Project Description: Predicting Housing Prices

This Kaggle project predicts housing prices in Ames, Iowa with 79 housing features (2006-2010). A training set (only 1460 obs) and a test set (1459 obs) are provided. To prepare a clean dataset for predictions, I correct the two variables in the test set where the year it's sold is less than the year it's built. Drop unnecessary variables for predictions, such as indexes(ID). Then, I draw scatter plots to drop outliers in explanatory variables, draw distribution plots to study the label, and decided to perform a log transform as it's right-skewed. Depending on the characteristics of features, I fill in the missing values of both training and test sets by the median/mean/mode of those variables in the training set, or simply None/0 according to the provided data descriptions.

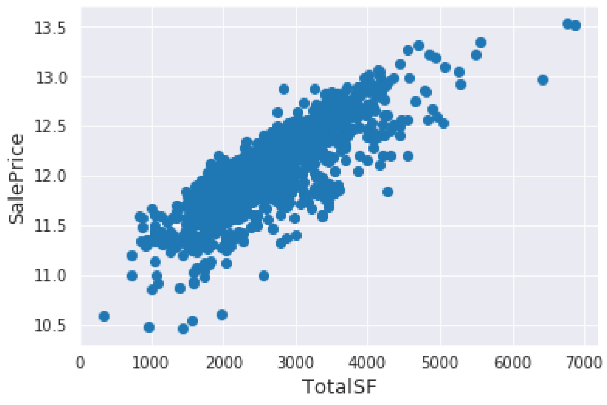

Figure 1: Scatter Plot--x axis total areas, y axis sale prices.

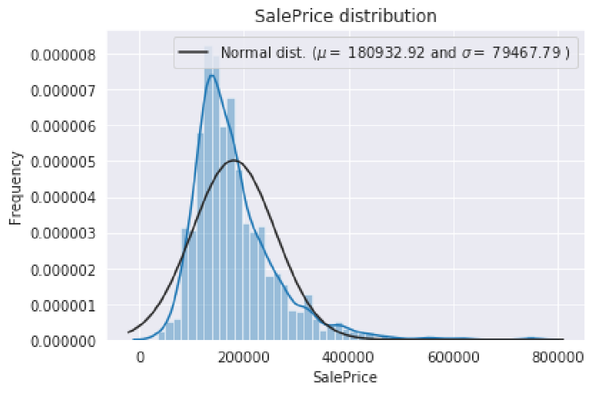

Figure 2: Distribution of "SalePrice" compared to Normal distribution.

With a cleaned dataset, I engineer categorical features and numerical features separately. A few categorical variables were coded as numerical in the training dataset, and they are transferred back to be categorical. Then, all categorical variables are label-encoded to be used in regressions. Since the order generated by label-encoder doesn't make sense regarding several variables, such as the names of neighborhoods, one-hot encoder is applied to label-encoded categorical variables (e.g., [0, 1, 0, 0, …, 0]). A potential problem with this method is that one-hot encoder eliminates the natural order of a categorical variable, such as "excellent," "good" and "poor." If I had more time, a better approach would be label encode categorical variables with orders and one-hot encode categorical variables without orders.

As for numerical variables, apply box cox transformation to skewed features so that they are more normally distributed. With 220 features, it's natural to perform PCA: standardize features and choose the principal components such that 95% of variance is retained in the training set, then apply them to the test set, which retains 143 features only.

To start modeling, first try a linear regression with 10 cross-validation folds. Its mean squared error is 0.2776. Since the number of observations of this training set is relatively small, a safe approach would be tune hyperparameters of a representative of different types of models, and stack them together with different weights as a meta model. Without spending too much time fine tuning hyperparameters with grid search, raw results are: Elastic Net, as the combination of Ridge and Lasso, has MSE less than any of the latter two. Random Forests perform better than a single decision tree, however, its accuracy is a little worse than Elastic Net (0.1876 > 0.1193). The XGB family performs the worst. XGBboost has an error of 8.512 with 3-fold cross validation. Accordingly, assign the most weights on Elastic Net and the least weight on XGBboost, I build a meta model with the final predicted values being the weighted average of these three different types of models. The final MSE is better than the MSE obtained from any of three models used.

Models	Linear Reg	Elastic Net	Random Forest	XGBboost	Meta Model
Score: MSE	.2776	.1193	.1883	8.512	.11549(test dataset)
Additional information	Largest error	Better than Lasso or Ridge	Better than a decision tree	Long runtime	.8*Elasticnet +.15*RDF +.05*XGB