Prediction of Ames House Prices

Emily Goren, Andrew Sage, Haozhe Zhang

May 5, 2017

Introduction

We'll do this last.

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,[1] 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

[1] Carbonati, Tanner (2017) "Detailed Data Analysis and Ensemble Modeling." Retrieved from https://www.kaggle.com/tannercarbonati/detailed-data-analysis-ensemble-modeling/

(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 forests 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, say $\boldsymbol{T}$. 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 $\boldsymbol{X}_k^*$ $(k = 1, \ldots, 10)$. Training was then performed based on $\boldsymbol{X}_k^*$. 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 rate, but also a larger Kaggle score. For elastic net regression, the "honest" method repeated (10 times) cross-validation produced a RMSE of 0.17409 and a Kaggle 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 because we are not certain our code was working correctly. Due to the coding challenges involved in implementating the custom models, for the remainder of this paper we will use the "default" cross-validation method.

Model Tuning

Elastic Net Regression

Emily

Extreme Gradient Boosting

E**x**treme **G**radient **Boost**ing (XGBoost) belongs to a family of boosting algorithms that convert 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 hyperparameters compared with 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 tuning parameters. Each hyperparameter plays a significant

role in the model's predictive performance. Before hypertuning, let's first understand these parameters and their importance.

1. *max_depth*: the depth of the tree. Larger the depth, more complex the model; higher chances of overfitting. There is no standard value for max_depth. 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 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 eta leads to slower computation with an increase in *nrounds*.

Fine-tuning XGBoost by exploring the 7-dimension space in an efficient is a big challenge in terms of computation. There are a lot of methods and literatures about searching the global minimizer, to name a few, such as random search[2], bayesian optimization[3,4], and efficient global optimization of expensive black-box functions[5]. Due to the limitation of time and computing hardware ability, we customized the grid of tuning parameters as follows, based on our previous experiences and professional guides.

```
xgb_grid = expand.grid(nrounds = c(2000, 4000, 8000),
                       eta = c(0.01, 0.005, 0.001),
                       max_depth = c(2, 4, 6, 8, 10),
                       colsample_bytree=c(0.8,1),
                       min_child_weight = c(2, 3),
                       subsample=c(0.6, 0.8, 1),
                       gamma=c(0,0.01))
```

A repeated 10-fold cross validation were conducted to search over the above grid on condo2017 server that allows paralleling 16 jobs. It took 10 hours to finish running the code. The cross validation error is 0.1160675. The final values of parameters used for the model 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

Andrew

Random Forest

Andrew

[2]Bergstra, James, and Yoshua Bengio. "Random search for hyper-parameter optimization." Journal of Machine Learning Research 13.Feb (2012): 281-305.

[3]Snoek, Jasper, Hugo Larochelle, and Ryan P. Adams. "Practical bayesian optimization of machine learning algorithms." Advances in neural information processing systems. 2012.

[4]http://blog.revolutionanalytics.com/2016/06/bayesian-optimization-of-machine-learning-models.html

[5]Jones, Donald R., Matthias Schonlau, and William J. Welch. "Efficient global optimization of expensive black-box functions." Journal of Global optimization 13.4 (1998): 455-492.

Model Stacking

Haozhe

Summary

We'll do this last.