Kaggle Housing Project Report
Lawrence Chen, Yassin Elsharafi, Sandy Lin, Philip Moceri

When it comes to machine learning, predictive modeling stands as a cornerstone for extracting valuable insights and making informed decisions. This report delves into the intricacies of our approach to predicting house prices given by the Kaggle competition. The methodology adopted combines a suite of advanced regression techniques and harnesses the power of ensemble learning through stacking. By navigating through the implementation of Elastic Net, Gradient Boosting, Kernel Ridge Regression, Lasso Regression, and a sophisticated stacking model, this report explains the thought process behind the many decisions that led to our robust predictive model capable of accurately estimating house prices. Each section provides a simplified explanation of a portion of code as well as the decision-making processes involved in the creation of said code.

Generic Imports, Loading, and Exploring Data, Data Preprocessing

We begin our code by importing many commonly used libraries such as pandas and numerous regression models. After fetching our data from the csv files and placing that information into dataframes, we then display the heads of these data frames as well as their sizes. Now that we have our data, we can begin tweaking it to be more practical.

After viewing our data, we begin to preprocess it by dealing with the NaN values. For both train and test, we drop the ‘PoolQC’ and ‘MiscFeature’ columns due to the number of NaN values within them being more than 3 quarters of the data’s length. We then proceed to drop the ID column, as it contributes nothing to the data. We then view the SalePrice column and see that it’s skewed. To work around this, we filter the data and use the rows where GrLivArea is less than 4500, and then use that data to logarithmically transform SalePrice. From there, we combine train and test data in order to handle NaNs and missing values more easily. To ensure that the features MSSubClass, YrSold, and MoSold, which are categorical in nature, do not contain numerical values, we convert them all to string type. We fill the NaNs values in the Functional, Electrical, and KitchenQual columns with their most suitable values. Additionally, we impute missing values in the Exterior1st, Exterior2nd, SalePrice, and MSZoning columns with their respective most frequent values.We fill the garage NaN data, along with the rest of the numerical data with 0 and fill the basement NaN data, along with the rest of the categorical data with “None”.

Feature Engineering, Handling Categorical Features, Creating Train and Test Sets

The code proceeds with feature engineering, creating new features by combining existing ones. New features such as YrBltAndRemod, TotalSF, Total_sqr_footage, Total_Bathrooms, and Total_porch_sf were added to clarify which features each house has and doesn't have. Other column features such as haspool, has2ndfloor, hasgarage, hasbdmt, and hasfireplace were added to further clarify what features each house has. After adding in the new features, it is observed that it will be best to fill in the LotFrontage NaNs with some values. To do this, we will use features that seem to have a correlation with LotFrontage in order to train. We decided to use features such as LotArea, LotConfig, LotShape, MSZoning, BldgType, Neighborhood, Condition1, Condition2, and GarageCars, based on Kaggle's *LotFrontage Fill In Missing Values: House Prices* article by Oleksandr (Alex) Akulov. However, instead of using the SVR classifier as the article does, we attempted to use a gradient boosting regressor to train and predict the values to fill in the missing or NaN values in both the train and test datasets. From those values, we are able to see which values highly correlate with LotFrontage. From there, we created the final prediction column with dummies and normalized it. To process the categorical values, we decided to use OneHotEncoder to make them suitable for the ML models to interpret and utilize the categorical data effectively.

The datasets are split back into train and test sets after the necessary preprocessing steps. We performed feature scaling before the script initializes the models and sets up functions for cross-validation and model evaluation.

Model Initialization, Cross-Validation and Model Training, Model Stacking and Final Prediction

The script initializes a variety of machine learning models, including Lasso, Elastic Net, Kernel Ridge, Gradient Boosting, XGBoost, and LightGBM. These models will be trained and their predictions combined to achieve a more robust and accurate final prediction.

The script then utilizes cross-validation to train each model and evaluates their performance using the defined root mean squared logarithmic error (RMSLE) function.

The script then defines a stacking model, which combines the predictions from various base models. This ensemble approach helps leverage the strengths of each model and improve overall performance.

Concluding thoughts

Through the amalgamation of Elastic Net, Gradient Boosting, Kernel Ridge Regression, Lasso Regression, and a strategically devised stacking model, we were able to construct a surprisingly accurate tool for predicting house prices. Our final model received a score of

0.12221 from Kaggle, putting our score in the top 370s of the leaderboard. By monitoring our advancements through each algorithm, we highlighted the inherent tedium associated with feature selection and hyperparameter tuning. When reviewing the convergence of these methodologies, the success of the ensemble model in surpassing individual algorithms becomes apparent. Our methods perfectly demonstrate the dynamic nature of predictive modeling, showcasing both the sophistication of algorithms and the creativity embedded in crafting robust solutions. The predictive prowess attained here exhibits the possibilities within the realm of data science, reinforcing the perpetual evolution required to stay at the forefront of this transformative field.