CS273P Machine Learning & Data Mining

Project: House Prices - Advanced Regression Techniques

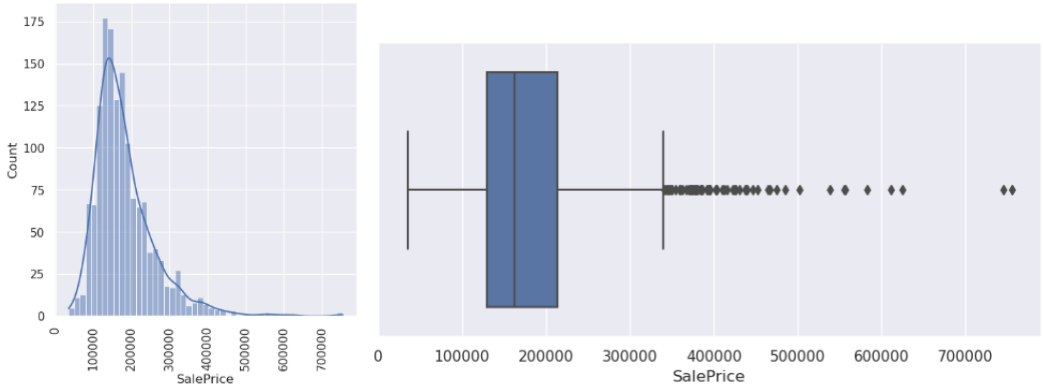
Final Report

Team 42

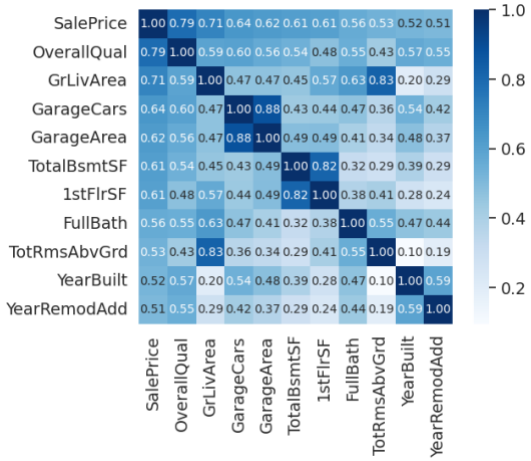
Antonio Rodriguez

Data Exploration

For my project, I decided on the House Prices - Advanced Regression Techniques Kaggle competition. For the competition, you are tasked with predicting the sale price of a house by using the 79 features given to you. I started off by looking at the data description text file and calling the info function to get a deeper understanding of the data. Looking at the summary, there are some variables that are missing a few entries, but some are missing a majority of their entries such as PoolQC and Fence. Next, I wanted to see how the house prices are distributed. From the histogram graph, we see that the sale price distribution is right skewed meaning there are some outliers present. To investigate the outliers further, I created a box plot and see there are a few in particular that are extremely alienated from the rest.



From there I focused on exploring the features. When first reading the feature descriptions, the features I thought would be very telling of the SalePrice were OverallQual, MSZoning, LotArea, Neighborhood, OverallCond, and Bedrooms. To test this I created a heatmap of all the features and SalePrice to see the correlations. There were 10 features that had above a 50% correlation value to SalePrice with the most correlated feature being OverallQual. When looking at a box plot between OverallQual and SalePrice, the higher the ratings the wider the ranges of the minimum and maximum are for each category. The surprising value I saw was OverallCond having a negative correlation value of - 0.078. The higher the quality of the house the lower the price is shocking. A feature I missed when making my initial predictions of the highest correlated features was GrLivArea (Ground Living Area). With the second-highest correlation, it had a clear linear relationship with SalePrice but did have two outliers.



Model Exploration

For my first model, I decided to go with XGBoost as my first model. XGBoost was a recommended model by Kaggle in the competition description. While we did not learn the XGBoost model in class, it is essentially the gradient boosting model we learn in class but is regularized which helps to prevent overfitting and speeds up the train time for the model. The hyperparameters I chose to tune were split into three groups to accommodate the amount of time to complete a grid search. The first group was max_depth and min_child_weight, the second group was eta, reg_lambda, and reg_alpha, the third group was colsample_bytree and subsample.

Name	Tested Values	Best Value
max_depth	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15	4
min_child_weight	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	4
eta	0.1, 0.05, 0.01, 0.005, 0.001	0.1
reg_lambda	0.0, 0.1, 0.2 , 0.3, 0.4, 0.5	0.0
reg_alpha	0.0, 0.1, 0.2 , 0.3, 0.4, 0.5	0.0
colsample_bytree	0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0	0.4
subsample	0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0	0.9

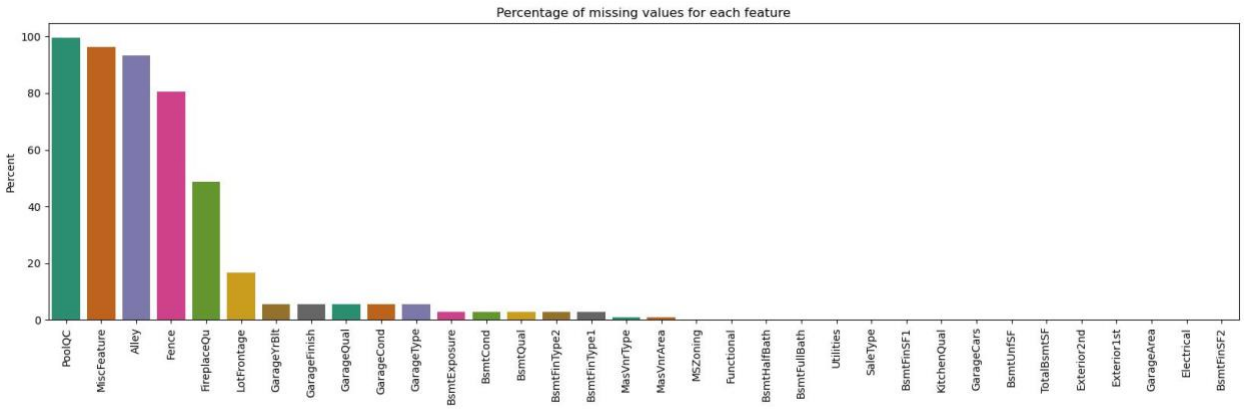
For my second model, I chose to go with another ensemble model but a bagging technique, Random Forest. The hyperparameters I chose to tune were split into two groups to accommodate the amount of time to complete a grid search. The first group was max_depth and max_features, second group was n_estimators, min_samples_leaf, and min_samples_split .

Name	Tested Values	Best Value
max_depth	10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150	70
max_features	‘auto’, ‘sqrt’	‘auto’
n_estimators	0.1, 0.05, 0.01, 0.005, 0.001	100

min_samples_leaf	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 25	1
min_samples_split	0.0, 0.1, 0.2, 0.3, 0.4, 0.5	0.001

Data Preprocessing and Feature Design

The first problem I wanted to address was locating any missing values. In the graph below I show the percentage of missing values for each feature.



There are 34 features with missing values. There were two approaches I tried to solve this problem: train the data while deleting all rows missing values (test one) and filling in the missing values (test two). To test whether it which approach was better, I created a basic XGBoost model using the default parameters to check the Root Mean Squared Log Error (the scoring technique Kaggle uses for this competition) and the R-squared value. For test one, the RMSLE score was 0.154 and the R-squared value was 0.879. For test two, the RMSLE score was 0.143 and the R-squared value was 0.889. While the value is not substantially different, I decided to with the model where I replace all the missing values to keep the maximum data to train the model.

The features were a mixture of categorical and numerical score values. When filling in the missing values, some didn't require any extra work, but others did. For the missing MSZoning values, since MSSubClass is not missing any values now, I used the group by operation and took the mode of that given MSSubClass and assigned it to that entry. Now that MSZoning is complete, it was used to assign the mode for the following values: 'Fence', 'LotFrontage', 'Utilities', 'Electrical', 'SaleType', 'Exterior1st', 'Exterior2nd'. For the following features the missing values were set to 'NA': 'MiscFeature', 'Alley', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond'. For the following features the missing values were set to 0: 'GarageYrBlt', 'TotalBsmtSF', 'BsmtFinSF2', 'BsmtUnfSF', 'BsmtFinSF1', 'BsmtHalfBath', 'BsmtFullBath', 'MasVnrArea', 'BsmtExposure', 'BsmtQual', 'BsmtFinType2', 'BsmtFinType1'.

When looking at the PoolQC, I used an if statement to make sure before setting the value to 'NA', there was no pool by checking if PoolArea is 0, and if it's not set the value to 'FA'. For

the following features, the missing values were set to the mode: 'KitchenQual', and 'Functional'. For 'MasVnrType', the missing values were set to 'None'. For 'GarageArea', 'GarageCars', I grouped them by 'Neighborhood', and 'GarageType' and assigned the mode. For 'TotalBsmtSF', if 'BsmtCond' is not zero set it to 'TA' else set it to 'NA'. After all the data cleanup, I use the pandas get dummies tool to set all the categorical features into indicator variables for the model to run better.

Performance validation

For this project, Kaggle uses RMLSE (Root Mean Squared Logarithmic Error) to score the accuracy of the model. While doing the hyperparameter tuning, the RMSLE was the score I used to determine which combination was best. While I started off with just Random Forest and XGBoost, I added the original Gradient Boosting model to test how much more accurate XGBoost was. From this I could see that boosting might fit the model better than bagging. On last model I tried was stacking technique using the StackingCVRegressor from the mlxtend library. For the StackingCVRegressor, the group of regressors I combined were Lasso, Ridge, Random Forest, XGBoost, and Gradient Boosting. The meta_regressor was XGBoost. Even with the stacking regressor, XGBoost still proved to be the beset model to predict SalePrice.

$$\text{RMSLE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i+1)-\log(a_i+1))^2}$$

Model	RMSLE	R-Squared
Random Forest	0.156206	0.878678
Gradient Boosting	0.142188	0.892072
XGBoost	0.137425	0.914571
Stacked	0.138610	0.898286

Adaptation to under- and over-fitting

For both models, to combat overfitting and underfitting. For the XGBoost model, to address overfitting hyperparameter tuning was used. The main parameters that help with this are max_depth, min_child_weight , eta, subsample and colsample_bytree. For these five parameters, lower numbers were better. The other two, reg_lambda and reg_alpha, both help underfitting that may occur do to the outliers. For Random Forest, the max_depth, min_samples_leaf and min_samples_split helped fight overfitting, the larger the number the more overfitting the model will be.