Group 24 -- House Prices Prediction Final Report

Botao Li, Qili Zeng, Wenbin Teng, Ziran Min
{tonyli, qzeng, wbteng15, minziran}@bu.edu

1. Project Task

The goal of this project is to build Machine Learning models to predict the sale prices of houses in Ames, Iowa given the data sets (year 2006-2010) from Kaggle.

The difficulties of this project include cleaning 80 original variables, building effective algorithms by ourselves, finding best parameters for models, and ensembling models to reach final high score standing in Kaggle Leaderboard.

2. Related Work

From Kaggle Kernels, we have read through some project reports written by people in the past. In this way, we can quickly get a general sense of what the process of our project should be. In Pedro Marcelino's "Comprehensive Data Exploration with Python", we get an idea of how to do Exploratory Data Analysis and Feature Engineering for our house dataset. In Serigne's "Stacked Regressions: Top 4% on LeaderBoard", we learn some feasible regression models and model ensembling method.

3. Approach

3.1 Linear Regression

Minimize cost function:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

- Method 1: Gradient Descent

$\theta_j := \theta_j + \alpha\sum_{i=1}^{m}(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$ for $j = 0,\dots,n$

- Method 2: Normal Equation

$$\theta = (X^TX)^{-1}X^Ty$$

- Method 3: scikit-learn library

3.2 Ridge Regression

Minimize cost function:

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}{\theta_j}^2\right]$$

- Method 1: Gradient Descent

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_j := \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

for m$j = 1,\dots,n$

- Method 2: Normal Equation:

$\theta = (X^TX + \lambda I')^{-1}X^Ty, \lambda > 0, I' = diag(0,1,1,\dots,1)$

- Method 3: scikit-learn library

3.3 Lasso Regression

Minimize cost function:

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}|\theta_j|\right]$$

- Method 1: Gradient Descent

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_j := \theta_j(1 - \alpha\frac{\lambda}{2m}sgn(\theta_j)) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

for $j = 1,\dots,n$

- Method 2: scikit-learn package

3.4 Elastic Net Regression

Minimize cost function:

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda_1\sum_{j=1}^{n}{\theta_j}^2 + \lambda_2\sum_{j=1}^{n}|\theta_j|\right]$$

- Method: scikit-learn package

3.5 Random Forest Regression:

Because we have many dummy variables and label encoded ordinal variables among the total 202 features, it's a good idea to use tree-based models. Therefore, we decide to use random forest for regression. However, it is really difficult for us to build decision trees and calculate information gain by programming. We import random forest regression function from scikit learn.

3.6 Support Vector Regression:

The Support Vector Regression (SVR) is a regression model that uses the same principles as the SVM for classification, with only a few minor differences. In SVM, it is expected that samples can be classified with largest interval between different categories. However, as a regression model, SVR expects samples to be close to the fitted curve as much as possible, which requires all the samples, under ideal conditions, fall into a certain interval surrounding the curve with ε -deviation at most.

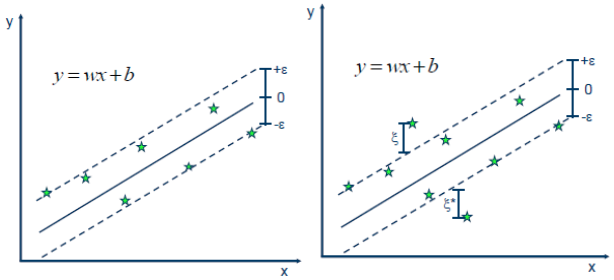

Figure 1 Comparison between SVM (left) and SVR (right)

Analogously to the "soft margin" loss function which was adapted to SVM, SVR is introduced with slack variables ξ_i, ξ_i^* to cope with otherwise infeasible constraints of the optimization problem. Hence we arrive at the formulation stated in [3]:

$$\min\ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{\ell}\left(\xi_i + \xi_i^*\right)$$

$$s.t.\begin{cases} y_i - < w, x_i > -b \le \varepsilon + \xi_i \\ < w, x_i > +b \le \varepsilon + \xi_i^* \\ \xi_i + \xi_i^* \ge 0 \end{cases}$$

which can be also converted into corresponding dual form by Lagrange Multiplier and Kernel method:

$$L(a,\hat{a}) = -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(a_n - \hat{a}_n)(a_m - \hat{a}_m)k(x_n, x_m)$$
$$-\varepsilon\sum_{n=1}^{N}(a_n + \hat{a}_n) + \sum_{n=1}^{N}(a_n - \hat{a}_n)t_n$$
$$s.t.\ \sum_{n=1}^{N}(\alpha_n - \hat{\alpha}_n) = 0,$$
$$0 \le \alpha_n, \hat{\alpha}_n \le C$$

where $k(\square)$ is the kernel function and in most cases, RBF kernel is chosen. For dataset with extremely large feature dimension, linear kernel will be adopted.
Thus the prediction can be made by

$$y(\mathbf{x}) = \sum_{n=1}^{N}(a_n - \hat{a}_n)k(\mathbf{x}, \mathbf{x}_n) + b$$

What is discussed above is called epsilon-SVR, which is actually used in our project. Another kind of SVR, ν-SVR whose parameter ν controls the number of support vectors, is not used in our project since ν is not intuitively understandable. Our experiment is based on our implementation of this model.

3.7 XGBoost:

XGBoost is a kind of improved Gradient Boosted Decision Trees (GBDT) model. Comparing to his predecessors, XGBoost utilizes 2-order Taylor expansion of the loss function and thus provides better optimization. Regularization for both tree volume T and leaf weight w are also introduced to control the complexity of the model and reduce the variance for over-fitting prevention.

$$\mathrm{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k),\ \Omega(f) = \gamma T + \frac{1}{2}\lambda \| w \|^2$$

$$\mathrm{L}^{(t)} = \sum_{i=1}^{n}\left[g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2$$

XGBoost learns from Random Forest and adopt column subsampling techniques to reduce computation and improve fitting performance, which is different from GBDT. XGBoost is also able to handle incomplete dataset and learns the splitting direction automatically for those missing data. However, due to Feature Engineering, all the missing data in our project has been processed, so XGBoost model does not demonstrated its effectiveness in this area here. Our experiment for XGBoost is based on open-source project xgboost since our own implementation works not well.

3.8 Stacking (Ensemble Method):

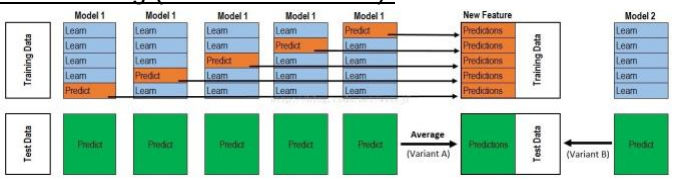

Figure 2 Illustration of Stacking procedure

Stacking (also called meta ensembling) is a model ensembling technique used to combine information from multiple predictive models to generate a new model. Previous literature demonstrates the stacked model will outperform each of the individual models in most cases, and at least perform as good as the best base model [5]. Stacking is most effective when the base models are significantly different, which makes it a proper computing structure for this group project.

Every base model in the stacking structure will be trained through K-fold strategy and make prediction for both testing set and the rest 1/K training set. Predictions of all the K parts of training set will be concatenated and form the prediction of the whole training set. Predictions of testing set will be averaged and considered as the representation of testing set.
Predictions generated by each base model will be put together with training set prediction serving as the input of 2nd-layer model for training and testing set prediction used for final regression.

4.Dataset and Metric

Training set and testing set are provided by Kaggle. There are 1460 and 1459 rows in each set. The training set has 79 independent variables and 1 dependent variable (Sale Price). After removing outliers, there are 1456 rows left in training set.

Training	Validation	Testing	Total
1092	364	1459	2915

We use training set to build models. To evaluate each model, we use 4-fold cross validation to compute the root mean square error (RMSE) between the logarithm of predicted sale price and logarithm of observed sale price based on the training set. After we build our first model, linear regression model, we will use its RMSE as the baseline to compare the score of other models

We combine training and testing sets as one to implement data cleaning and feature engineering. By building correlation map of all original numerical variables, we find some having very high correlation with the Sale Price are: Above Ground Living Area, Basement Area, Size of Garage in Cars Capacity, Overall Quality, and Original Construction Year Built. We will check whether they are really important features or not after building several models. We also drop some independent variables that are highly correlated (r > 0.8) with others to reduce multicollinearity.

After filling missing values for different variables case by case, computing log transformation for all skewed continuous variables, label encoding for ordinal variables, and one-hot encoding for categorical variables, we clean 79 original independent variables into 202 features (plus 1 for sale price) for modelling.

5. Results

For all the models we build, we use the training set (1456 * 202) and testing set (1460 * 202) that we clean in the previous feature engineering step.

We rank all individual models by their Cross-Validation RMSEs (CV RMSEs) if we can compute them. XGBoost Regression and Ridge Regression are the top two best single models we have.

Model	Score (CV RMSE)
XGBoost Reg (xgboost)	0.118954
Ridge Reg (sklearn)	0.121036
Linear Reg (sklearn)	0.126783
Support Vector Reg (our)	0.128861
Ridge Reg (our normal eqtn.)	0.131965
Random Forest Reg (sklearn)	0.137224
Lasso Reg (sklearn)	0.142892
Elastic Net Reg (sklearn)	0.223191
Linear Reg (our grad. descent)	0.826657
Lasso Reg (our grad. descent)	0.826661
Ridge Reg (our grad. descent)	0.826686

The followings are results and analysis for specific models.

5.1 Linear, Ridge, Lasso Regression:

Because our project is to predict the house price which is a continuous variable, the first model we want to build is Linear Regression. If we put all 202 features into one model, the problems of overfitting and multicollinear may affect the model performance, so we also build Ridge and Lasso Regression by adding norm 2 and norm 1 penalty terms respectively in the cost function.

To minimize the cost functions, we firstly try to further develop the gradient descent code in programming assignment 1. However, the simple gradient descent methods give us RMSEs that are above 0.82 for all Linear, Ridge, and Lasso Regression. That is much higher than the scores of any other models. Therefore, the gradient descent method we learned in lecture doesn't work good in our project. One reason might be that the 202 features make the dimension of cost function so high that the local minimum fund by gradient descent is not close to the global minimum.

Then, we decide to directly compute the normal equation for the solution of coefficients to minimize cost function. We easily compute the normal equations for Linear Regression and Lasso Regression, but not for Lasso Regression because of the mathematical difficulty of calculating norm1 matrix.

Surprisingly, our Ridge Regression algorithm by normal equation can perform as good as with other models by imported libraries (RMSE ≈ 0.13). However, when we try to get the CV RMSE for Linear Regression, the error of uninvertible singular matrix occurs. Similar with what we learn in lectures, we indeed find the complexity of computing normal equations.

Since both gradient descent and normal equation have obvious disadvantages, we also import the function for

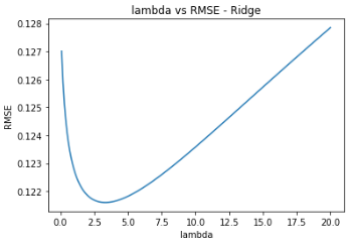

Figure 3

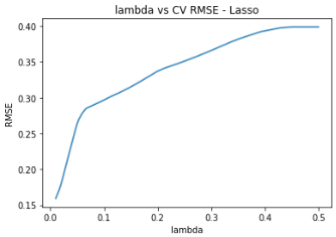

Figure 4

Linear, Ridge, Lasso Regression from scikit-learn library. By using sklearn functions, we would like to analyze how penalty parameter lambda affect the CV RMSE of the model.

In Figure 3 and 4, we can see how RMSE changes as lambda changes in two models. In Ridge Regression, at lambda ≈ 3.3, the model can reach a minimum of RMSE. We regard lambda = 3.3 as the optimal parameter in Ridge. However, we can't find optimal lambda for Lasso, and Figure 4 implies that as long as we add norm1 penalty in cost function, the model will never be better. Another

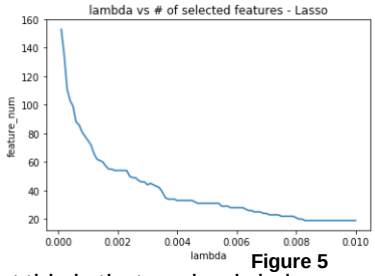

Figure 5

way to interpret this is that, as lambda increases, the error of bias increases faster than the error of variance decreases, so the total error increases.

Even so, we still want to use Lasso Regression because it can penalize many feature coefficients to zero and reduce the number of useful features. From Figure 5, we would like to set lambda = 0.005 to reduce feature numbers to about 30.

5.2 Elastic Net Regression:

Elastic Net Regression is the combination of Ridge of Lasso. It doesn't work so well (RMSE ≈ 2.2) in our dataset

and we think the reason is the same as the lambda of Lasso illustrated in Figure 4.

After we use whole training set to fit Ridge, Lasso, and Elastic Net Regression. We list the top 10 features with the highest (lowest) coefficients in each model to which features are considered important by models (no enough space to show graphs). We find that our hypothesis in section 4 is almost right: Above Ground Living Area, Size of Garage in Cars Capacity, and Original Construction Year Built are the three main features that affect house price.

5.3 Random Forest Regression:

After analyzing how single parameter changes will affect the model, we find that the more trees (number of estimators) is added into the forest, the lower the CV RMSE we will get. We also notice that the number of features (tree depth) selected into a tree doesn't affect too much the CV RMSE. This is not what we expected. We should consider multiple parameters at one time in later study.

5.4 Support Vector Regression:

Free parameters in epsilon-SVR are C and eps. To figure out the optimal value, we employ Grid Search to determine the best pair of these 2 parameters on our dataset, according to the resulted CV RMSE. The final optima are 2.5 and 0.031, respectively.

5.4 XGBoost:

XGBoost has more and complex parameters, and influence the model in different perspective. Max_depth(3, opt) and min_child_weight(7.) parameters has very great influence on the final result and should be optimized through Grid Search first. Regularization parameter gamma(0.) determines the complexity of the tree and should be optimized then. After fixing these parameters, subsample(1) and colsample_bytree(0.55) needs furthur optimization. XGBoost model has some insensitive parameters which can be determined manually instead of time-consuming fine-grained Grid Search.

5.5 Stacking (Ensemble Method):

Firstly, we set the second level model as sklearn Linear Regression. Then we try different combinations of single models in the first level of stacking. We find that the more models combined in the first level of the stacking, the better the performance of ensembled model is (because of the limited space, we don't show tables of CV RMSE for stacking here).

The best first level model we have is:

Ridge (normal equation method with lambda = 0.6) +
Lasso (sklearn with lambda = 0.005) +
ElaNet (sklearn with alpha=3.3 + 0.005 and 1_ratio=0.005/3.305) +
SVR (our) +
XGB (XGBoost) +
Random Forest Reg (sklearn with n_estimators=100 and max_depth=50)

Then we fixed this first level combination and try different single model for the second level. We find that the "simpler" the model in the second level of the stacking is, the better the performance of ensembled model is. No model works better than Linear Regression (sklearn).

By the comparing the RMSE based on training set cross validation and the RMSE score on Kaggle leader board, we conclude that stacking do perform better than single model.

Model	Score (CV RMSE)
Best Stacking Ensemble	0.111951
XGBoost Reg (xgboost)	0.118954

Kaggle Result:

There are totally 4645 teams' scores listed on Kaggle leaderboard. Our best model is ranked at about top 17%.

#	Team	Score	Entries
1	Alex plus BG que Lucas	0.00000	1
2	Lucas le bg	0.03176	4
3	Zheng Pan	0.08021	1
…			
803	**Our Best Stacking**	0.11864	15
1389	**Our Best Single Model**	0.12543	5

In conclusion, we have studied 7 regression methods and one useful model ensemble method. We found Above Ground Living Area, Size of Garage in Cars Capacity, and Original Construction Year are the most important features in several models. Our best single model is XGBoost and we reach our best Kaggle score by stacking. In future study, we would like to go deeper into the Machine Learning algorithms from open source libraries, implement our own random forest algorithm, and compare stacking with other ensemble method, such as Begging and Boosting.

6. Timeline and Roles

Task	**Deadline**	**Lead**
EDA for 20 features	11/15	Everyone
Feature Engineering for 20 features	11/29	Everyone
Combine and finalize datasets	11/29	Botao
Build Linear, Ridge, Lasso, Elastic Net Regression	12/06	Wenbin, Ziran
Build XGBoost and SVR	12/06	Qili
Build Random Forest regression	12/06	Ziran
Parameter selection for all models, Model ensemble by stacking	12/10	Qili, Ziran
Write report and poster	12/10	Everyone

7. References

1) Marcelino, Pedro. *Comprehensive Data Exploration with Python | Kaggle*, Feb. 2017, www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-python.

2) Serigne. *Stacked Regressions : Top 4% on LeaderBoard | Kaggle*, July 2017, www.kaggle.com/serigne/stacked-regressions-top-4-on-leaderboard

3) Drucker, H., Burges, C.J., Kaufman, L., Smola, A.J. and Vapnik, V., 1997. Support vector regression machines. In *Advances in neural information processing systems* (pp. 155-161).

4) Chen, T. and Guestrin, C., 2016, August. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794). ACM.

5) Wolpert, D.H., 1992. Stacked generalization. *Neural networks*, *5*(2), pp.241-259.