掃文資訊

2017-11-20 / **www.r-bloggers.com**

Dkqoe j c Lne_ao e j =i ao(Eks]6G] cchaто =` r] j _a` Nacnaooekj ?ki l apepekj

Introduction

Residential real estate prices are fascinating… and frustrating. The homebuy - er, the home-seller, the real estate agent, the economist, and the banker are all interested in housing prices, but they’ re sufficiently subtle that no one per - son or industry has a complete understanding.

For our third overall project and first group project we were assigned Kag - gle’ s Advanced Regression Techniques Competition . *The goal, for the projec t and the original competition, was to predict housing prices in Ames, Iowa.* Wh ile this particular competition is no longer active, the premise proved to be a v eritable playground for testing our knowledge of data cleaning, exploratory da ta analysis, statistics, and, most importantly, machine learning. What follows is the combined work of Quentin Picard, Paul Ton, Kathryn Bryant, and Hans Lau.

Data

As stated on the Kaggle competition description page, the data for this projec t was compiled by Dean De Cock for educational purposes, and it includes 79 predictor variables (house attributes) and one target variable (price). As a re - sult of the educational nature of the competition, the data was pre-split into a training set and a test set; the two datasets were given in the forms of csv file s, each around 450 KB in size. Each of the predictor variables could fall under one of the following:

- lot/land variables
- location variables
- age variables
- basement variables
- roof variables
- garage variables
- kitchen variables
- room/bathroom variables
- utilities variables
- appearance variables
- external features (pools, porches, etc.) variables

The specifics of the 79 predictor variables are omitted for brevity but can be found in the data_description.txt file on the competition website. The target variable was Sale Price, given in US dollars.

Project Overview (Process)

The lifecycle of our project was a typical one. We started with data cleaning and basic exploratory data analysis, then proceeded to feature engineering, individual model training, and ensembling/stacking. Of course, the process in practice was not quite so linear and the results of our individual models alerted us to areas in data cleaning and feature engineering that needed improvement. We used root mean squared error (RMSE) of log Sale Price to evaluate model fit as this was the metric used by Kaggle to evaluate submitted models.

Data cleaning, EDA, feature engineering, and private train/test splitting (and one spline model!) were all done in R but we used Python for individual model training and ensembling/stacking. Using R and Python in these ways worked well, but the decision to split work in this manner was driven more by timing with curriculum than by anything else.

Throughout the project our group wanted to mimic a real-world machine learning project as much as possible, which meant that although we were given bot

h a training set and a test set, we opted to treat the given test set as if it were "future" data. As a result, we further split the Kaggle training data into a private training set and a private testing set, with an 80/20 split, respectively. This allowed us to evaluate models in two ways before predicting on the Kaggle test data: with RMSE of predictions made on the private test set and with cross validation RMSE of the entire training set.

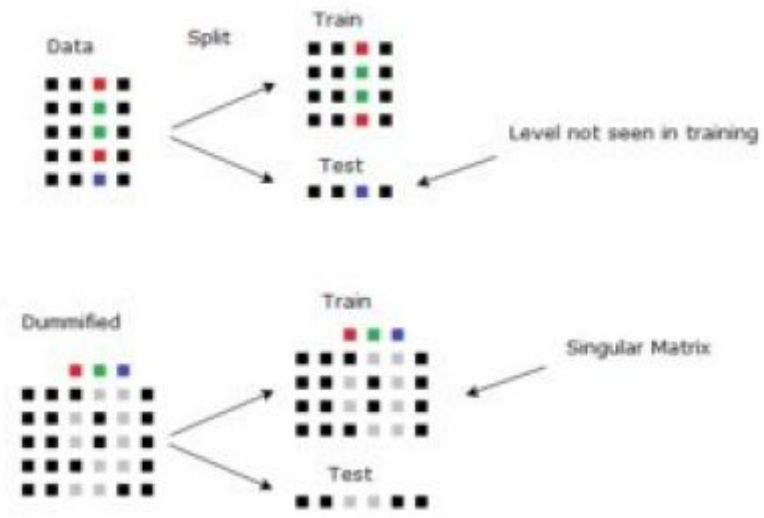
Given the above choices, the process for training and evaluating each individual model was broken down as follows:

1. Grid search to tune hyper parameters (if applicable) to private train set
2. Fit model to private train set with tuned parameters
3. Predict on private test set; if RMSE okay, proceed.
4. Fit new model to Kaggle train set with private train hyperparameters
5. Cross-validate on Kaggle train set; if CV RMSE okay, proceed.
6. Predict on Kaggle test set
7. Submit predictions to Kaggle

Computing RMSE of predictions made on the private test set served as a group sanity check, in that if anything was amiss with a model we found out at this point and could correct for it before proceeding. The decision to fit a new model to the Kaggle train in step 4 set using the *private train* hyperparameters found in step 1 was one for which we never felt completely at ease; we were trying to minimize overfitting to the Kaggle training set, but also recognized that we weren't using all the possible information we could by not re-tuning using the full training set. One benefit to this choice was that we never saw any major discrepancies between our own cross validation scores in step 5 and the Kaggle scores from step 7. With more time, we would have further investigated whether keeping the private train parameters or re-tuning for the final models was more beneficial.

The last noteworthy choice we made in our overall process was to feed differently-cleaned datasets to our linear-based models and our tree-based models. In linear-based models (linear, ridge, LASSO, elastic net, spline), prediction v

alues are continuous and sensitive to outliers so we opted to sacrifice infor - mation about "rare" houses in favor of gaining better predictions on "com mon" houses. We also wanted to minimize the likelihood of rare levels only sh owing up in our test data and/or causing columns of all 0's in dummifed dat a. We executed this tradeoff by releveling any nominal categorical variables th at contained extremely rare classes, where "rare" was defined to be "oc - curing in less than 1% of the observations."



Housing Prices in Ames, Iowa: Kaggle's Advanced Regression Competition

For example, the Heating variable had six levels but four of them combined ac counted for only about 1% of the observations; so, we combined these four le vels into a new 'other' level so that releveled Heating had only three levels, all accounting for 1% or more of the observations. Depending on the variable, we either created an 'other' level as in the example or we grouped rare level s into existing levels according to level similarity in the variable documentation.

We opted not to relevel data fed into our tree-based models because tree pre dictions are more robust to outliers and rare classes; trees can separate rare observations from others through splits, which prevents common observation predictions from being distorted by rare observations in fitting.

Data Cleaning and EDA

We were fortunate with the Kaggle data in that it came to us relatively clean. The only basic cleaning tasks were to correct typos in levels of categorical variables, specify numeric or categorical variables in R, and rename variables beginning with numbers to satisfy R's variable name requirements. There were a number of "quality" and "condition" variables that had levels of Poor, Fair, Typical/Average, Good, and Excellent which we label encoded as integers 1-5 to preserve their inherent ordinality. For nominal categorical variables, we used one-hot encoding from the 'vtreat' package in R. (Most machine learning algorithms in Python require all variables to have numeric values, and although R's machine learning algorithms can handle nominal categorical variables in non-numeric/string form, R's computations effectively use one-hot encoding under the hood in these cases.)

After taking care of these basic cleaning issues, we needed to address missingness in our data. Below is a plot that shows the variables containing missing values and the degree to which missingness occurs in each (shown in yellow):

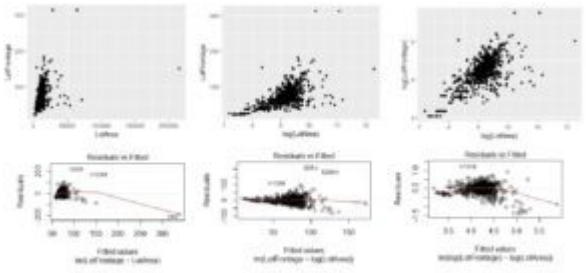
Housing Prices in Ames, Iowa: Kaggle's Advanced Regression Competition

For all variables except Garage Year Built and Lot Frontage, we performed basic mode imputation. Mode imputation was chosen for simplicity and because it could be done with both categorical and numerical data. Of course, mode imputation has the negative side effect of artificially decreasing variance within the affected variable and therefore would not be appropriate for variables with

higher degrees of missingness. It was for this reason, in fact, that we ap - proached missingness differently for Garage Year Built and Lot Frontage.

For missing values in Garage Year Built, we imputed the Year Built for the hou se. We justified this because most garages are built at the same time as the h ouse, and houses without garages get no penalty or benefit by having the Gar age Year Built equal to the Year Built for the house.

For Lot Frontage, the variable with the greatest degree of missingness, was re searched and explored in order to arrive at an imputation strategy. Lot Fronta ge was defined to be the linear feet of street connected to the property. Given that most properties were either regularly or only slightly irregularly shaped ac cording to Lot Shape, we deduced that most properties would have Lot Fronta ge values that were correlated with Lot Area values. However, since length is measured in units and area is measure in *square* units, we found it most ap - propriate to relate log(Lot Frontage) with log(Lot Area), so as to get a linear rel ationship between the two. See plot below.

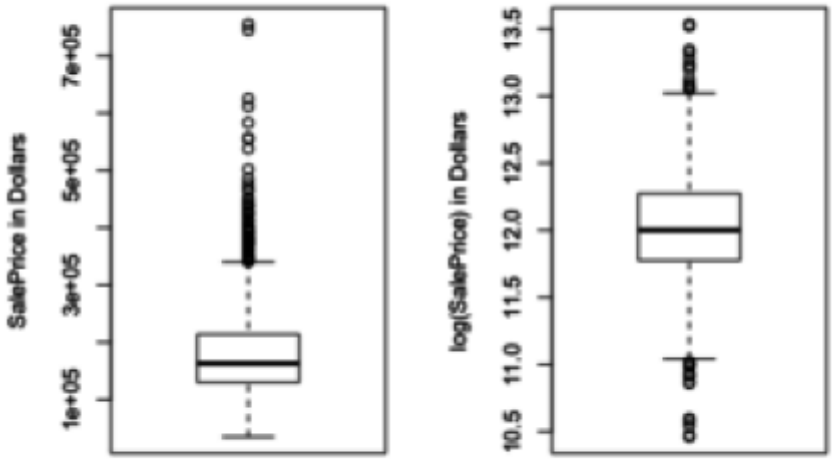


Housing Prices in Ames, Iowa: Kaggle's Advanced Regression Competition

Thus, we imputed missing values for Lot Frontage by fitting a linear model of l og(Lot Frontage) regressed onto log(Lot Area), predicting on the missing val - ues for Lot Frontage using Lot Area, and then exponentiating (inverse log-ing) the result.

The next step in EDA after finding and addressing missingness was to look at outliers. Looking at boxplots of both Sale Price and log(Sale Price) we saw that there were quite a few outliers, where 'outliers' mathematically defined to b

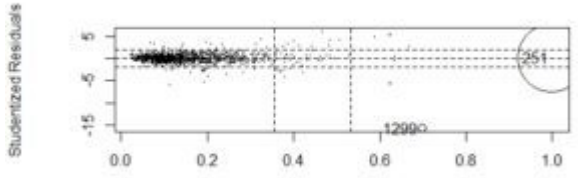
e observations lying more than 1.5*IQR (Inner Quartile Range) above the third quartile and below the first quartile.



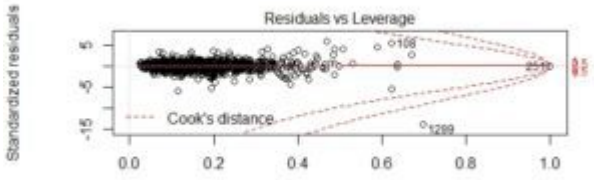
Housing Prices in Ames, Iowa: Kaggle's Advanced Regression Competition

Although removing the outliers may have improved our predictions for aver - age-priced homes, we were hesitant to do so due to the relatively small size of our sample (~1600 observations in Kaggle training set). We felt that removing any outliers without further justification than them simply being outliers would likely jeopardize our predictions for houses in the extremes.

Since outliers would have the most impact on the fit of linear-based models, w e further investigated outliers by training a basic multiple linear regression mo del on the Kaggle training set with all observations included; we then looked at the resulting influence and studentized residuals plots:



Housing Prices in Ames, Iowa: Kaggle's Advanced Regression Competition



Housing Prices in Ames, Iowa: Kaggle's Advanced Regression Competition

From these, we saw that there were only two observations that could justifi - ably be removed: observation 1299 and observation 251. These were both be yond or on the lines representing Cook's Distance, meaning that as individual observations they had a significant impact on the regression formula; as such, we removed these two observations from consideration.

The last bit of preprocessing we did was dealing with multicollinearity. This, as for outliers, was cleaning done mostly for the benefit of linear-based models; i n fact, it was done *only* for vanilla multiple linear regression since regulariza - tion in ridge, LASSO, and elastic net models deals with collinearity by con - struction. To eliminate collinearity issues, we used the findLinearCombos() fun ction from the 'caret' package in R on our dummified data. This function id entified linear combinations between predictor variables and allowed us to easi ly drop linearly dependent variables.

Feature Engineering

For feature engineering we took a two-pronged approach: we used anecdotal data/personal knowledge and we did research.

- **Garage interaction:** Garage Quality * Number of Cars Garage Holds.
  - If a home has a really great or really poor garage, the impact of that quali ty component on price will be exacerbated by the size of the garage.

- **Total number of bathrooms:** Full Bath + Half Bath + Basement Full Bath + Bas ement Half Bath.
- In our experience, houses are often listed in terms of total number of bedrooms and total number of bathrooms. Our data had total number of bedrooms, but la

cked total number of bathrooms.

- **Average room size:** Above-Ground Living Area / Total Number of R00ms Abov e Ground.
- "Open concept" homes have been gaining popularity and homes with large ro oms have always been popular, and with the provided variables we believe avera ge room size might address both of these trends.
- **Bathroom to room ratio:** (Full Bath + Half Bath) / Number of Bedrooms Above Ground
- The number of bathrooms desired in a house depends on the number of bedro oms. A home with one bedroom will not be viewed nearly as negatively for havin g only one bathroom as would a house with three bedrooms.
- **Comparative size of living area:** Above-Ground Living Area / mean(Above-Gro und Living Area)
- This variable attempts to capture house size as it directly compares to other ho uses, but in a manner different from mere number of rooms.
- **Landscape-ability interaction:** Lot Shape * Land Contour
- Landscaped homes tend to sell for more (see this article) and the ability for a l ot to be easily landscaped is, in part, determined by the shape of the lot (regula r, slightly irregular, irregular, very irregular) and the land contour (level, banked, hillside, depressed). Certain combinations may be workable (regular shape, hillsi de) while other combinations (very irregular shape, hillside) may make landscapi ng difficult and/or expensive.

Of the six features that we added, only "landscape-ability" resulted from res earch; we either already had the important industry variables in the data (total above-ground square footage or neighborhood, for example) or we did not ha ve the information we needed to create the desired variables. Additional fea - tures we would have liked to have added include geolocation data for proxim - ity to schools and shopping centers, quality of nearby schools, and specific ro oms/house features remodeled (if applicable).

One step that we glossed over a bit was extensive adding and removing of fea tures. We noticed with just a few trials that removing existing features seemed to negatively impact the performance of our models and that the models im -

proved when we added features. Given that the regularized linear models would give preference to more important features via larger coefficients and tree-based models would give preference to more important features by splitting preferences, we opted not to spend too much time manually adding and removing features. Essentially, we allowed our models to select the most predictive features themselves.

Modeling

For our individual models, we trained the following:

- Multiple linear regression
- Ridge regression
- LASSO regression
- Elastic net regression
- Spline regression
- Basic decision tree
- Random forest
- Gradient boosted tree
- XGBoosted tree

Overall our models consistently had RMSE values between .115 and .145. As stated earlier, our cross validation scores were usually very close to our scores on Kaggle. To our surprise, our linear-based models did very well compared to our tree-based models. Even the much-hyped XGBoost model was at best on par with our linear-based spline and elastic net models, and our random forest models tended to be much worse. An abbreviated table of our results is shown below:

	Elastic Net	Spline	Random Forest	Gradient Boost	XGBoost	Ensemble
Cross Validation	0.12157	0.11181	0.14171	0.12491	0.12282	0.11227

Held-out Test	0.11762	0.11398	0.13834	0.11403	0.11485	n/a
Kaggle	0.12107	0.11796	n/a	n/a	n/a	0.11710

As we endeavored to find a final, "best" model via ensembling and stacking, we followed the advice of our instructor (and successful Kaggler) Zeyu by attempting to combine models with different strengths and weaknesses.

We took the predictions from each of our best base models (Spline, Gradient Boost, XGBoost) as the features to use for the next level. Our best result came from a weighted average of the three, with weights determined by a grid search. The blend of 76% Spline, 14.5% Gradient Boost, and 9.5% Xgboost gave us a 0.11227 RMSE on our training set and 0.11710 on Kaggle.

We also experimented with using a second level meta-model to do the stacking, but with a linear meta-model, the coefficients were highly unstable because the predictions were so closely related to each other, and with a gradient boost meta-model, we were unable to beat our best base model alone.

Conclusions

As mentioned above, we were surprised by the strength of our various linear models in comparison to our tree models. We suspect this had a lot to do with the data itself, in that Sale Price (or rather, log(Sale Price)) likely has a relatively linear relationship with the predictor variables. This highlights one of the most important takeaways from this project: that linear models have a real place in machine learning.

In situations like this, where performance between a simpler model and a more complex model are similar, the better interpretability and the ease of training of the simpler model may also prove to be deciding factors on model choice. In most cases, stacking multiple base-layer models into a second-layer is impractical, despite performing slightly better.

END

Kaggle 數據挖掘

READ THIS
