Predicting House Prices in Ames, Iowa

Shasha Li
University of California Riverside
900 University Ave.
Riverside, CA 92521, USA
sli057@ucr.edu

Tu Dang Nguyen
University of California Riverside
900 University Ave.
Riverside, CA 92521, USA
tnguy208@ucr.edu

1. INTRODUCTION

Accurate prediction of housing sales price is important in the operation of the housing market. Home sellers and buyers wish to know a fair value for their house in particular at the time of the sales transaction. A precise estimate of the sales price of a house is of real importance to investors who face choices among housing securities and other investment opportunities. Financial institutions try to obtain an accurate estimate of the market value to manage the risk better and consequently reduce the cost related to financing homeownership. In addition, many cities and counties base property taxes on the market value of a house, which must be updated periodically. Inaccurate appraisal of house values may result in substantial property tax adjustments. However, the accurate prediction of the house price is difficult because residential housing is a composite good which is typically sold as a package of various factors, such as location, environment, structural attributes, etc. It is not obvious how to select relevant factors among others and how to account for these factors in predicting the selling price of a house [16].

Our team decide to work on the price prediction of houses in Ames, Iowa, the information of which can be found on Kaggle [11]. We have Ames Housing dataset with 79 explanatory variables describing almost every aspect of residential homes in Ames, Iowa. Root-Mean-Squared-Error (RMSE) between the the logarithm of the predicted value and the logarithm of the observed sales price is used to estimate our model.

There are many potential methods for this problem such as regularized linear regression(Ridge and Lasso) [17], random forest regressor [6] and gradient boosting with the popular XGBoost library [19]. We implement all these four methods to predict house prices in our project. Beside, feature engineering really provides us rich opportunities to improve the prediction performance. Last but not least, model ensembling is a very powerful technique in our project.

2. RELATED WORK

CS 235 Data Mining 2016, California, USA

DOI: 10.475/123_4

2.1 A General Framework for Accurate and Fast Regression by Data Summarization in Random Decision Trees [5]

Summary: To solve the difficult and sometimes frustrating process to choose among many parametric and nonparametric regression methods, then either determine the component functions or select the right options and parameter values for a chosen methodology, Fan *et al.* propose the use of random decision trees (RDT) as a general framework for regression. Their goal is high accuracy, simplicity and efficiency. The procedure to generate random trees for regressions is exactly the same as for classification and posterior probability estimation problems. It summarizes the training data into k =10 to 30 trees randomly. When constructing each tree, the algorithm picks a feature randomly when expanding a node without any purity check (such as information gain). When picking a categorical feature, it ensures that the feature-value combination has not been chosen previously on the path from that node to the root node. Continuous features on the other hand can always be chosen and are set with a random threshold. We stop expanding a node when the number of instances that reach that node are below a threshold. For regression problems, each tree outputs the average value of the dependent variable of all examples in the classifying leaf node, just as CART. Then the outputs from multiple random trees are averaged as the final regression output.

Pros: RDT is remarkably simple and fully automatic, but highly accurate and efficient regression technique that is applicable to all known areas of regression applications. Compared with many state-of-the-art regression algorithms (e.g., CART [3] and GUIDE [15]) at that time (2006), the proposed method requires little next to none knowledge from the user and fully automates the model construction process, but returns either significantly better or similar results as more sophisticated approaches in a large number of problems. It works particularly well for highly non-linear high dimensional problems. RDT improves the randomization of random forests by not only selecting predictors randomly, but by also selecting a random point, sometimes deterministically chosen from a small set of random candidate split points [1].

Cons: First, as shown in the paper, RDT outperforms CART and GUIDE only when the number of decision trees is 30. This will require a lot of time and resources to build and store the random trees. Second, as a decision-tree-based

algorithm, RDT also suffers from the same drawbacks of its base - a decision tree is merely an efficient representation for a set of hyper-rectangles that partition the decision space. For ordinary decision trees, each hyper-rectangle is aligned with at least one of the axes of the chosen coordinate system, resulting in axis parallel decision boundaries. This results in very characteristic piecewise constant stair shapes, even when the number of trees in the ensemble is large, as can be observed visually in low dimensional graphical examples. As a consequence, a much greater number of trees is needed to accurately approximate an oblique decision boundary than a decision boundary that is axis aligned with standard tree ensembles [1]. Third, RDT also uses ensemble methods, which combine the predictions of multiple base learners to construct more accurate aggregate predictions. Established supervised learning algorithms inject randomness into the construction of the individual base learners in an effort to promote diversity within the resulting ensembles. An undesirable side effect of this approach is that it generally also reduces the accuracy of the base learners [1].

2.2 Better Models for Prediction of Bond Prices [7]

Summary: The authors of this paper aim to use the techniques and algorithms of machine learning and a set of data describing trade histories, intermediate calculations, and historical prices in order to more accurately predict up-to-date bond prices using data that would be viable to obtain at a particular moment in time. First, they evaluate the performance of various supervised learning algorithms for regression followed by ensemble methods, with feature and model selection considerations being treated in detail. They further evaluate all methods on both accuracy and speed. Finally, they propose a hybrid time-series aided machine learning method that could be applied to such datasets in future work.

Here is the list of methods which are evaluated in the paper:

- Feature Selection: Principle Component Analysis (PCA)
- Supervised Learning Methods: Generalized Linear Models (GLMs), Regression Trees (RT), Principal Component Regression (PCR), and Support Vector Regression (SVR)
- Ensemble Methods: Bagging, LS-Boosting, and Random Forests
- Hybrid Time-Series Methods
- Neural Networks

The key idea of the paper is to implement a time-series forecasting method that would allow authors to create a set of time-series predictions that could be used to either augment our feature set or even replace all of the historical features in a concise fashion. To do this, they have used the Auto-Regressive Moving Average (ARMA) Models, slightly simple method to predict future values of a variable based on its historical behavior. Their algorithm is as follows:

1. Estimate an ARMA(1,1) model for 10 samples of each bond type in the training set using a variable defined as the difference in the trade and curve prices at each time point
2. Average ARMA parameters to create an average TS model for the difference in trade and curve price for that bond type
3. Forecast one period forward from the historical data, which gives a prediction of the difference between the trade and curve price for each bond type for the prediction period
4. Use this forecast variable as a new feature in GLM models

Pros: Via extensive experiments, they authors have made several definitive conclusions regarding the relative performance of the tested models in predicting bond price: (i) GLM models perform well with low computational cost (order of seconds), (ii) Feature set augmentation with time-series models improves results, (iii) Ensemble methods do not substantially improve results, and require much more computational investment, (iv) Neural networks give very accurate results without overfitting in reasonable amounts of time (order of hours), (v) NNs and GLMs give best results in terms of combined speed and accuracy.

Cons: The authors have investigated and used one time-series model (ARMA) for feature generation. If they used other models, the performance might be improved. In this paper, NN has only two layers (one hidden layer and one output layer). Since NN shows very good performance, it could yield better bond price predictions with multilayer networks.

2.3 Regression Shrinkage and Selection via the Lasso [20]

Summary: This paper proposes a linear regression method called Lasso (least absolute shrinkage and selection operator). Generally speaking, Lasso minimizes the residual sum of squares subject to the sum of absolute value of the coefficients being less than a constant. It?s hard to understand some details in this paper because it uses lots of mathematical knowledge, but we can still get a big picture of this method.

Pros: Lasso can produce interpretable models like subset selection. It tends to produce some coefficients that are exactly 0 due to the nature of this constraint. Lasso also exhibits the stability of ridge regression. it sacrifices a little bias to reduce the variance of the predicted values and hence is more stable.

Cons: Lasso is not a very satisfactory variable secretion method when the number of predictors is much bigger than the number of observations. 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 [9].

Extensions: The *L1 penalty* is crucial for the success of the lasso. However, there are some potential extensions to this constraint, for example, we can use adaptive weighs for penalizing different coefficients [22].

2.4 Prediction of Salary in UK [14]

Summary: This paper provides detailed steps to predict the salary of the job posing based on the advertisement alone. Firstly, the paper explores the characteristics of the dataset, which play useful roles both in data preprocessing and feature selection. Secondly, it does some data

prepossessing and feature exploration works. Thirdly, several methods are used to model data, such as Support Vector Regression, Nearest Centroid, Linear Regressor, Logistic Regressor, K-neighbors Regressor and Random Forest Regression. It turns out that Random Forest Regressor gives the best result.

Pros: This paper lists detailed data exploration steps and thus gives us insight of some principles for data prepossessing and feature selection. Detailed analysis is provided on how to design and select models. It starts from single feature and linear regression, and then based on the results of single feature, it combines features and uses various regression methods.

Cons: The number of features explored in this paper is small. This limits the prediction accuracy.

Extensions: With some text-mining methods, we can explore more useful features to do prediction work. Many neural network methods are convinced to perform well on several production works. We can try to use neural network models on this problem.

2.5 Random Forests [2]

Summary: Random forests are a combination of tree predictors such that each tree depends on the value of a random vector sampled independently and with the same distribution for all trees in the forest. Two different forms of random features are provided in this paper: 1) The first one uses random section form the original inputs; 2) The second creates new attributes that are a linear combinations of existing attributes, which can reduce the correlation between individual classifiers when there are only a few attributes available.

The accuracy of random forest depends on the strength of the individual trees in the forest and the correlation between them. The paper also analyzes internal estimates of variable importance to try to understanding the mechanism of the random forest "black box".

Pros: Its accuracy is as good as Adaboost and sometimes better. Random forests are robust to errors and outliers. The paper uses Strong Law of Large Numbers and shows that the generalization error for a forest converges as long as the number of trees in the forest is large. Thus, overfitting is not a problem. Random forests consider many fewer attributes for each split, so they are efficient on large datasets. Besides, it?s easily parallelized. Random forest is insensitive to the number of features selected to split each node.

Cons: Results of learning are incomprehensible. Compared to a single decision tree, or to a set of rules, they don't give you a lot of insight [18].

Extensions: The only types of randomness used in this paper is bagging and random feature. It may well be that other types injected randomness give better results, such as random Boolean combinations of features.

3. REGRESSION METHODS

3.1 Data Cleaning and Preparation

3.1.1 Handling missing data with rpart

kNN Imputation: DMwR::knn Imputation uses k-Nearest Neighbors approach to impute missing values. What kNN imputation does in simpler terms is as follows: For every observation to be imputed, it identifies k closest observations based on the euclidean distance and computes the weighted average (weighted based on distance) of these k observations.

rpart: The limitation with DMwR::knn Imputation is that it sometimes may not be appropriate to use when the missing value comes from a factor variable. Both rpart and mice has flexibility to handle that scenario. The advantage with rpart is that you just need only one of the variables to be non NA in the predictor fields.

How to use rpart

1. Import rpart library using the command *library(rpart)*
2. Identify the attributes/columns used to predict the missing attribute. For example, we want to predict the missing value of the attribute "swimming pool quality". We will use the following attributes as predictors:
   col.pred <- c("YearBuilt", "YearRemodAdd", "PoolQC", "PoolArea", "WoodDeckSF", "OpenPorchSF", "EnclosedPorch", "X3SsnPorch", "ScreenPorch", "ExterQual", "ExterCond", "YrSold", "SaleType", "SaleCondition")
3. Predict the missing value using the following two commands:
   qlty.rpart <- rpart(as.factor(PoolQC) ., data = df[!is.na (df$PoolQC), col.pred], method = "class", na.action = na.omit)
   df$PoolQC[is.na(df$PoolQC)] <- predict(qlty.rpart, df[is.na (df$PoolQC), col.pred], type = "class")

Note: Since most of the missing values are not important attributes, rpart imputation does not contribute much improvement in predicting the house prices. Thus, we do not use this technique in our prediction models. However, we think that this technique may be very useful to impute the missing values of important attributes.

3.1.2 Feature selection with Boruta

Description: Boruta is an all relevant feature selection wrapper algorithm, capable of working with any classification method that output variable importance measure (VIM); by default, Boruta uses Random Forest. The method performs a top-down search for relevant features by comparing original attributes' importance with importance achievable at random, estimated using their permuted copies, and progressively eliminating irrelevant features to stabilize that test.

Details: Boruta iteratively compares importance of attributes with importance of shadow attributes, created by shuffling original ones. Attributes that have significantly worst importance than shadow ones are being consecutively dropped. On the other hand, attributes that are significantly better than shadows are admitted to be Confirmed. Shadows are re-created in each iteration. Algorithm stops when only Confirmed attributes are left, or when it reaches maxRuns importance source runs. If the second scenario occurs, some attributes may be left without a decision. They are claimed Tentative. You may try to extend maxRuns or lower pValue to clarify them, but in some cases their importance do fluctuate too much for Boruta to converge. Instead, you can use TentativeRoughFix function, which will perform other, weaker test to make a final decision, or simply treat them as undecided in further analysis.

How to use Boruta

1. Import Boruta library using the command *library(Boruta)*

Table 1: Feature selection result with Boruta

Attribute	normHits	Decision
GrLivArea	1.00	Confirmed
OverallQual	1.00	Confirmed
TotalBsmtSF	1.00	Confirmed
X2ndFlrSF	1.00	Confirmed
...	...	...
MasVnrType	0.66	Tentative
BsmtExposure	0.60	Tentative
LotShape	0.57	Tentative
...	...	...
BsmtFinSF2	0.06	Rejected
ExterCond	0.03	Rejected
...	...	...

2. Retrieve data for analysis:
sample.df <- read.csv(file.path("..", "train.csv"), stringsAsFactors = FALSE)

3. Extract candidate feature names:
candidate.features <- setdiff(names(sample.df),c("Id", "SalePrice"))

4. Pull out the response variable:
response <- sample.df$SalePrice

5. Remove identifier and response variables:
sample.df <- sample.df[candidate.features]

6. Run Boruta analysis:
set.seed(13) and *bor.results <- Boruta(sample.df, response, maxRuns=101, doTrace=0)*

7. Some parts of the result are shown in the table 1. Then, we may use the confirmed (and tentative) attributes to predict the house prices.

3.1.3 Feature engineering methods

Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work [21].

We use four steps in the feature engineering in our project.

1. The first variable MSSubClass is actually a categorical variable that is recorded as numeric, so we change this variable to type string.

2. We create feature 1stFlr_2ndFlr_Sf based on the original features 1stFlrSF and 2ndFlrSF and feature All_Liv_SF based on the 1stFlrSF, 2ndFlrSF, LowQualFinSF and GrLivArea.

   Figure 1 shows that the top ten largest coefficients in the Lasso Model and we can see these two features we created play leading roles in the prediction. The features we engineered end up pretty important. This step actually improves our rank from 670 to 248.

3. We calculate the skewness of every numeric feature and do log-transform for those whose skewness ≥ 0.75. Figure 2 shows how the data looks like before and after transformation.

4. Since data mining models cannot handle categorical variables, we must 0/1 encode them as a different column for each unique category for each variable.

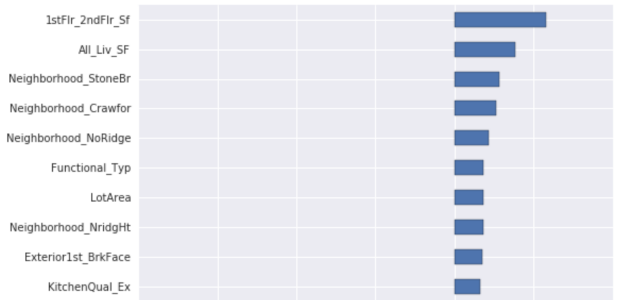

Figure 1: Top ten largest coefficients in the Lasso Model

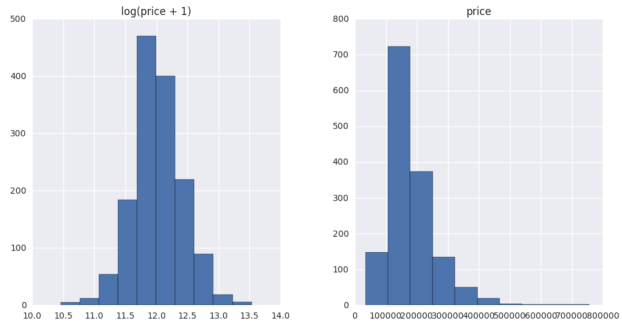

Figure 2: How does skew data and transformed skew data look like

3.2 Regression Models

3.2.1 Lasso

We has introduced Lasso in section 2.3. The only parameter in Lasso model is the coefficient α of the penalty term. We test 50 different values from 10^{-5} to 1 and finally we use α =1.26485521686e-05 in our project.

3.2.2 Ridge

Ridge[8] is another regularized linear regression model. Lasso minimizes the residual sum of squares subject to the sum of absolute value of the coefficients being less than a constant. Ridge, however, is constrained to the sum of the square value of the coefficients instead of absolute value. Ridge also has only one parameter, α in the penalty term. We test 50 different values from 10^{-3} to 10^{2} and α=19.3069772888 gets the best result. Figure 3 shows how the cross-validation scores change with α increases.

3.2.3 Random Forest

We has introduced Random Forest in section 2.5. In our project, we use random forest regressor to predict the house price. However, it works not very well. The reason may be that we get too many parameters for RandomForestRegressor and we just haven't made wise choices.

There are 15 parameters in the function RandomForestRegressor (imported from sklearn package), for example, the number of trees in the forest, the maximum depth of the tree, the minimum number of samples required to be at a leaf node, and the number of features to consider when looking for the best split. In our experiment, we concentrate on max_features (the number of features to consider when

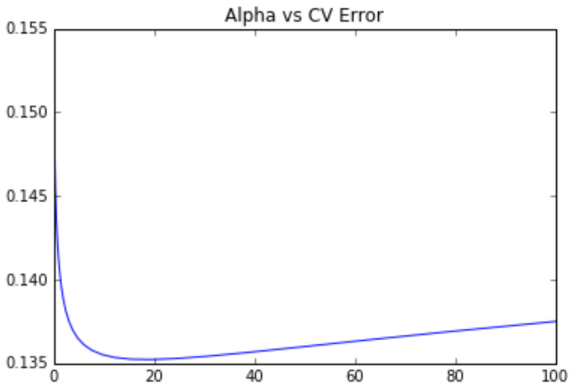

Figure 3: α vs cross-validation score

looking for the best split). We test different percentages of features to be considered and value 0.3 gets the best scores, so we use max_features=0.3 in our random forest regression model. Figure 4 shows how the cross-validation scores change with max_features increases.

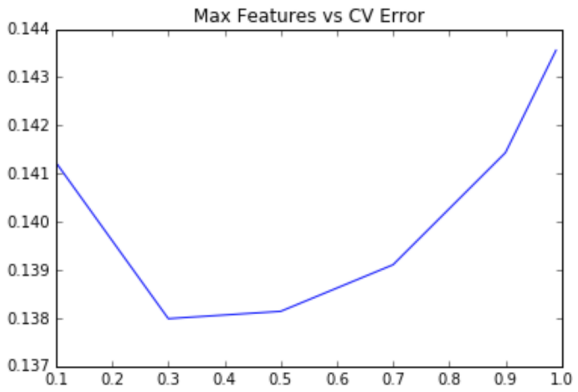

Figure 4: max_features vs cross-validation score

3.2.4 XGBoost

Description: The XGBoost library implements the gradient boosting decision tree algorithm. This algorithm goes by lots of different names such as gradient boosting, multiple additive regression trees, stochastic gradient boosting or gradient boosting machines. Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made. A popular example is the AdaBoost algorithm that weights data points that are hard to predict. Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models. This approach supports both regression and classification predictive modeling problems.

How to use XGBoost:

1. Import XGBoost library using the command *library(xgboost)*
2. Load data:
   Training <- read.csv("./input/train.csv")
   Test <- read.csv("./input/test.csv")
   n <- ncol(Training)
3. Set parameters:
   param <-list(objective = "reg:linear", eval_metric = "rmse", booster = "gbtree", max_depth = 10, eta = 0.03, gamma = 0.1, subsample = 0.734, colsample_bytree = 0.4)
   Booster Parameters:
   - *eta*: The default value is set to 0.3. You need to specify step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features. and eta actually shrinks the feature weights to make the boosting process more conservative. The range is 0 to 1. Low eta value means model is more robust to overfitting.
   - *gamma*: The default value is set to 0. You need to specify minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm will be. The range is 0 to ∞. Larger the gamma more conservative the algorithm is.
   - *max_depth*: The default value is set to 6. You need to specify the maximum depth of a tree. The range is 1 to ∞.
   - *min_child_weight*: The default value is set to 1. You need to specify the minimum sum of instance weight(hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning. In linear regression mode, this simply corresponds to minimum number of instances needed to be in each node. The larger, the more conservative the algorithm will be. The range is 0 to ∞.
   - *max_delta_step*: The default value is set to 0. Maximum delta step we allow each tree's weight estimation to be. If the value is set to 0, it means there is no constraint. If it is set to a positive value, it can help making the update step more conservative. Usually this parameter is not needed, but it might help in logistic regression when class is extremely imbalanced. Set it to value of 1-10 might help control the update.The range is 0 to ∞.
   - *subsample*: The default value is set to 1. You need to specify the subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collected half of the data instances to grow trees and this will prevent overfitting. The range is 0 to 1.
   - *colsample_bytree*: The default value is set to 1. You need to specify the subsample ratio of columns when constructing each tree. The range is 0 to 1.
4. Prepare train dataset:
   re_train<- as.matrix(Training,rownames.force=NA)
   re_train<- as(re_train,"sparseMatrix")
   retrain_Data<- xgb.DMatrix(data = re_train[,1:n-1], label = re_train[,"SalePrice"])

5. Train the model:
   bstSparse_retrain<- xgb.train(params = param, data = retrain_Data, nrounds = 600, verbose = FALSE, nthread = 6)

6. Prepare test dataset:
   Test_Matrix<-as.matrix(Test, rownames.force = FALSE)
   Test_Matrix<-as(Test_Matrix, "sparseMatrix")
   Test_Matrix<-xgb.DMatrix(data = Test_Matrix[,1:n-1])

7. Run the prediction:
   XGBoostPrediction <- predict(bstSparse_retrain, newdata = Test_Matrix)

8. Print the result to the output file:
   XGBoostSubmission <- cbind(Id= TestID, SalePrice = XGBoostPrediction)
   colnames(XGBoostSubmission) <- c("Id", "SalePrice")
   write.csv(XGBoostSubmission, file = "submission.csv", row.names=FALSE)

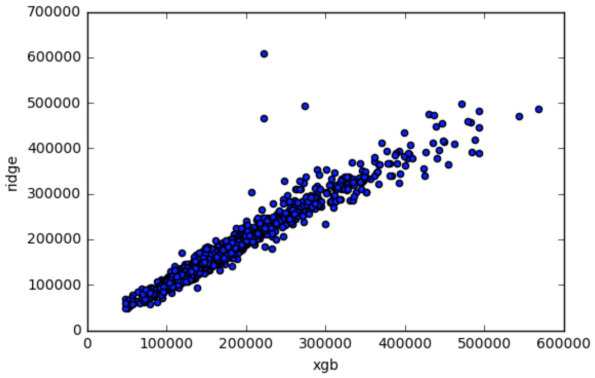

Figure 5: XGBoost vs Ridge

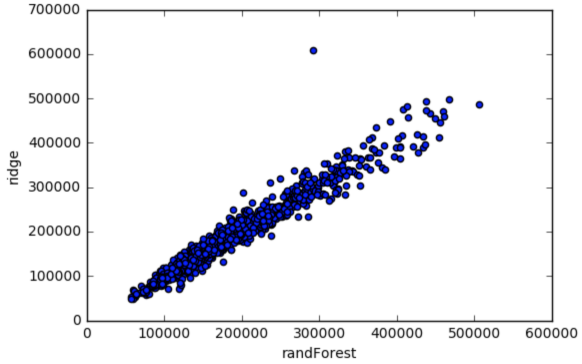

Figure 6: Random Forest vs Ridge

3.2.5 Ensemble

Model ensembling [4] is a very powerful technique to increase accuracy on a variety of data mining tasks.[12] As Vitaly Kuznetsov said, "This is how you win ML competitions: you take other peoples' work and ensemble them together".

There are two typical ways to create ensembles. The most basic and convenient way is to ensemble the test results. We only need the predictions on the test set for these methods; no need to retrain a model. In addition, uncorrelated output results usually do better when ensembled than correlated outputs of models. We can also use stacking and blending to create ensembling.

Our project use the first emsembling method, which improves the performance obviously. Weighted average ensemblings are used for Ridge & Random Forest, and Ridge & XGBoost. The results show that the emsembling between Ridge & XGBoost is better. We can see from Figure 5 and 6 that Ridge & XGBoost seem to be more correlated.With $0.8 * Ridge + 0.2 * XGBoost$, we rank from 248 to 206 in the Kaggle competition.

4. PERFORMANCE EVALUATION

Dataset: Ames Housing dataset [11]

Evaluation/Score: Root-Mean-Squared-Error(RMSE) between the the logarithm of the predicted value and the logarithm of the observed sales price

Others:10-fold cross-validtion used in training for each model

4.1 Data Visualization

Figure 7 [10] shows the correlations between numeric features, which can give us a whole picture of the numeric data.

We can also do visualization for categorical features, for example, Figure 8 shows the distribution of saleType and saleCondition features

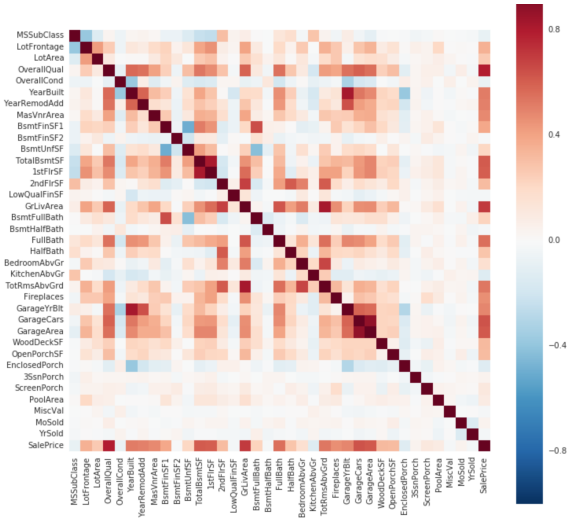

Figure 7: The correlations between numeric features

4.2 Feature Engineering

We use different kinds of methods to handle features.

1. method 1: only transform categorical values to dummy values
2. method 2: feature engineering (add two new features, do log-transform for some numeric values)
3. method 3: feature selection using XGBoost

Table 2 shows the scores to different methods and we can see that feature engineering definitely improves the performance.

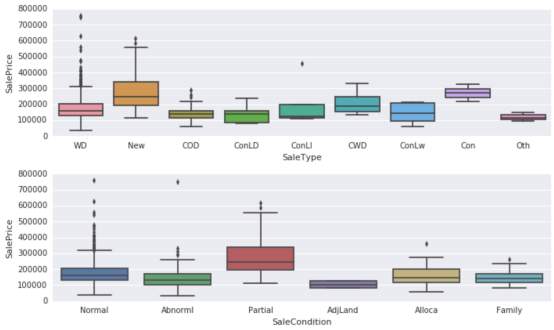

Figure 8: Distribution of two categorical features: saleType and saleCondition

Table 2: Compare feature handling methods with Ridge method

methods	scores
method 1	0.12433
method 2	0.11877
method 3	0.11935

4.3 Regression Models

We test different kinds of models to do prediction. The results are showed in Table 3. For both Random Forest and XGBoost, there are too many parameters to define, so the results using these complicated models are not very satisfying.

Table 3: Compare different regression models with feature engineering

regression models	scores
Ridge	0.11877
Lasso	0.12059
Random Forest	0.14261
XGBoost(n_estimators=350, max_depth=2)	0.13381
XGBoost(n_estimators=450, max_depth=4)	0.13369

XGBoost and Random Forest with Boruta First we use Boruta to select important attributes. Then, we use XGBoost and Random Forest to predict the house prices based on the chosen attributes. Table 4 shows important parameters used in XGBoost prediction. These parameters are tuned manually.

As shown in figure 9, XGBoost's prediction is better than that of the simple ensemble method - a weighted average of Random Forest and XGBoost.

4.4 Ensemble

Table 5 shows the results of different ensembling methods. We can see that the emsembling between Ridge and XGBoost is better than that between Ridge and Random Forest, which is consistent with what we mentioned at section 3.2.5. Besides, we need to wisely choose weights to gain better performance, for example, $0.8*Ridge+0.2*XGBoost$ is much better than $0.5*Ridge+0.5*XGBoost$ which don't

Table 4: Parameters used in XGBoost prediction

Parameters	Value
max_depth	10
eta	0.03
gamma	0.1
subsample	0.734
colsample_bytree	0.4

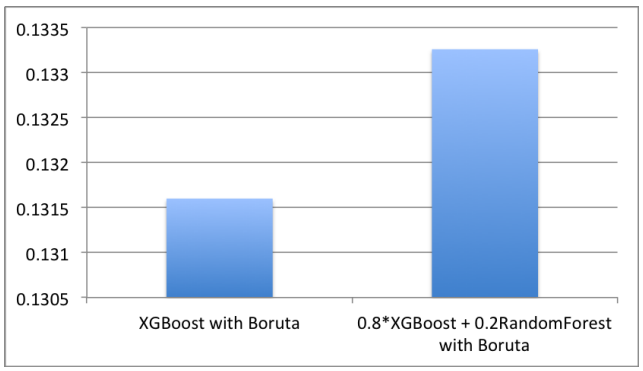

Figure 9: RMSE of XGBoost and Random Forest with Boruta.

even improve the prediction

Table 5: Compare emsembling methods

emsembling	scores
Ridge	0.11877
0.7*Ridge+0.3RandomForest	0.12059
0.7*Ridge+0.3XGBoost	0.11835
0.5*Ridge+0.5XGBoost	0.12036
0.8*Ridge+0.2XGBoost	0.11802
0.9*Ridge+0.1XGBoost	0.11802

4.5 Remove Outliers

Inspired by Meichengshih [13], we first use Ridge model to do regression and find several outliers in training set. We then remove these outliers and do regression again. The experimental results show that we can actually improve the score from 0.11877 to 0.11871. Although it's a little improvement, the idea is novel and creative.

5. CONCLUSION

From the original score 0.1316 (rank 839), we improve our method step by step and eventually achieve 0.11802 (rank 206, top 10.2%). We find three valuable things during this period.

1. Feature engineering is extremely important. By simply creating two features based on original features, we improve our rank from 670 to 248.

2. Ensembling is pretty powerful. We can improve the performance significantly by simply combine different existing models.

3. We can do some simple regressions first to detect outliers and then build regression models without outliers.

Table 6: Remove outliers

Remove outliers or not	scores
no removing	0.11877
remove outliers	0.11871

To improve our current method, we have two directions in mind.

1. Feature engineering: We should look closer to these features and maybe consult some domain experts to create new features. We can also do more feature selection works, for example, Sebastian Bayer[13] in Kaggle Forum said what helped his score was removing highly unbalanced categorical variables (he remove a dummy variable if the share of 0's or 1's is below 3%).

2. Ensemble: We should try to stack and blend existing models, which seems to have promising results.

3. Parameters selection: It is not easy to find the optimal values for parameters of complex algorithms such as XGBoost and Random Forest since they use many parameters. We plan to apply Minimum Description Length (MDL) concepts on this issue.

6. REFERENCES

[1] R. Blaser and P. Fryzlewicz. Random rotation ensembles. *The Journal of Machine Learning Research*, 17(1):126–151, January 2016.

[2] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, January 2001.

[3] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, New York, 1984.

[4] T. G. Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[5] W. Fan and J. McClosky. A general framework for accurate and fast regression by data summarization in random decision trees. In *SIGKDD*, pages 136–146. ACM, August 2006.

[6] B. Fitz-Gerald. Randomforestregressor. In *Kaggle Competition in House Prices Prediction*, pages 1–1. Kaggle, September 2016.

[7] S. Ganguli and J. Dunnmon. *Better Models for Prediction of Bond Prices*. Standford, Standford, USA, 2014.

[8] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[9] Z. Hui and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society*, Series B(67.2):301–320, January 2005.

[10] Kaggle. Detailed data exploration in python. In *Kaggle Competition in House Prices Prediction*, pages 1–1. Kaggle, September 2016.

[11] Kaggle. House prices: Advanced regression techniques. In *Kaggle Competition in House Prices Prediction*, pages 1–1. Kaggle, September 2016.

[12] Kaggle. Kaggle ensembling guide. In *Kaggle Competition in House Prices Prediction*, pages 1–1. Kaggle, September 2016.

[13] Kaggle. Sharing my approach to motivate more discussions. In *Kaggle Competition in House Prices Prediction*, pages 1–1. Kaggle, September 2016.

[14] L. Li, X. Liu, and Y. Zhou. *Prediction of Salary in UK*. UC San Diego, California, USA, 2015.

[15] W.-Y. Loh. Regression trees with unbiased variable selection and interaction detection. *Statistica Sinica*, 12(2):361–386, April 2002.

[16] B. Okmyung. A prediction comparison of housing sales prices by parametric versus semi-parametric regressions. *Journal of Housing Economics*, 13(1):68–84, January 2004.

[17] A. Papiu. Regularized linear models. In *Kaggle Competition in House Prices Prediction*, pages 1–1. Kaggle, September 2016.

[18] Quora. *When is a random forest a poor choice relative to other algorithms?* Quora Website, https://www.quora.com/When-is-a-random-forest-a-poor-choice-relative-to-other-algorithms, 2016.

[19] J. Thompson. Ensemble modeling: Stack model example. In *Kaggle Competition in House Prices Prediction*, pages 1–1. Kaggle, September 2016.

[20] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society*, Series B(1):267–288, January 1996.

[21] wiki. https://en.wikipedia.org/wiki/Feature_engineering. WIKI.

[22] H. Zou. The adaptive lasso and its oracle properties. *Journal of the American statistical association*, 101(476):1418–1429, January 2006.