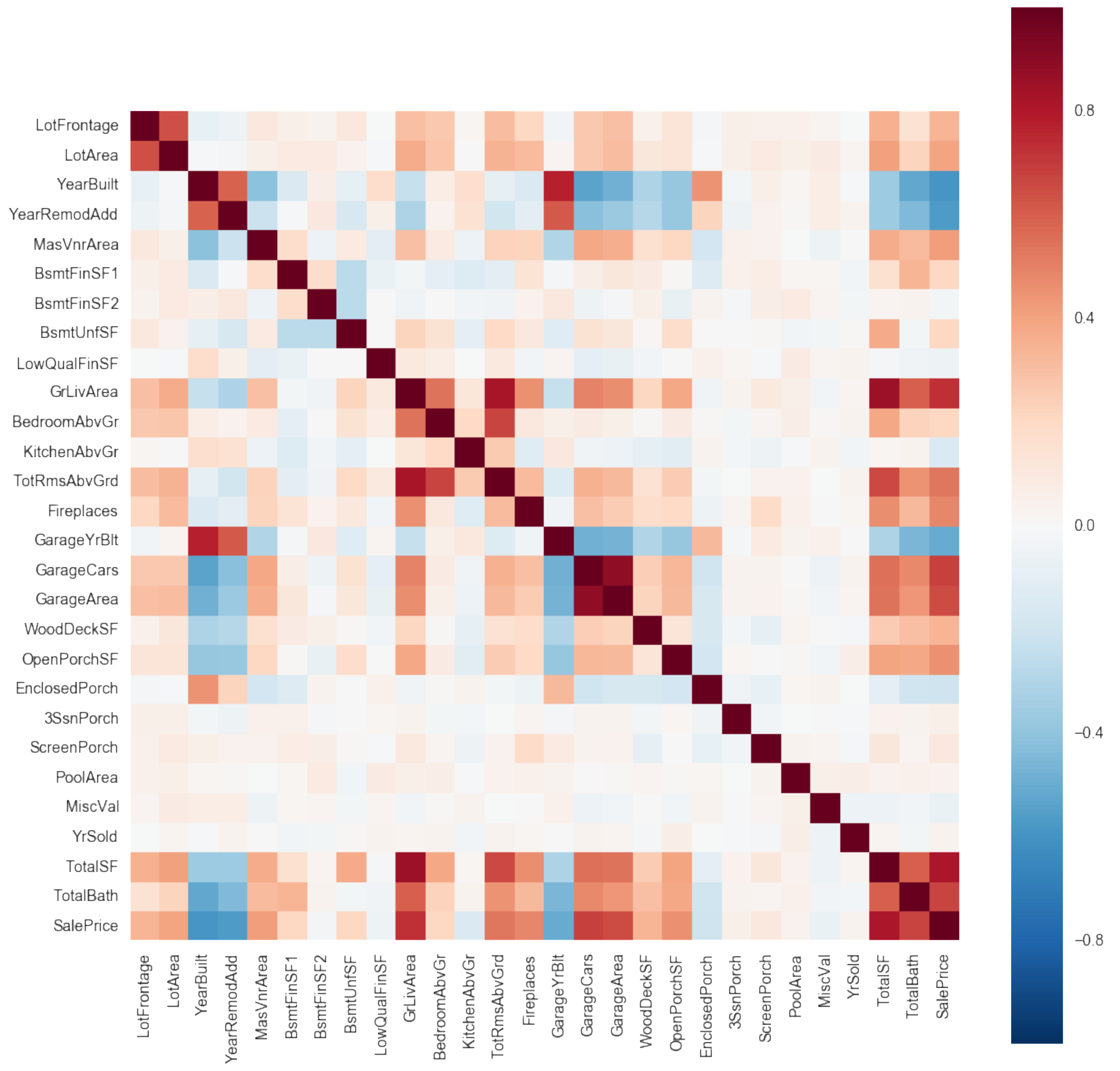


FIGURE 7 HEAT MAP OF CORRELATIONS

Random Forests Decision Trees

In addition to the correlation with the predictor, there are other ways to measure the feature importance. Random Forests provide a very useful function on feature selection. In sklearn, feature_importance can be called in tree models to see the feature importance. The metrics used in feature importance is Gini impurity.

5 different random seeds are generated first. Then Random Forests runs on each seed and compute the feature importance. The mean of 5 feature importance scores will be the final importance and features are ranked by this score.

Figure 8 shows the top 10 features according to the correlation matrix and according to feature importance in Random Forests. Two rankings share almost the same features. Size of area and the age of house are two main factors here.

Rank	Correlation	Feature Importance
1	TotalSF	TotalSF
2	GrLivArea	YearBuilt
3	GarageCars	YearRemodAdd
4	TotalBath	GarageArea
5	GarageArea	GrLivArea
6	ExterQual_TA	BsmtQual_TA
7	YearBuilt	ExterQual_TA
8	GarageFinish_Unf	TotalBath
9	YearRemodAdd	LotArea
10	BsmtQual_TA	GarageCars

FIGURE 8 CORRELATION VS. FEATURE IMPORTANCE

In total, the training set has 297 features. After dropping highly correlated features and features with 0 feature importance, the subset of features contains 172 features. It is unclear if dropping features benefits the prediction at this stage; so, two sets of features are both kept. Pre-picking features may not be very effective for regularization and tree-based models. But it is still useful to get insight from the dataset. In this case, several ML algorithms will run on both sets of features to compare the performance.

Regularization Models

Regularization is another popular method to reduce dimensionality. Three regularization algorithms are implemented here: Ridge, Lasso and Elastic Net.

Ridge Regression
Ridge regression have the coefficient values which minimizes the following:

$$\mathrm{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2,$$

By assigning different λ (it is called alpha in sklearn) values, the model can have different effects of shrinkage penalty. 10-fold-cross validation selects alpha = 5. Among 296 features, ridge shrinks only 1 feature's coefficient value to 0.

Lasso Regression
Lasso is similar with ridge in terms the goal of shrinkage penalty. Lasso minimizes the following:

$$\mathrm{RSS} + \lambda \sum_{j=1}^{p} |\beta_j|.$$

Compared to Ridge, Lasso can force coefficients values to 0. Therefore, more features will be "eliminated" and Lasso performs like a feature selection tool. In this case, 10-fold-cross validation on training set selects alpha = 0.0003. Among 296 features, 157 features' coefficients are 0. The performance of Lasso is slightly better than Ridge.

However, Lasso has its disadvantages. According to Zou and Hastie, "If there is a group of variables among which the pairwise correlations are very high, then the lasso tends to select only one variable from the group and does not care which one is selected" (Zou, Hastie 2015, p304). [4] This may have an impact on the prediction because the situation mentioned does exist in this house price dataset.

Elastic Net
Elastic Net Regularization is more like a combination of Ridge and Lasso. It minimizes the following:

$$\mathrm{RSS} + \lambda_2 \|\beta\|^2 + \lambda_1 \|\beta\|_1$$

It is obvious that Elastic Net can be converted to Lasso or Ridge with certain λ. In sklearn, the main hyper parameters in Elastic Net model are alpha and l1 ratio. The penalty is equal to a*(L1 penalty) + b*(L2 penalty), and alpha = a + b, l1 ratio = a/ (a + b)

10-fold CV gives alpha = 0.0005 and l1 ratio = 0.5. 147 coefficients estimates are 0 after shrinkage (10 less than Lasso). Elastic Net performs slightly worse than Lasso but better than ridge on CV.

Support Vector Machines Regression

Although Support Vector Machine is mainly used for classification problem, it is still applicable for regression problem. Support Vector Regression is no different from SVM's application on classification problem. The goal is to find the optimal generalization bounds. If the prediction

[4] Zou, H. and Hastie, T., 2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *67*(2), pp.301-320.

falls within the tube (the distance from the actual value is no bigger than epsilon), it will not contribute to the loss function. SVR is also capable to handle non-linear regression with the help of kernel. There are three parameters correlated to the model, C, the penalty parameter, epsilon, the distance of tolerance, and the kernel. The performance of the model depends heavily on parameter settings. In this case, C = 1, epsilon = 0.05, kernel = default 'rbf.'

SVR performs slightly worse than regularization models.

Ensemble Learning

Ensemble learning is a good way to combine weak learners to produce strong learners. This 'crowdsourcing' idea is highly practical and beneficial in improving the prediction. The Random Forest method implemented before is a type of ensemble learning. The core concept in Random Forest is bootstrap aggregating, or Bagging. Boosting is another popular method of ensemble learning. Two boosting algorithms are applied here.

Adaptive Boosting

Boosting is an improvement of the decision tree. It is the method which allows the weak learners to focus more on prediction errors from the previous weak learners to incrementally increase the model accuracy. Adaptive Boosting, or AdaBoost, is expected to minimize the exponential loss function. At the beginning, each sample in the training set will be assigned an equal weight; then regressor will be trained on the training set. Based on the error, each sample will be assigned to different weights at each iteration. The descendant regressors will be more focusing on samples which have high error in previous regressions.

Like Random Forests, AdaBoost deals with generalization error well and does not require too much parameter tuning. Since it is the ensemble of many weak learners, the final model variance is lower; and the variance is decreased because each regression is the forced improvement of the previous iteration.

The base model in AdaBoost can be Decision Tree or other models. It is encouraged to use weak regressors because it is the adaptive learning.

Gradient Boosting

Gradient Boosting performs well in supervised regression problems and is one of the most popular ML algorithms in Kaggle competitions. The base model of Gradient Boosting is decision tree. Each regression tree in Gradient Boosting model tries to reduce the incremental residuals of trees from previous iterations. The concept is different from AdaBoost in which weights of samples are changed according to prediction errors. The loss function used here is 'huber', which is the combination of least square error and least absolute deviation. Since GB is robust to overfitting, the number of boosting stages are set to a big number, which is 3000 in this case. The other parameters including learning rate are determined by grid search.

The final performance on CV is in middle of all models. GB is supposed to be the highest performing method; the poor performance may result from parameter tuning. In this case, only simple grid search is implemented to find number of estimators and learning rate.

Stacking

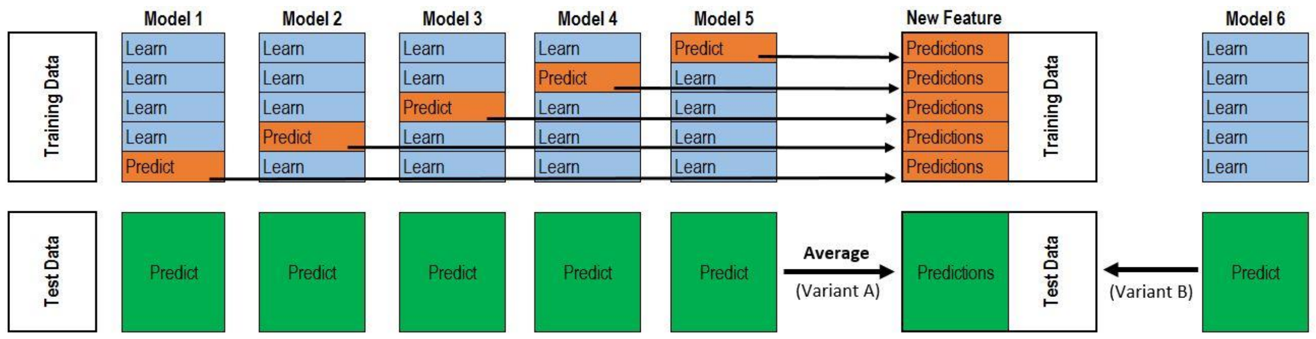


FIGURE 9 STACKING[5]

The general idea of stacking is to train a model using other models' predictions as features. This allows more information being exploited by model and the generalization errors will be reduced.

In this case, there are two layers of models: the first layer contains 5 different base models, and second layer is the model for final training/predicting. Figure 9 shows a 5-fold stacking using 5 base models.

- The training data are divided into 5 folds. In each iteration, 4 out of 5 folds will be used for training each base model and the rest fold will be used for prediction.
- In addition, each base model will also make prediction on the whole test data.
- As shown in figure 9, each model will predict on holdout fold of the training data at each iteration. After all iterations, 5 base models will have 5 predictions on training data. Those predictions will be the new training data for second layer model.
- Suppose the original training data have 1000 rows. Then the new training data will have 1000 rows and 5 columns. (Each column is base model; each row is base model's prediction on training data). 5 different models' predictions will become the new features.
- Base models' predictions on the test data (green block) will form a new test dataset by taking the average.
- The second layer model will be trained on the new training set (formed by base models' predictions on original training data), and the prediction will be made on the new test data. This prediction will be the final output.

To reduce the variance, base models should be optimally less correlated. However, the correlations between models are all very high (AdaBoost and Lasso share the lowest correlation, which is 0.94). As a result, stacking may not be able to boost the variance dramatically here.

[5] Zhang, Linghao. "How to Rank 10% In Your First Kaggle Competition".2016. Web.

Ensemble- Simple Mean

The last model is to take the mean of different models' predictions. It follows the basic idea of ensemble learning. If the base models are less correlated, the bias will be lower. Five best performing single models are picked, GBM, Lasso, Ridge and Elastic Net). The final model performs the best on CV. In classification problem, the final classification will follow the simple majority vote.

There are huge number of ensemble methods. Bagging, boosting, stacking, etc. They are all possible to be combined. In this project, stacking only uses two layers; multi layers and more complicated model constructions are possible to boost the final performance.

Training and Performance

Cross Validation Performance

10 models are test on 5-fold cross validation on the training set. The performance is following:

	All features	Subset of features	Difference
Random Forest	0.141388078	0.140555658	0.00083242
SVR	0.115575385	0.115301671	0.000273714
Ridge	0.113608303	0.113139646	0.000468657
Lasso	0.111621074	0.109972619	0.001648454
Gradient Boosting	0.114297137	0.114935335	-0.000638198
Elastic Net	0.111996584	0.111803629	0.000192955
AdaBoost	0.176308371	0.178416284	-0.002107913
Stacking 1	0.110190236	0.110040106	0.00015013
Stacking 2	0.110044457	0.109942939	0.000101518
Ensemble Mean	0.109399726	0.109544306	-0.00014458

Random state = 42

Errors are measured in RSLME

- AdaBoost uses Decision Tree as base model
- Stacking 1 uses Gradient Boosting, Random Forests, Lasso, SVR and Elastic Net as base models and the second layer training model is Linear Regression.
- Stacking 2 uses Gradient Boosting, Lasso, Ridge and Elastic Net as base models and the second layer training model is Linear Regression.
- Ensemble Mean uses Gradient Boosting, SVR, Lasso, Ridge and Elastic Net as base models and the final prediction is the mean of 5 models' predictions.

As the table shows, the performances on whole features and subset of features do not make big differences. Therefore, the conclusion is not to pre-pick any features when doing final predictions.

In general, regularization models and gradient boosting perform well. Ensemble models are better than any single model. The best model is the ensemble mean model.

Kaggle Leaderboard (Test Data)

The following models are test on the test set and submit to Kaggle:

Model	Public LB
SVR	0.13192
Lasso	0.13141
Gradient Boosting	0.12905
Elastic Net	0.13082
Stacking 1	0.12714
Stacking 2	0.12818
Ensemble Mean	0.12694

Errors are measured in RSLME

The best model is still ensemble mean. Two stacking models and elastic net have the same score. Compared to local CV, the RSLME on test data is a lot bigger.

Kaggle Leaderboard provides a signal showing if the model is overfitting in training set. The comparison between local CV and Leaderboard shows that Lasso makes the biggest difference. It may suggest that model is overfitting, or CV process may require double check.

However, Kaggle Leaderboard uses 50% of the test data to calculate the score. Therefore, the score on LB does not completely represent the model performance on test data. It is a worth-debating topic on which score is more trustable, public LB or local CV. One approach is that if both local CV and LB scores go with the same direction, the effort is on the right track.

Possible Improvements

Since Kaggle provides real data, there are lots of uncertainties on dataset. For example, training and testing data may follow different distributions. Testing data could have some missing values and outliers. However, there are still many ways to improve the model performance.

Feature construction/engineering is critical to the final perdition accuracy. A consensus is that good features determine the upper limit of the final accuracy. There are big numbers of improvements can be applied to feature engineering stage. A better approach may be to look at each of the features and do some analysis within features:

1. some features can be grouped together using k-mean clustering.

2. Location relevant features such as ‘Neighborhood’ can be encoded in different ways.
3. Polynomials of features can capture the diminishing effect. For instance, the square of the house age will show that age’s effect on price is not always constant. A 50-year-old house does not differ too much from a 60-year-old house in price; but a 10-year-old house will differ a lot from a 20-year-old house.
4. Dependent features need special analysis. Some features are dependent with each other but they are not easy to be detected by correlation analysis. One feature can be the condition of another. It is worth considering how to deal with dependent features.

Domain knowledge usually is more helpful than purely data-driven approaches. From a practical perspective, surprisingly, the simplified and essentially not data-driven strategy seems to perform well, at least for prediction purpose (Genuer, 2010, p2236)[6]. Visualization should be relied heavily for feature study.

Ensemble learning should still be the correct approach to go.

1. One possible improvement may be resembling different models using different features. It happens a lot that performance gets worse after careful feature construction. In response to this risk, different sets of features (before and after construction) can be put into ensemble model using different learning algorithms.
2. Another possible improvement is to set different random seeds to the ensemble model. This is like the bagging techniques. By setting different seeds, the training sample will be different in each iteration. The ensemble model can be trained on those different training sets and final prediction can be the mean of different predictions from different seeds.

Tuning parameters is also critical to the success. Some models will differ a lot in performance with different parameter settings. Choosing the best parameters is not an easy job. It requires lots of experiences and the insights from the model. Simple grid search may not work well and the computational time will be too long if there are lots of hyper parameters.

Conclusion

The house prices prediction is a typical supervised learning problem on regression. A huge number of algorithms can be applied to this problem. Regularization models seem to work well in linear regression problem and they can also reduce the dimensionality effectively. Tree-based regression models can also be powerful once hyper parameters are selected properly. An ensemble model which takes the mean of base models’ predictions performs the best on cross validation.

The main takeaways from the project are following:

[6] Genuer, R., Poggi, J.M. and Tuleau-Malot, C., 2010. Variable selection using random forests. *Pattern Recognition Letters*, *31*(14).

1. Create good pipelines and workflow. Data cleaning and model testing are the most time-consuming parts. It is often the case to go back to the data cleaning stage. It is important to have a workflow to define each stage of work and have a pipeline to efficiently execute.
2. Features require more time than models. Good models cannot save bad features but optimized features can make moderate models perform well.
3. Make results reproducible. Usually codes will be changed and re-formatted a lot; some results are not consistent after changes, which make it very difficult to compare the model performance.

For more detailed data analysis, please look at my GitHub repository. (https://github.com/louisvitamin/Kaggle-Housing-Price/blob/master/HousePriceRegression.ipynb)

Reference

Albon, Chris. "Feature Selection Using Random Forest - Machine Learning". Chrisalbon.com. N.p., 2017. Web.

De Cock, D., 2011. Ames, Iowa: Alternative to the Boston housing data as an end of semester regression project. *Journal of Statistics Education*, *19*(3).

Genuer, R., Poggi, J.M. and Tuleau-Malot, C., 2010. Variable selection using random forests. *Pattern Recognition Letters*, *31*(14), pp.2225-2236.

James, G., Witten, D., Hastie, T. and Tibshirani, R., 2013. *An introduction to statistical learning* (Vol. 6). New York: springer.pp.224

"Kaggle Ensembling Guide | Mlwave". Mlwave.com. N.p., 2017. Web.

Zhang, Linghao. "How to Rank 10% In Your First Kaggle Competition". dnc1994.com. N.p., 2016. Web.

Zou, H. and Hastie, T., 2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *67*(2), pp.301-320.

Appendices

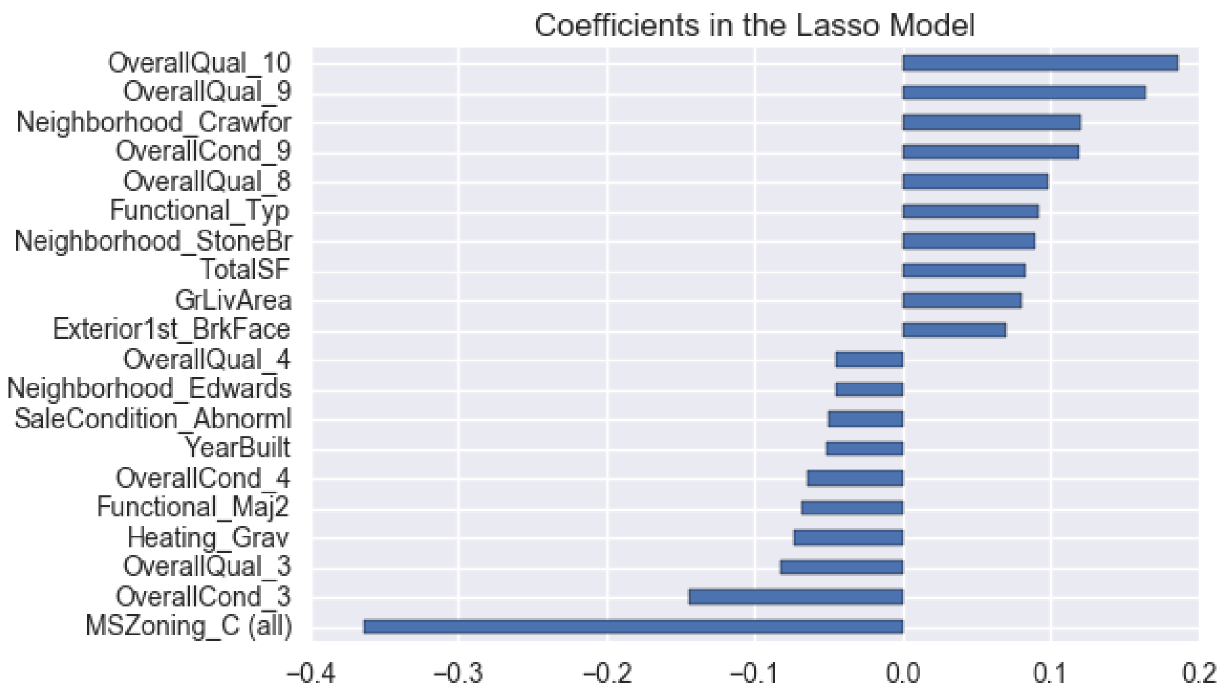
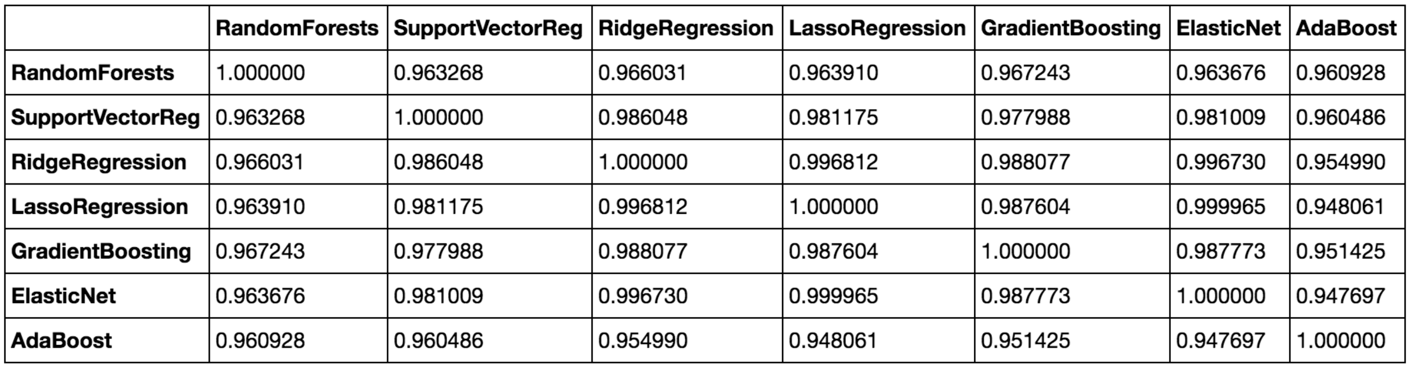


FIGURE 10 LASSO COEFFICIENTS (TOP AND BOTTOM 10)



	RandomForests	**SupportVectorReg**	**RidgeRegression**	**LassoRegression**	**GradientBoosting**	**ElasticNet**	**AdaBoost**
RandomForests	1.000000	0.963268	0.966031	0.963910	0.967243	0.963676	0.960928
SupportVectorReg	0.963268	1.000000	0.986048	0.981175	0.977988	0.981009	0.960486
RidgeRegression	0.966031	0.986048	1.000000	0.996812	0.988077	0.996730	0.954990
LassoRegression	0.963910	0.981175	0.996812	1.000000	0.987604	0.999965	0.948061
GradientBoosting	0.967243	0.977988	0.988077	0.987604	1.000000	0.987773	0.951425
ElasticNet	0.963676	0.981009	0.996730	0.999965	0.987773	1.000000	0.947697
AdaBoost	0.960928	0.960486	0.954990	0.948061	0.951425	0.947697	1.000000

FIGURE 11 MODEL CORRELATIONS

```
class Ensemble(object):
    def __init__(self, n_folds, stacker, base_models):
        self.n_folds = n_folds
        self.stacker = stacker
        self.base_models = base_models
    def fit_predict(self, X, y, test):
        X = np.array(X)
        y = np.array(y)
        test = np.array(test)
        folds = list(KFold(n_folds=self.n_folds, shuffle=True, random_state=42))
        S_train = np.zeros((X.shape[0], len(self.base_models)))
        S_test = np.zeros((T.shape[0], len(self.base_models)))
        for i, clf in enumerate(self.base_models):
            S_test_i = np.zeros((T.shape[0], len(folds)))
            for j, (train_idx, test_idx) in enumerate(folds.split(X)):
                X_train = X[train_idx]
                y_train = y[train_idx]
                X_holdout = X[test_idx]
                # y_holdout = y[test_idx]
                clf.fit(X_train, y_train)
                y_pred = clf.predict(X_holdout)[:]
                S_train[test_idx, i] = y_pred
                S_test_i[:, j] = clf.predict(test)[:]
            S_test[:, i] = S_test_i.mean(1)
        self.stacker.fit(S_train, y)
        y_pred = self.stacker.predict(S_test)[:]
        return y_pred
```

FIGURE 12 STACKING CODES IN PYTHON