Brief Explanation of a prediction model of 1985 Auto Imports Database

Jihoon Kim

August 9, 2017

Abstract

In this project, I am asked to build a model to predict normalized losses, the relative average loss payment per insured vehicle year, of car insurance by using '1985 Auto Imports Database'. This document outlines my solution.

Overview

The objective of this project is training a prediction model to infer normalized loss ratio of automobiles. This project has four stages. First, in project setup stage, it prepares the data to be ready for data processing. Second, exploratory data analysis is conducted to visualize the data. In the third stage, a prediction model is implemented. Lastly, performance is recorded and visualized.

Contents

1 Datasets

This project uses '1985 Auto Imports Database'. You can see original data description at: `https://archive.ics.uci.edu/ml/datasets/Automobile`

1.1 Description

This data set consists of three types of entities:

- the specification of an auto in terms of various characteristics
- its assigned insurance risk rating
- its normalized losses in use as compared to other cars

The second rating corresponds to the degree to which the auto is more risky than its price indicates. Cars are initially assigned a risk factor symbol associated with its price. Then, if it is more risky (or less), this symbol is adjusted by moving it up (or down) the scale. Actuarians call this process "symboling". A value of +3 indicates that the auto is risky, -3 that it is probably pretty safe.

The third factor is the relative average loss payment per insured vehicle year. This value is normalized for all autos within a particular size classification (two-door small, station wagons, sports/speciality, etc...), and represents the average loss per car per year.

* Note: Several of the attributes in the database could be used as a "class" attribute.

1.2 Features

Attributes	Attribute Range
symboling	-3, -2, -1, 0, 1, 2, 3
normalized-losses	continuous from 65 to 256.
make	alfa-romero, audi, bmw, chevrolet, dodge, honda, isuzu, jaguar, mazda, mercedes-benz, mercury, mitsubishi, nissan, peugot, plymouth, porsche, renault, saab, subaru, toyota, volkswagen, volvo
fuel-type	diesel, gas
aspiration	std, turbo
num-of-doors	four, two
body-style	hardtop, wagon, sedan, hatchback, convertible
drive-wheels	4wd, fwd, rwd
engine-location	front, rear
wheel-base	continuous from 86.6 120.9
length	continuous from 141.1 to 208.1
width	continuous from 60.3 to 72.3
height	continuous from 47.8 to 59.8
curb-weight	continuous from 1488 to 4066
engine-type	dohc, dohcv, l, ohc, ohcf, ohcv, rotor
num-of-cylinders	eight, five, four, six, three, twelve, two
engine-size	continuous from 61 to 326
fuel-system	1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi
bore	continuous from 2.54 to 3.94
stroke	continuous from 2.07 to 4.17
compression-ratio	continuous from 7 to 23
horsepower	continuous from 48 to 288
peak-rpm	continuous from 4150 to 6600
city-mpg	continuous from 13 to 49
highway-mpg	continuous from 16 to 54.
price	continuous from 5118 to 45400.

1.3 Size

1. Number of Instances: 205
2. Number of Attributes: 26 total
   (a) 15 continuous
   (b) 1 integer
   (c) 10 nominal

1.4 Missing Values

Missing values are denoted as "?"

Attributes	Number of instances missing a value
normalized-losses	41
num-of-doors	2
bore	4
stroke	4
horsepower	2
peak-rpm	2
price	4

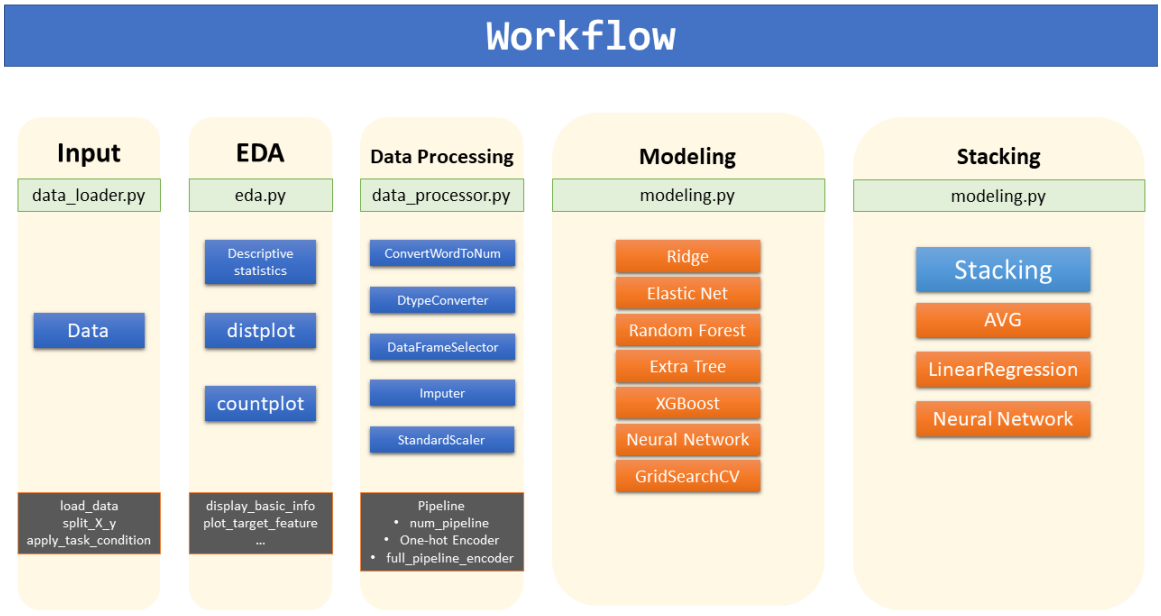
1.5 Instruction of Task

- Target Variable: Normalized-losses
- Missing values: denoted by quotation marks ('?'). Skip data samples with missing values in the target.
- Features to ignore: 'Symboling'

There are 41 missing values in the target variable, normalized-losses. Therefore only 164 instances are available.

Symboling feature, which is rated by actuaries, contains risk measures. This will be highly correlated to our target variable. This feature also will be dropped in analysis.

2 Workflow



3 Project Setup

In this stage, data is loaded into memory with the instructions of the task applied. 'data_loader.py' implements this task. 'load_data' function loads data. It also changes values of missing values from '?' to Numpy's NaN. This helps to count null values, especially through Pandas's 'isnull' function.

After loading the data on to the memory, it applies instructions of task. First, it drops the instances with null values in our target data (normalized losses). Next, it drops 'symboling' features, which should be ignored.

4 Exploratory Data Analysis

The 'eda.py' provides various plotting function for visualizing feature characteristics of data. It helps users to understand how the '1985 Auto Imports Database' looks like.

5 Data Processing

The 'data_processor.py' file processes data to be a input form of the prediction model. It uses scikit-learn's pipeline module. Since the dataset is composed of only 164 available instances with 26 features, I didn't use any dimensionality reduction technique.

5.1 ConvertWordToNum

This class is a custom transformer class that transforms 'number words', such as 'two' or 'five' to numerical values, '2' or '5'. It makes data processing easier rather than using theses numbers in string formats.

5.2 DtypeConverter

This is also a custom transformer class that converts data types.

5.3 DataFrameSelector

It changes Pandas DataFrame format to numpy ndarray format for the following processing.

5.4 Imputer

There are some missing values in data. It fills them with a median value of the feature.

5.5 StandardScaler

Some regression algorithms, such as ridge regression or elastic net regression, the scale of features affects a lot to their performance. Therefore all numerical features should be scaled before modeling.

5.6 Pipelining

The 'num_pipeline' combines all processing above into one single pipeline. It helps to preserve the integrity of processing when the same process applies to test set. The 'full_piepeline' combines both numerical processing and categorical processing (One-hot Encoding) into one process.

6 Modeling

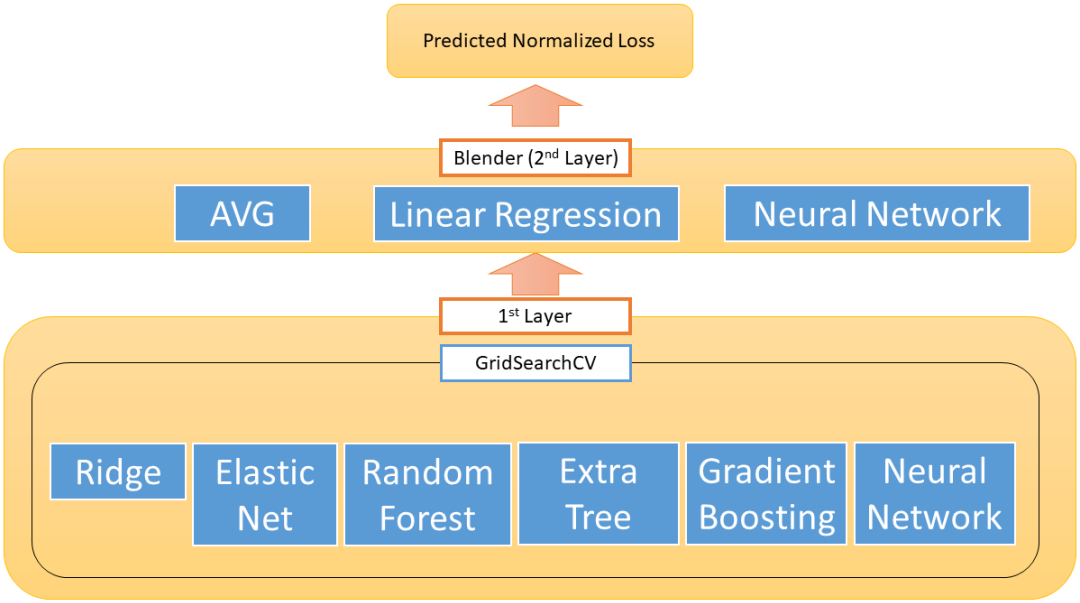
6.1 Single Estimator Approach

In this stage, the 'modeling.py' builds several prediction models to infer the target variable. Modeling is composed of two parts. First, it trains 6 regressors. (ridge regression, elastic net regression, random forest regression, extra tree regression, gradient boosting regression and neural network) Performance of each regressor is recorded in './log/' directory.

6.2 Application of Stacking

The performance of single model varies considerably for each trial. Therefore, instead of using single predictors to predict our target value, I stacked these single models to aggregate. I designed 2-layer stacking structure. In the first layer, 6 predictors are trained and second predictors use the outputs of the first layer as input values. The second layer is trained by using held-out set, which is a subset of a training set. (In this project, 30% of training set is used as a held-out set) Following algorithms are used for blending:

- Average Blender
- Linear Regression Blender
- Neural Network Blender



7 Conclusion

Considering the dataset is a small dataset, it shows pretty moderate performance. Empirically, simple linear or elastic net regression predicts better than other algorithms. But this seems to be the result of a limited number of data. (In many cases, test error is even lower than training error). In most cases, performance on the test set is between 15 to 25 in RMSE scale. I believe that this performance can be improved by training with a large dataset.