Contents

1 Introduction **1**

2 Summary of functionalities **1**

2.1 Possible Improvements 3

3 Repo Structure **3**

A Sources **3**

1 Introduction

The purpose of this project was to implement basic but robust regression techniques to predict some quantity from a dataset of correlated properties. The example discussed here revolves around predicting the housing price from a collection of house listings and their sale price. Initially, we are assumed to be given an uncleaned dataset, so the main work the script does is to clean the dataset. This is accomplished through a combination of basic imputations and machine learning (Random Forest Classifier) at user specified thresholds giving when we drop data or try and fill it in.

2 Summary of functionalities

The user specifies a quantity to predict and thresholds *Drop* and *BasicImpute* such that above threshold *Drop* we drop the data column and below threshold *BasicImpute* we use basic imputation functions to fill in missing values. See https://scikit-learn.org/stable/modules/impute.html for description of imputers. Essentially, if the column only has a small amount missing the missing values are filled in with the most frequent value from that data column if it is categorical, and with a multivariate imputer if the data is numerical.

What's left are columns with either no missing data or that are missing more than *BasicImpute* but less than *Drop*. For each column in the latter class, the script implements a random forest classifier to predict the missing values for that column based on the data in columns without any missing values. This is one of the first primary sources of error as here we are predicting values absent some possible correlations coming from the columns with missing values. This is why it's important to choose a *BasicImpute* that

is not too low, otherwise we would be predicting a lot of missing values from a minimal amount of clean data.This step is accomplished by two functions. One that returns a model (random forest) to predict the missing values, and one that encodes and predicts that missing data. When the predictions are made cross validation is done via a stratified k-fold cross-validator with $k = 6$. A stratified cross-validation is utilized to ensure we are measuring how accurate the model predicts housing prices of different sizes, not just large, small, medium prices etc. The validation values are reported and can be plotted in order to see the model's effectiveness. Average CV values for the housing dataset, where we are using a random forest to predict the 'MasVnrType', 'FireplaceQu' and 'LotFrontage' missing data were respectively 0.823, 0.771 and 0.462. This is in reverse order of how much of the data is missing from each column (i.e., 'LotFrontage' has the lowest amount missing). This indicates that the model predicting the 'LotFrontage' parameter is vastly overfitting. This could mean that our predicted values for 'LotFrontage' are not good and sense the ultimate prediction of SalePrice in the housing example depends on these values, it follows that this may be another source of errors in the predictions.

Finally, the cleaned data is concatenated into one dataframe and then split into the input data X and the data to predict y where y corresponds to a user specified column to predict. The script then makes an 80:20 train:test split on the cleaned X and y and then uses a CatBoost regressor to predict the quantity y. This may seem a novel piece of the algorithm, being relatively new (2017). However, the CatBoost regressor has several key advantages and utilities. Firstly, it is well suited to predicting values from a dataset with both numeric and non-numeric values. This saves some work, however with strictly numeric and categorical data we do not need to work that hard to change this into something the average machine learning algorithm can handle. However, the CatBoost regressor offers the benefit that it can take a dataset for which only a weak model can be generated and sequentially combine many weak decision tree models together into a more robust predictive model. The sequential nature of this approach means that the fitted trees will learn from the mistakes of the previous trees and reduce the errors. In this manner CatBoost regressor exhibits a "greedy" approach to regression on a multi-feature data set.

At this stage the script will compute the errors between the true and predicted values and return the mean squared error, the absolute error, and the R^2 score from the prediction. A plot of the true vs predicted values is also included. In the case of the housing dataset from Kaggle, this method will achieve an R^2 value of $\sim 0.90\%$.

2.1 Possible Improvements

Possible improvements could come from parameter tuning, though this will likely be specific to the dataset being studied. Tuning the thresholds of when to drop vs. impute vs. use machine learning classifiers to predict missing values is particularly dataset specific, and should be done only after initial data exploration. One could also tune the imputers to be more optimal, perhaps being more creative with the choices. The logic is that below the threshold at which basic imputation is used that basic imputation will not be much worse than a more sophisticated prediction. However, perhaps an appropriately chosen more sophisticated algorithm would lead to improvements, as these imputed values are later used to predict the bulk of the missing data.

Improving the accuracy of the random forests could also improve prediction accuracy. These predictors are in general not expected to perform well in this circumstance, given that they are predicting data that is mostly missing. It would still be interesting to explore using a largely number of estimators (currently using default value of 100) or even using boosted forests (such as from XGBoost). These methods become computationally expensive obviously, and may only give marginal gains.

3 Repo Structure

There are two main scripts: data_cleaning.py and predict_quantity.py. Their roles are essentially their titles. data_cleaning.py contains functions used in cleaning a dataset such as basic imputation functions, functions to collect/ delete null values, and the aforementioned functions which create and then use random forest classifiers to fill in other missing values. predict_quantity.py then takes in user inputed train/test data file paths, quantity to predict, and thresholds for deletion/imputation and calls the appropriate functions. It will also return heatmaps of the cleaned data before/after imputation as confirmation that the missing data is gone, as well as the cross validation results.

Additionally, there is a folder of datasets which in particular contains the Kaggle housing prices train/test dataset. There is a folder catboost_info containing the RMSE errors and run times of the CatBoost models generated during the model fitting of the regressor. There is a folder called images_ in which are saved plots of various metrics/validation scores.

A Sources

For more on the catboost regressor model: "Catboost regressor in 6 minutes"

For more on k-fold validation: StratifiedKFold

For more on the random forest classifiers: RandomForestClassifier

The dataset and the inspiration for many of the methods came from the Kaggle housing prices advanced regression techniques competition - see Housing Price Kaggle Competition