Advanced Regression Techniques in Python

Predicting Housing Prices based on a given dataset using Multiple Regression Techniques

Brekken Bozzi
College of Engineering and Computer Science
Syracuse University
Syracuse, NY, United States of America
bjbozzi@syr.edu

***Abstract*—The purpose of this report is to show the various techniques I used in the *"House Prices - Advanced Regression Techniques"* Kaggle competition. In this report I will outline how I pre-processed the data given to me, which regression models I experimented with, tuning each model, and the results that led me to receive my best Kaggle submissions score.**

Keywords—machine learning, regression, linear regression, lasso regression, bagging regressor, random forest regression, gradient boosting regression, Python, Kaggle

I. Introduction

Throughout the 2023 Fall semester in CIS400 (Introduction to Machine Learning) at Syracuse, I have learned an adequate amount of material covering a variety of topics and techniques used in data science and machine learning. This report will cover my solution to the final course project's problem. The same data set was provided to everyone in the class through a data science competition website. Kaggle is a data science competition platform where users are given datasets and problem descriptions where they can use their knowledge to submit solutions which will be scored to possibly win prizes or recognition. This specific problem involved a dataset with multiple features describing previously sold house listings. Using this data, the goal was to generate some sort of regression model to predict the value of the "SalePrice" feature in the given test set.

II. Pre-Processing of Data

Given both a training and test set of data from Kaggle, the first step was to load both CSV files into my Python environment. When showing the headings for both data frames, you will see that the training data has one more feature than the test data. As I briefly mentioned in the introduction, this is the "SalePrice" column. It is important to define this column of data as the "y_train" data, or the dependent variable. To ensure the data is properly shaped, I dropped the "Id" column from the training data and concatenated both data sets together based on the rows. This will result in a merged data frame with 2919 rows. This merged data frame are the independent variables. The "Id" and "SalePrice" columns are once again dropped from the data frame.

Although the data is now properly shaped on the column axis, there is going to be a lot of missing and null data. To take care of this, I needed to fill in both the missed numeric and categorical variables for each feature. I chose to fill all missed numeric variables with zero (0), while I chose to fill categorical variables with the first category in the array of possible options. Once I did this, I was able to check for the sum of all null cells was zero, which allowed me to move onto the next step.

A common technique before creating a regression model for data is splitting all categorical variables to binary. This is done using the "get_dummies" function in the Pandas library. I performed this conversion to the merged data set of independent variables. This resulted in a total of 287 features.

The final step before I was able to start generating regression models was to re-align the data on the row axis. When I previously concatenated the train and test data sets, there were 2919 resulting rows, while the Kaggle submission is only expecting 1459 rows. This is because the data still needs to be split into the "X_train" and "X_test" manually. Using the length of the previously defined "y_train" data, I took the first 1460 rows of data for the "X_train" data which will be used to train the regression model. The remaining 1459 rows are assigned to "X_test", which is the data set that will be applied to the model for new dependent variable (SalePrice) predictions.

III. Methods of Regression

Now that the data is properly pre-process, I was able to start training regression models using the "X_train" and "y_train" data sets. For this data, I chose to use five of the common regression methods we learned over the course of the class. These models include a simple Linear regression, Lasso regression, Bagging regression, Random Forest regression, and Gradient Boosting regression. Each of these models use their own unique parameters to tune predictions.

My initial approach involved using the default parameters for each method of regression with a random state set to 42. Using the "score" function, I was able to print out the calculated baseline accuracy for each of the five models. The score should be a value between zero (0) and one (1). My Bagging Regressor model started giving me some problems towards the end of the project, and I can't seem to figure out why. The score value is far outside of the specified bounds.

Below you will find Table I, showing the initial score results from my un-tuned regression models. Once again these scores were generated using only default parameters.

TABLE I. UN-TUNED REGRESSION SCORES (PYTHON)

Baseline Regression Model Scores	
Model Name	***Score***
Linear Model	0.920500941501453
Lasso Model	0.9204858563241928
Bagging Model	-48177.15022627854
Random Forest Model	0.9793143499024441
Gradient Boosting Model	0.9649663536782351

Given these initial baseline scores, I knew there was room for improvement, although I was still happy with these results. At this point in experimenting, it was already quite clear to me that my best Kaggle score was going to be generated from a Random Forest or Gradient Boosting regression model.

Now that I had models trained, it was time to create parameter grids where I could use the "GridSearchCV" functions to loop through each model to obtain the best possible score. For scoring I chose to use the "neg_mean_squared_error" parameter within the grid search function considering it heavily penalizes over-fitting of the model. Each regression model has different parameters, so I chose the more common ones and input various values into the parameter grid to ensure it covered the entire spectrum of possible scores. Below in Table II are the scores obtained from the automatically tuned regression models using the grid search method.

TABLE II. TUNED REGRESSION SCORES (PYTHON)

Best Regression Model Scores	
Model Name	***Score***
Linear Model	0.9205011531740586
Lasso Model	0.9181444993732942
Bagging Model	-48177.15022627854
Random Forest Model	0.97737596433221
Gradient Boosting Model	0.9978600115531687

As you can see, in most cases the score has improved among the models. Some of the more important parameters I chose to tune were "n_estimators", "max_depth", "learning_rate", "min_samples_split", and "min_samples_leaf". These parameters seemed to have the greatest effect on the model score in my experimenting.

IV. KAGGLE SUBMISSION

After tuning the models, I created separate Kaggle submission files for each. This was done by creating an "Id" column, which was taken from the original test data set of 1459 rows and a "SalePrice" column from the model predictions.

I ended up with five submission files, one for each tuned regression model. I made a separate submission for each to Kaggle. I have provided Table III, which shows the Kaggle score for each submission file based on my initial "best score" for each tuned regression model shown in Table II.

TABLE III. TUNED REGRESSION KAGGLE SCORES

Best Kaggle Scores by Model	
Model Name	***Score***
Linear Model	0.53186
Lasso Model	0.53133
Bagging Model	2.52367
Random Forest Model	015404
Gradient Boosting Model	0.1391

Based on the Kaggle scores, with the lower number being a more accurate prediction, the Gradient Boosting Regression model was the clear winner. I opted to go ahead and manually try to more accurately tune this model to get a better submission score.

Doing this, I found out that the best model score calculated in Python doesn't necessarily mean a better Kaggle prediction score. An overfit model will generate a better score in Python, but a lesser score on Kaggle. With many more submissions to Kaggle, I found that the best parameters for this Gradient Boosting Regression model are:

```
gb_model_final =
GradientBoostingRegressor(n_estimators=150,
random_state=42, min_samples_split=10,
min_samples_leaf=10, max_depth=5)
```

This code provided me with a **model score of 0.985** and a final **Kaggle score of 0.13277**, which was my most accurate submission. This placed me right around 1300th on the leaderboard of nearly 5,400 competitors trying to predict the same values.

V. CLOSING REMARKS

I greatly enjoyed this project and cherish everything I have learned throughout CIS400 Introduction to Machine Learning at Syracuse. I certainly have a newfound interest and basic understanding of data science.

If I were to do this project again, I think I could've done some more research to find other methods to possibly improve my score and position on the Kaggle leaderboard. I thoroughly enjoyed the competition this project brought to the classroom, and I was more than happy to complete this on my own (without a team).

Although I hoped for a better Kaggle score, I am still proud of myself for being able to apply my knowledge of regression models to real data. I could easily see myself applying to a Data Engineering position once I finish my degree.