Louis Massera

Car Fuel Efficiency Prediction

The main goal of this work is to predict the fuel efficiency of car models in 2018 given their efficiency in 2015, 2016 and 2017.
We will also evaluate our final model for Task 1 and Task 3 making an i.i.d. assumption, i.e. splitting data at random.

Task 1: Linear Models

Training and test sets for this work were developed based on the csv files of Fuel Economy Guide that can be found at https://www.fueleconomy.gov/feg/download.shtml.
Training set consists of data from 2015, 2016, 2017 years, while test set covers 2018 year data.
Predict variable: "Comb Unrd Adj FE - Conventional Fuel”

Preprocessing steps included:

- deleting any columns and rows with 100% missing data.
- deleting any columns with more than 60% of missing data.
- getting rid of the features that are correlated with the predict variable. Any features containing ['FE', 'MPG', 'EPA', 'CO2', 'Guzzler', 'cost ', 'range', 'GHG', ‘Smog’] in its name were deleted. (cf. *)
- deleting all the description variables (containing ‘Desc’ in their names) since they are redundant and will not add any value for the purposes of our analysis.
- identifying types of remaining features, categorical, boolean or numerical.
- converting 'Release Date’ into unix timestamp.

Dealing with missing values:

To deal with missing values in the numerical data, in both training and test datasets, we computed distances between all the points, and filled missing values with the mean of the three non missing closest neighbors values.
We considered missing values in categorical as a category itself.

Scaling:

We scaled the data for the training and test set to allow comparisons and smooth the scales.

Dealing with categorical variables:

We generated dummy variables for both the training and test sets, and filtered obtained features by selecting only the ones shared by both our training and test sets.

We divided our training set into X_train and y_train ,where y_train contains the objective variable and X_train all other features. We also added a column of 1s to X_train to account for the constant term in the linear regression. We split the test set in the same way.

To build our linear model, we performed a grid search cv with three types of linear regressions: Ridge, Lasso and Elastic Net.
Our best CV score is : 0.9389 using Ridge regression and alpha = 31.6

Our R^2 test score with that model : 0.848

To account for the iid assumption in the assignment we shuffled the training data. We implement our best linear model (Ridge linear regression) on the resulting data and achieve a test score of: 0.903.

Task 2: Feature Engineering

To keep the computation time low we generate our polynomial features only on numerical data. We split the data set into categorical and numerical and work with the later to perform polynomial transformation. Based on this analysis our model performance improved to CV score of 0.97

Task 3: Non-linear Models

We used gradient boosting to improve the model. To run it we took our unscaled imputed data from the preprocessing step. Since GradientBoosting uses many decision trees in the computations there was no need to scale the data. As a result, GradientBoosting significantly improved our model yielding a CV of: 0.962

Thus, we decided to see if we can improve even further by using polynomial features in Gradient Boosting. The resulting CV was: 0.964

Using the iid assumption from Task 1 and Gradient Boosting, we generate the test score of 0.982 Without iid assumption test score 0.987. Which means that without iid the test score is much better.

Task 4: Feature Selection

Based on previous tasks our best model is gradient boosting using polynomial features since it has the highest test score. Using the get_score method on Gradient boosting we were able to extract the most important features by weight. We then fit the Gradient Boosting again only with the relevant features to achieve a test score of 0.987 (which is the highest score so far).

“final_model.booster().get_score(importance_type='weight')”

We then collected only the features that has weight > 5. By running Gradient Boosting only on these features we achieved a better test score of 0.9871. Further increase in the value of weight leads to decrease in test score. Which means that we found our ‘sweet spot’.

*Variables that were left for analysis after deleting all the extra columns:

```
categorical = ['Mfr Name',
            'Division',
            'Carline',
            'Carline Class',
            'Verify Mfr Cd',
            'Air Aspir Method',
            'Index (Model Type Index)',
            'Transmission',
            'Trans',
            'Drive Sys',
            'Fuel Usage  - Conventional Fuel',
            'Fuel Unit - Conventional Fuel',
            'Fuel Metering Sys Cd',
            'Car/Truck Category - Cash for Clunkers Bill.',
            'Oil Viscosity']

numerical = ['Model Year',
           'Eng Displ',
           '# Cyl',
           'Intake Valves Per Cyl',
           '# Gears',
           'Annual Fuel1 Cost - Conventional Fuel',
           'Exhaust Valves Per Cyl',
           '$ You Spend over 5 years (increased amount spent in fuel costs over 5 years - on label) ',
           'Max Ethanol % - Gasoline',
           'Release Date']

boolean = ['Lockup Torque Converter',
          'Trans Creeper Gear',
          'Cyl Deact?',
          'Var Valve Timing?',
          'Var Valve Lift?',
          'Camless Valvetrain (Y or N)',
          'Stop/Start System (Engine Management System) Code',
          'Suppressed?',
          'Police/Emerg?',
          'Label Recalc?',
          'Unique Label?']

predict = ["Comb Unrd Adj FE - Conventional Fuel"]
```