Prediction of Ames House Prices

Emily Goren, Andrew Sage, Haozhe Zhang

May 5, 2017

Introduction

The Kaggle House Price competition involves predicting prices of houses sold in Ames, Iowa between 2006 and 2010. Using the prices of houses in a training set that consists of half of the houses sold, and information provided by more than 80 explanatory variables, we seek to predict the prices of the other half of the houses sold. This test set is further divided, with half of its observations contributing to a public score posted on Kaggle's leaderboard, and the other half contributing to a private score, which we only learned after our final submission.

We use statistical machine learning techniques, discussed in Iowa State University's stat 602 course, to make these predictions. An important consideration involves objectively assessing the performance of our predictions prior to seeing the public Kaggle scores, as such scores would typically not be available until after final predictions are made.

In this report, we describe our attempts to create useful features, implement machine learning algorithms, and combine resulting predictions in an effective way. Furthermore, we detail our efforts to gauge how well our predictions would perform on test data, through cross-validation. Some of our strategies proved more effective than others, and in the final section, we reflect on the lessons we learned through our participation in this competition.

Data Cleaning and Feature Engineering

Construction of Feature Matrix

The dataset consists of 1,460 training cases and 1,459 test cases. The response variable is sale price and there are 79 covariates. Thirty-six of which are numeric and the rest are categorical. A quick exploratory analysis reveals that although there are a number of "NA's" in the data, most of these correspond to the absence of a characteristic, rather than an a value that is truly lost or unknown. For example, 158 of the 159 houses with garage type "NA" have garage area 0, suggesting that these houses do not have garages. A category called "None" was created for all categorical variables pertaining to the garage, and numerical variables pertaining to the garage, such as number of cars were set to 0. The exception is year built, which was set to be the same as the year the house was built. We then created an indicator variable for missingness of garage. The same strategy was used for variables pertaining to alley, fence, fireplace, pools, basements, and masonry vaneer.

We are left with only a handful of true missing cases (if any), for all but one of the explanatory variables. For categorical variables, we initially created a new level called "unknown," for these cases. For numerical predictors, we initially set these to zero and created binary variables to indicate whether the value was missing. The variable with a large number of missing values is lot frontage, which contains 486 missing values.

After accounting for missingness, we considered ways to make better use of some of the variables provided in the dataset by changing their type, or creating new features. Although MSSubClass is coded as a numeric variable, it is clear from the data description that it should be treated as categorical. The variables Condition 1 and Condition 2 each contain nine levels, indicating proximity to such conditions such as railroads or arterial or feeder streets. We replaced these two variables with nine indicator variables, one corresponding to each type of condition. Additionally, we created new features for the average room size, and indicators for whether the house contained a second floor, was remodeled, was sold the same year it was build, and was sold during the months of May, June or July.

Many of the categorical variables in the dataset can be associated with a natural ordering, (i.e., excellent, good, typical, fair, poor). It is advantageous to utilize this ordering, when using tree-based methods, since these methods are invariant to scale. A version of the data matrix, intended for tree-based approaches, was created in which ordinal variables were converted to numeric and ordered appropriately. Categorical variables, such as type of basement finish, for which there is no natural ordering were initially left as categorical. When performing linear regression these variables were treated as categorical and indicators for each level were used.

The steps taken thus far rely on rules derived from the data descriptions, and do not depend on the actual data. Whether these steps are taken on the entire training and test data together, or within the folds of a cross-validation procedure has no bearing on the resulting features. Using the training data to inform our creation of features and to perform imputation might result in improved predictive performance. A Kaggle kernel by Tanner Carbonati (2017), which was discussed by other groups in class, provides a number of ideas for construction of such features.

Numerical variables with missing values, most notably lot frontage, can be imputed by using the values of lot frontage for houses in the same neighborhood. Similarly, missing values for categorical variables can be filled using the most frequently occurring category for that variable, for similar cases. For example, there are three houses with pools, whose quality was not reported. Carbonati suggests assigning these pools the quality that occurs most frequently for pools with size similar to the ones with missing values.

In addition to using training data to impute missing values, Carbonati uses it to determine an appropriate ordering for categorical variables, such as type of basement finish. This can be done by replacing each level of the categorical variable with the median sale price for that category. This results in a dataset consisting entirely of numeric explanatory variables, which is helpful when using the XGBoost technique. We include both numerical variables constructed in this manner and indicator variables for each level of categorical variables in our dataset.

Finally, we considered whether to include four potential outliers that occur in the training data. These houses have much larger amounts of above ground living area than other houses. We are not convinced that excluding these houses is appropriate. Although they have an unusual value for an important covariate, it is clear that they are outliers in the sense of having an unusual price, conditional on all of the covariates. However, we have obtained better predictive performance when these values are excluded. Since we were allowed two Kaggle submissions, we included one where these values were included and another where they were not.

Use of the training data, including the response variable, in imputation and feature selection has the potential to create powerful features that are useful in prediction. However, using all of the training and test data to create a feature matrix before performing cross-validation leads to optimistic cross-validation error rates. This results from the information in the holdout set leaking into the data through the feature matrix it was used to create. In order to perform an honest cross validation, imputation and feature selection must be performed within each fold of a cross-validation. In the next section, we examine the impact of imputation performed on each cross-validation reduced training set compared to imputation performed on the entire dataset.

Imputation and Cross-Validation

To obtain a cross-validation error that fully or "honestly" captures the performance of a predictor, all steps involved in producing a predictor must be applied to each cross-validation training set, including preprocessing (i.e., centering and scaling) and imputation. However, the preprocessing and imputation functionality in the `caret` package for `R` is limited and cannot be applied to categorical variables. Consequently, we created custom models for elastic net regression, partial least squares, and random forest approaches (see the next section for model and tuning details) that accept the raw dataset and perform all of the imputation procedures described in the previous section (except for removing outlying observations) prior to training using the corresponding built-in model in `caret`. As part of the custom model, we changed the prediction functionality to incorporate the imputation and preprocessing (so that the prediction behavior for the custom models mirrors the built-in ones).

Our "honest" 10-fold cross-validation procedure using `caret` is as follows. Let $\boldsymbol{T}_1, \ldots, \boldsymbol{T}_{10}$ represent the "folds" of the full training dataset, $\boldsymbol{T}$. Our custom `caret` models then perform imputation followed by preprocessing on each of the reduced training sets $\boldsymbol{T} - \boldsymbol{T}_k$ to produce the new feature matrices $\boldsymbol{X}_k^*$ that were used for training ($k = 1, \ldots, 10$). We define the cross-validation error resulting from this procedure to be "honest."

In contrast, the "default" 10-fold cross-validation procedure performs imputation using *both* the training dataset, $\boldsymbol{T}$, and the test dataset, $\boldsymbol{T}^{\text{new}}$, prior to splitting the training dataset into "folds." Let $\boldsymbol{T}^*$ denote the rows of the resulting imputed dataset that correspond to the training cases. Then, we form the cross-validation "folds" to obtain the reduced training sets $\boldsymbol{T}^* - \boldsymbol{T}_k^*$ and preprocess each one to obtain the new feature matrices $\boldsymbol{X}_k^{**}$ used for training ($k = 1, \ldots, 10$).

Since cross-validation was also used to choose tuning parameters, use of the "honest" procedure resulted not only in a larger cross-validation error, but also a larger Kaggle public score. For elastic net regression, "honest" cross-validation (repeated 10 times) produced a RMSE of 0.17409 and a Kaggle public score of 0.17501. Using the "default" method, the values were 0.13384 and 0.13309, respectively. The "honest" method's higher error can be attributed to tuning parameter selection that is based on a feature matrix imputed using fewer samples, making some features less informative. We omit our results for partial least squares regression and random forest approaches because we are not certain our code was working correctly. Due to the coding challenges involved in implementing the custom models, for the remainder of this paper we will use the "default" cross-validation method.

Model Tuning

Elastic Net Regression

In prediction settings, many features may be unimportant and contribute to overfitting. For linear regression, using regularization approaches instead of ordinary least squares may help avoid overfitting. The two most prominent regularization methods are the lasso and ridge regression, which impose a scaled ℓ_1 or ℓ_2 norm penalty, respectively, on the estimated coefficient vector. The elastic net linearly combines these two penalties with weights α and $1-\alpha$, respectively. A tuning parameter, λ, controls the strength of the penalty.

To be able to capture interactions and nonlinear relationships, we included one-way interactions between all variables in the feature matrix described previously, where categorical variables were converted to dummy variables, as well as quadratic terms for all numerical variables. Using the resulting feature matrix, we selected the two tuning parameters for the Ames housing data using 10-fold cross-validation repeated 10 times via the `caret` and `glmnet` packages for `R`. We searched over a grid for $\alpha \in \{0, 0.1, 0.2, \ldots, 1\}$ and $\lambda \in \{0.001, 0.051, 0.101, \ldots, 0.951\}$. For fitting using log sale price as the response, the optimal values were $\alpha = 0.1$ and $\lambda = 0.051$ (resulting error rates are presented in Table 2). Preliminary versions of the design matrix that we considered resulted in $\alpha = 0$, producing ridge regression.

Extreme Gradient Boosting

E**x**treme **G**radient **Boost**ing (XGBoost) belongs to a family of boosting algorithms that convert what are often called weak learners into strong learners. A weak learner is one which is slightly better than random guessing. As a sequential boosting process, XGBoost usually has more tuning parameters or "hyperparameters" than other machine learning algorithms. Since the tree booster outperforms the linear booster in many cases, in this problem, we consider only tree-based XGBoost, which has 7 hyperparameters. Each hyperparameter plays a significant role in the model's predictive performance. Briefly, the hyperparameters are

1. *max_depth*: the depth of the tree. Larger the depth, more complex the model leading to a higher chance of overfitting. There is no standard value, but larger data sets require deep trees to learn the rules from data.
2. *nrounds*: the maximum number of iterations.
3. *min_child_weight*: the minimum sum of weights of all observations required in a child.
4. *gamma*: the minimum loss reduction required to make a split. A node is split only made when the resulting split gives a positive reduction in the loss function.
5. *subsample*: the proportion of samples (observations) supplied to a tree.
6. *colsample_bytree*: the proportion of features (variables) supplied to a tree.
7. *eta*: the learning rate. Lower values leads to slower computation with an increase in *nrounds*.

Fine-tuning XGBoost by exploring the 7-dimension space in an efficient way is a big challenge in terms of computation. There are a lot of methods and literature about searching the global minimizer, such as random search (Bergstra and Bengio 2012), Bayesian optimization (Snoek, Larochelle, and Adams 2012, Kuhn (2016)), and efficient global optimization of expensive black-box functions (Jones, Schonlau, and Welch 1998). To reduce the computational burden, we considered the tuning parameters based on a grid using *nrounds*

$\in \{2000, 4000, 8000\}$, $eta \in \{0.01, 0.005, 0.001\}$, $max\_depth \in \{2, 4, 6, 8, 10\}$, $colsample\_bytree \in \{0.8, 1\}$, $min\_child\_weight \in \{2, 3\}$, $subsample \in \{0.6, 0.8, 1\}$, and $gamma \in \{0, 0.01\}$.

Repeated 10-fold cross-validation was conducted to search over the above grid on the condo2017 server that allows paralleling 16 jobs. It took a total of 10 hours to finish running the code. The resulting cross-validation error was 0.11607 and the selected tuning parameters were $nrounds = 4000$, $max\_depth = 4$, $eta = 0.005$, $gamma = 0$, $colsample\_bytree = 0.6$, $min\_child\_weight = 2$ and $subsample = 0.5$.

Partial Least Squares

We used partial least squares to perform regression on a parameter space of lower dimension that that of the original data matrix, taking advantage of house price information in the dimension reduction process. We used 10-fold cross validation, repeated 10 times to determine the optimal number of components. Using the data matrix including all 1460 training cases, the optimal number of components was 9, and this resulted in a cross-validation error rate of 0.12482.

Random Forest

We attempted to use random forest methodology as a nonlinear method for predicting house prices. Cross-validation was used to set the number of predictor variables considered for each split, as a terminal nodesize below which no further splits are made. Initial results for this method were not promising, typically yielding cross-validation scores at least 0.015 higher than other methods.

We attempted to improve performance by using a 2-step process suggested by Breiman (2003) in situations where there are a large number of predictor variables. First a random forest is grown using all predictors, and variable importance scores are calculated. A subset of the most important predictors is then used to grow a second random forest, from which predictions are made. Cross-validation was used to determine the size of this subset, (denoted P), along with the proportion of these variables to be considered for each split (m), and terminal nodesize (n). Due to computational complexity, a single 10-fold cross validation was performed with forests consisting of 100 trees. This was done using a version of the feature matrix that included 95 predictor variables, with ordinal variables treated as numeric. The following table gives cross-validation error for different combinations of P and m, when n is held at its optimal (and also default) value of 5.

Table 1: Cross-validation errors (10-fold, not repeated) for random forest methodology with Breiman's 2-step process.

P \ m	0.3	0.4	0.5	0.6	0.7
55	0.1450352	0.1455936	0.1452127	0.1455201	0.1455318
65	0.1451470	0.1442016	0.1428470	0.1454558	0.1459102
75	0.1456480	0.1455370	0.1454964	0.1437217	0.1464847
85	0.1452051	0.1458660	0.1455621	0.1459192	0.1458570
95	0.1457250	0.1431714	0.1453318	0.1460281	0.1459981

There is some sign that using 65 of the original 95 predictors and then considering about half of these for each split might improve performance compared to a random forest that uses all 95. However, the gain is very slight, and given the small number of trees used in this cross-validation, it is not clear that this is really an improvement at all. Ultimately, we decided not to use random forest predictions, as this method proved inferior to other methods by itself, and did not lead to improvement when averaged with predictions from other techniques. Early attempts using conditional random forests, and individual regression trees also showed inferior predictive performance and were also abandoned.

Model Ensembling and Stacking

After performing the tuning process, we obtained the predictions from four methods: XGBoost, elastic net, partial least squares, and random forests (Table 2). Among the four methods, XGBoost had the smallest cross-validation error and the best Kaggle public score. Although each of these methods has been tuned to attain its minimal cross-validation error, we can still improve the prediction performance by stacking/ensembling them. This is possible because each model may capture a different aspect of the data, and different models perform best on different subsets of data. For instance, one model may always underestimate the price of one specific type of house while the other model may overestimate; so we may improve the prediction by averaging them.

We tried four methods for stacking: a simple weighted linear combination, adding other methods' predictions into feature matrix, stacking using the lasso, and the `caretEnsemble` package for `R`. The predictions produced by the later two methods were eventually chosen to be our official submissions. We preferred them over the other two stacking methods because the weights on the predictors were estimated from the data, not arbitrarily chosen. We did not consider any nonlinear combinations of the predictors.

Random forest predictions were excluded in model stacking for two reasons. First, random forests produced the largest cross-validation error among the four methods. The other reason is that random forests and tree-based XGBoost are both built by regression trees in this problem, so we suspect that their predictions may be highly correlated and thus using both in stacking would be somewhat redundant.

Lasso regression was applied to find a set of appropriate linear coefficients for stacking. Assume that y_i is the i-th case in the training dataset, and $\hat{y}_{i,\mathrm{xgb}}$, $\hat{y}_{i,\mathrm{pls}}$ and $\hat{y}_{i,\mathrm{elastic}}$ are the corresponding predictions of XGBoost, partial least squares, and elastic net. The solution of the following optimization problem was used for stacking:

$$\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1\hat{y}_{i,\mathrm{xgb}} - \beta_2\hat{y}_{i,\mathrm{pls}} - \beta_3\hat{y}_{i,\mathrm{elastic}})^2 + \lambda\sum_{j=1}^{3}|\beta_j|,$$

where λ was selected using cross-validation (10-fold, repeated 10 times). The resulting final predictor produced by the lasso was of the form $0.0312 + 0.5016\hat{f}_{\mathrm{xgb}} + 0.2480\hat{y}_{\mathrm{pls}} + 0.2329\hat{y}_{\mathrm{elastic}}$. Since this cross-validation procedure did not include producing the individual predictors, we do not have a valid cross-validation error. The public Kaggle score for this predictor was 0.11774 (Table 2).

The `caretEnsemble` package performs cross-validation to combine predictors using linear regression. For each cross-validation training set, it fits the individual models then takes the resulting predictors and fits a linear model to them. Tuning parameters are selected via this process. Following our previous rationale, we used the elastic net, partial least squares, and XGBoost methods for stacking. Due to time constraints, we did not include interactions or quadratic terms for the elastic net, nor did we fully search over all possible tuning parameters. In fact, we held tuning parameters constant for XGBoost as it was the most computationally intensive method. For partial least squares, we only considered the tuning parameter ranging from one to ten, and for the elastic net, we only searched for $\alpha \in \{0, 0.3\}$ and $\lambda \in \{0.001, 0.091\}$. We used 10-fold cross-validation repeated 10 times, and did not omit any training cases. The final predictor produced by `caretEnsemble` was of the form $-0.4946 + 0.2726\hat{y}_{\text{xbg}} + 0.2795\hat{y}_{\text{pls}} + 0.4952\hat{y}_{\text{elastic}}$ and had a cross-validation error of 0.12336 and a public Kaggle score of 0.12284 (Table 2).

We selected the predictions produced by Lasso stacking and `caretEnsemble` stacking to use as our final submissions. While the former had a better Kaggle public score, we had a cross-validation error for the later and thus some justification that it would provide a good Kaggle private score. The cross-validation error and both private and public Kaggle scores are shown in Table 2.

Summary and Conclusions

Our two official submissions reflect a combination of our desire to stack predictions in an honest, cross-validation based manner with our hope of obtaining the best score possible. Because caret ensemble's stacking is guided by cross-validation, it serves as a reliable gauge of predictive performance. The lasso-based stacking procedure discussed in the previous section resulted in predictions that ranked considerably higher on Kaggle's leaderboard. We were not able to perform legitimate cross-validation for the lasso predictions due to time and computational complexity.

The performance of the caret ensemble technique largely held up, achieving very similar public and private Kaggle scores as it did through cross-validation. We are not surprised that the private score for the lasso-based stacking method is inferior to the public score. Having not been able to use legitimate cross-validation this stacking procedure, we suspected that it might overfit the training data along with the portion of the the test data used to produce the public score. However, the magnitude of the difference between the public and private scores is striking. This is not to say that our lasso-based stacking was a bad idea, as it did very slightly outperform caret ensemble in terms of private score, but the hope that it would provide a substantial boost proved to be wishful thinking.

It is interesting to note that the predictions that achieved our best private Kaggle score (which were not selected in our two official choices) were made without excluding any potential outliers. This was a topic of much discussion within our group, as we did not see a strong rational for excluding the four largest houses, but consistently obtained better public scores when we excluded them. Ultimately, our official predictions based on caret ensemble included all houses, while our lasso-based stacking predictions did not. When the private scores were revealed, the improvement in our public score that resulted from dropping these four cases did not hold up. This experience provides a lesson on the risk of making ad hoc decisions rather than relying on precise rules that can be assessed through cross-validation.

We found the presence of the public Kaggle leaderboard to be something of a curse. While we did not consciously try to "game" the Kaggle system, it is difficult to not be drawn toward those techniques which are performing best on the leaderboard and this might have influenced some of our decisions on feature selection, model stacking, and the treatment of possible outliers. In the end, we were able to use and modify the methods available in caret to set tuning parameters and stack models using legitimate cross-validation. These methods led to cross-validation scores that proved to be reliable predictors for both public and private scores. More subjective decisions, such as as removing outliers, and our choice of model stacking, that initially seemed to be justified by strong public scores did not hold up for the private scores. Our results underscore the importance of establishing precise rules and testing them carefully through cross-validation. If this is not done, seemingly logical decisions guided, perhaps even subconsciously, by the entire training (or public test) data lead to results that are too good to be true.

Ultimately, we were able to produce two official sets of predictions that were competitive in Kaggle's public and private scoring. Though neither proved to be our best private score, the caret ensemble predictions that were built using careful cross-validation performed right on target with our expectations. The lasso-based stacking predictions achieved roughly the same performance, failing to realize the promise of its superior performance on the public data. More careful cross-validation in the stacking approach, as well as a systematic approach to dealing with outliers would have likely helped us realize this public score was too good to be true. Though legitimate cross-validation sometimes presents a computational challenge, our hands-on experience with this project highlights just how essential it truly is, further driving home a point that was emphasized throughout stat 602 and the discussion of this competition.

Appendix

Table 2: Cross-validation (10-fold, repeated 10 times) RMSE (root mean square error, based on the log sale price) and Kaggle scores for the four individual prediction methods we considered as well as the two stacking methods used for our final submissions.

Method	CV RMSE	Kaggle Public Score	Kaggle Private Score
Elastic Net	0.13384	0.13309	0.14449
XGBoost	0.11607	0.12362	0.12514
Random Forest	0.14342	0.14174	0.13476
PLS	0.12482	0.12290	0.12654
Caret Ensemble (E.Net, PLS, XGBoost)	0.12336	0.12284	0.12310
Lasso Stacking (E.Net, PLS, XGBoost)	?	0.11774	0.12291

References

Bergstra, James, and Yoshua Bengio. 2012. "Random Search for Hyper-Parameter Optimization." *Journal of Machine Learning Research* 13: 281–305.

Breiman, Leo. 2003. "Manual—setting up, and Understanding Random Forests V4.0." https://www.stat.berkeley.edu/~breiman/RandomForests/.

Carbonati, Tanner. 2017. "Detailed Data Analysis and Ensemble Modeling." https://www.kaggle.com/

tannercarbonati/detailed-data-analysis-ensemble-modeling/.

Jones, Donald, Matthias Schonlau, and William Welch. 1998. "Efficient Global Optimization of Expensive Black-Box Functions." *Journal of Global Optimization* 13: 455–92.

Kuhn, Max. 2016. "Bayesian Optimization of Machine Learning Models." http://blog.revolutionanalytics.com/2016/06/bayesian-optimization-of-machine-learning-models.html.

Snoek, Jasper, Hugo Larochelle, and Ryan Adams. 2012. "Practical Bayesian Optimization of Machine Learning Algorithms." *Advances in Neural Information Processing Systems.*