HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

INTRODUCTION TO ARTIFICIAL INTELLIGENCE

MINI PROJECT: OLD VEHICLES PRICES PREDICTION

Instructors: Nguyễn Nhật Quang, PhD.

Students: Trần Thanh Trường - 20214938
Hoàng Đình Dũng - 20214882
Phan Công Anh - 20210078

Hanoi - December, 2022

Abstract

This report describes how we develop a prediction model that use given data of old cars sold. With this model, we use several classic methods such that Linear Regression (**LR**), Random Forest (**RF**), Gradient Boosting (**GBM**), XGBoost (**XGB**), LightGBM. Also, we will provide experimental results demonstrating the performance of the models and evaluation metrics.

1. INTRODUCTION

In recent years, Artificial Intelligence have shown to be effective in many fields like machine learning, computer vision, natural language processing, image processing, speech processing. In the automotive industry, many companies are developing and deploying Machine Learning Model to predict vehicle's prices, then implement strategies to increase their revenue . In this project, we are working on Old Vehicle's Price Prediction which is a very useful application. The task of predicting prices is attract customers and optimize sales. In the scope of this subject, we give some methods to train and predict model like Linear Regression, Random Forest Regressor and Gradient Boosting Regressor,...

2. BACKGROUND

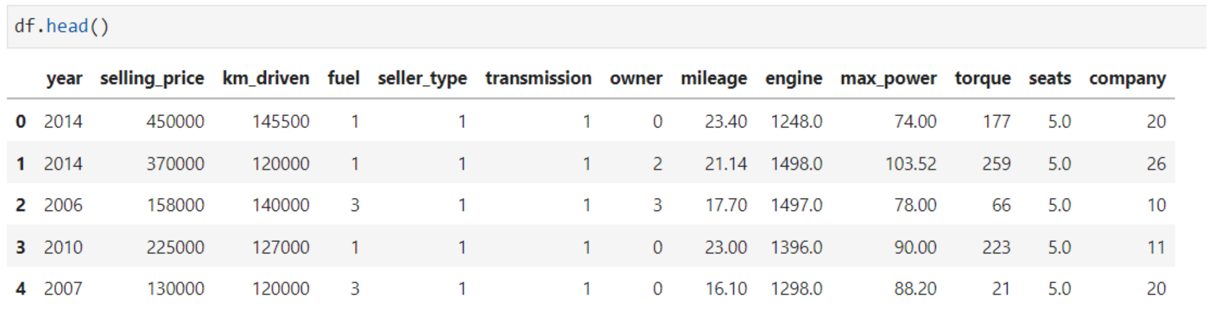
This section provides a brief explanation of the theoretical background necessary to understand our project report.

2.1. Data Preprocessing and Cleaning

We use some basic ways to clean dataset:

- Remove missing data
- Remove duplicate rows based on all columns
- Remove unnecessary columns
- Reset the index after removing missing data and duplicate rows
- Remove some unnecessary characters of several specific columns
- Encode the categorical data

After preprocessing and cleaning data, we get some information about dataset:



```
df.head()
```

	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats	company
0	2014	450000	145500	1	1	1	0	23.40	1248.0	74.00	177	5.0	20
1	2014	370000	120000	1	1	1	2	21.14	1498.0	103.52	259	5.0	26
2	2006	158000	140000	3	1	1	3	17.70	1497.0	78.00	66	5.0	10
3	2010	225000	127000	1	1	1	0	23.00	1396.0	90.00	223	5.0	11
4	2007	130000	120000	3	1	1	0	16.10	1298.0	88.20	21	5.0	20

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6712 entries, 0 to 6711
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   year           6712 non-null   int64  
 1   selling_price  6712 non-null   int64  
 2   km_driven      6712 non-null   int64  
 3   fuel           6712 non-null   int32  
 4   seller_type    6712 non-null   int32  
 5   transmission   6712 non-null   int32  
 6   owner          6712 non-null   int32  
 7   mileage        6712 non-null   float64
 8   engine         6712 non-null   float64
 9   max_power      6712 non-null   float64
 10  torque         6712 non-null   int32  
 11  seats          6712 non-null   float64
 12  company        6712 non-null   int32  
dtypes: float64(4), int32(6), int64(3)
memory usage: 524.5 KB
```

To train model conveniently, we replaced "name" column by "company" column in which company is the first word in "name" column.

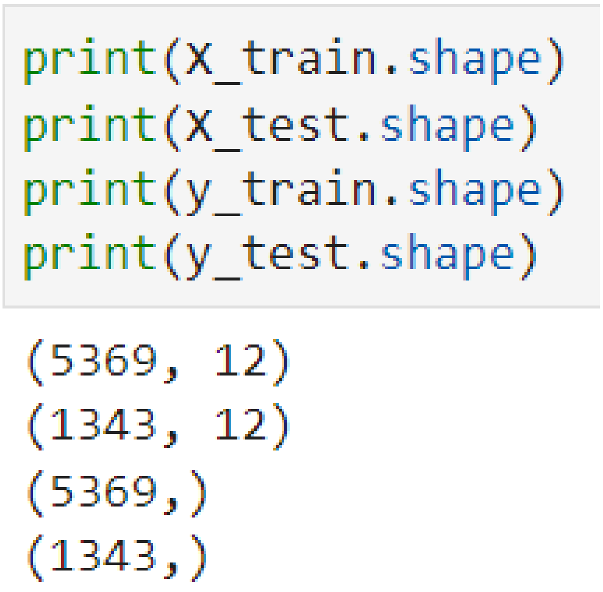
2.2. Splitting Data

Let's understand about features:

- **year**: year of manufacture
- **km_driven**: traveled distance of vehicle by kilometers
- **fuel**: type of fuel which vehicle is using
- **seller_type**: type of sellers
- **transmission**: way that vehicle moves the power from the engine to the wheels. (manual or automatic)
- **owner**: type of vehicle's owner
- **mileage**: traveled distance of vehicle by mileage
- **engine**: type of engine which vehicle is using
- **max_power**: the maximum of power of the vehicle

- **torque**: measurement of vehicle's ability to do work
- **seats**: number of seats of the vehicle
- **company**: name of the company of the vehicle
- **selling_price**: price of the vehicle and we will predict them (target)

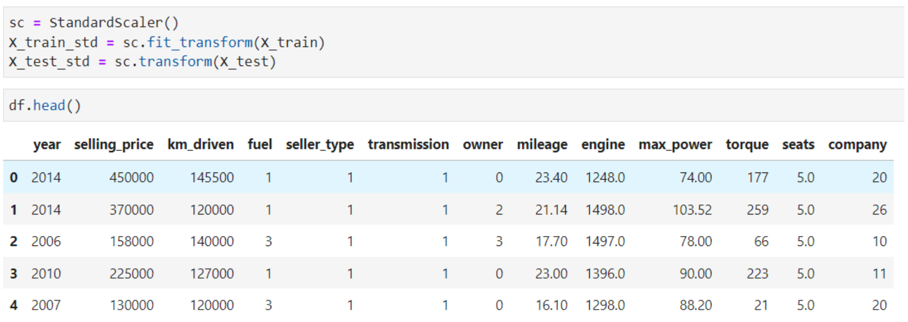
At each mentioned algorithm, we will analyze the importance of the above features that refers to the relative importance of each feature in the training data.
We split data into two sets, train set with size 0.8 and test set with size 0.2



```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(5369, 12)
(1343, 12)
(5369,)
(1343,)
```

Furthermore, we also scale features by using Standard Scaler to train model easier



```
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

```
df.head()
```

	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats	company
0	2014	450000	145500	1	1	1	0	23.40	1248.0	74.00	177	5.0	20
1	2014	370000	120000	1	1	1	2	21.14	1498.0	103.52	259	5.0	26
2	2006	158000	140000	3	1	1	3	17.70	1497.0	78.00	66	5.0	10
3	2010	225000	127000	1	1	1	0	23.00	1396.0	90.00	223	5.0	11
4	2007	130000	120000	3	1	1	0	16.10	1298.0	88.20	21	5.0	20

2.3. Linear Regression

2.3.1. What is Linear Regression

Linear regression is one of basic techniques predicting the value of target relied on other related and known data values. Model is trained based on labeled dataset, then use this to predict unknown values. Mathematically, the method creates a correlation between dependent variable and independent ones as linear equation. For example, suppose that there is a data showing your shopping expense and your salary. Last year, These two mount of money are proportional and correlation shown as first degree equation (Eg: y=a+bx).

In this report, we will implement and interpret 'Linear Regression' model based on the theory above.

2.3.2. How does Linear Regression model work?

The core problem of Linear Regression is that find a function f(x) from a training data $D = \{(x_1, y_1), (x_2, y_2), ..., \}$ in order that this function satisfies $y_i \cong f(x_i)$ for every i.

- Each observation of x is a n-dimension vector $x_i = (x_{i1}, x_{i2}, x_{i3}, ...)$ where $x_{i1}, x_{i2}, ...$ are features of an instance

The learning function is performed as a linear form:

$$f(x) = w_0 + w_1 x_1 + w_2 x_2 + ... + w_n x_n$$

[1]

- $w_0, w_1, w_2, ...$ are the regression coefficients, w_0 sometimes is called "bias"
- Observation $x = (x_1, x_2, ..., x_n)^T$
- Notes: learning a linear function is equivalent to learning the coefficient vector $w = (w_0, w_1, ..., w_n)^T$

The goal is to make prediction in the future is the best by minimizing loss function. The empirical error of the prediction is:

$$\begin{aligned} L &= \frac{1}{M}\sum_{i=1}^{M}(y_i - f(x_i))^2 \\ &= \frac{1}{M}\sum_{i=1}(y_i - w_0 - w_1 x_{i1} - w_2 x_{i2} - ... - w_n x_{in})^2 \end{aligned}$$

We find f^* so that L is minimized:

$$f^* = \arg\min_f L(f)$$
$$\iff w^* = \arg\min_w \sum_{i=1}^{M}(y_i - w_0 - w_1 x_{i1} - ... - w_n x_{in})^2$$

By taking the derivative of L and let it equals to zero, we have:

$$\frac{\partial L}{\partial w} = X^T(Xw - y) = 0$$
$$\rightarrow w^* = (X^T X)^{-1} X^T y$$

with X is data matrix of size Mx(n+1), whose the i^{th} row is $X_i = (1, x_{i1}, x_{i2}, ..., x_{in})$; $y = (y_1, y_2, ..., y_n)^T$.

Then, we produce new prediction:

$$y_x = w_0^* + w_1^* x_1 + ... + w_n^* x_n$$

One of the most popular ways to minimize loss function is use the Gradient Descent Algorithm.

We will try some values w into loss function until loss is minimize. Gradient Descent optimize this process based on gradient, the value w_{t+1} at step $t+1$ will base on gradient of loss function at previous step t:

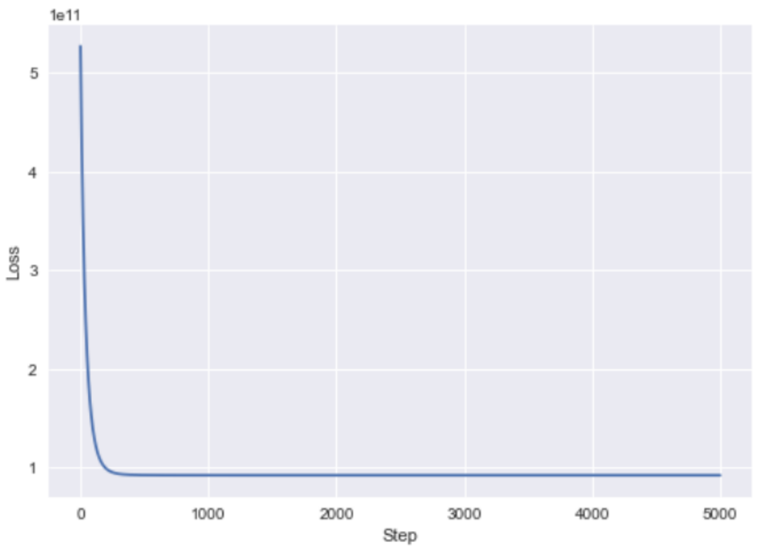
$$w_{t+1} = w_t - lr * \frac{\partial L}{\partial w}(w_t)$$
$$w_{t+1} = w_t - lr * X^T(Xw_t - y)$$

with **lr** is learning rate.

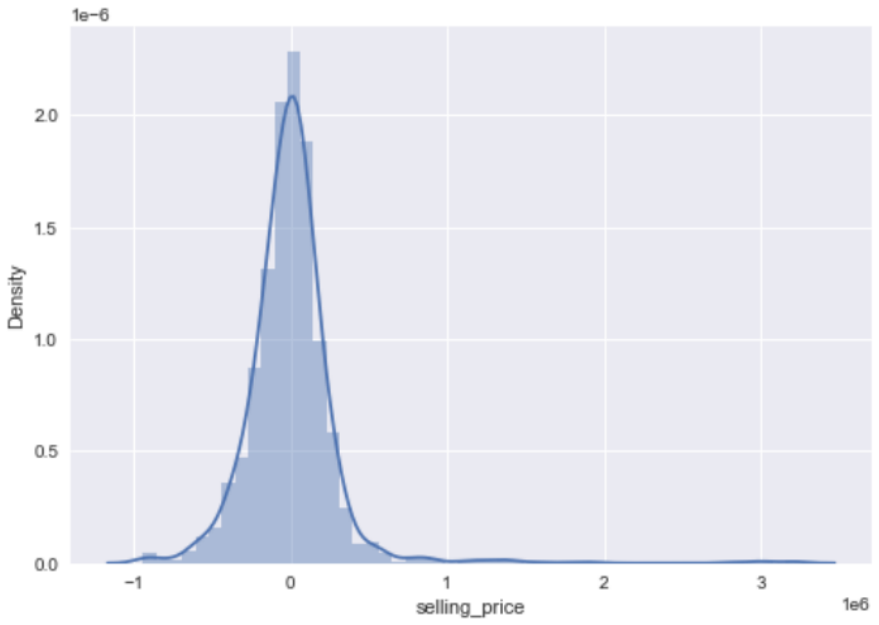
After using both Scikit-learn Library and Gradient Descend method on dataset, we get the same result that: 64% of accuracy on training set and 63.5% on test set.

```
RMSE on train set: 303581.6176214917
RMSE on test set: 316707.7767957077
Accuracy on Training: R Square: 0.6391372339546094
Accuracy on Testing: R Square: 0.6350810267524767
```

Let's check the decrease of loss function:



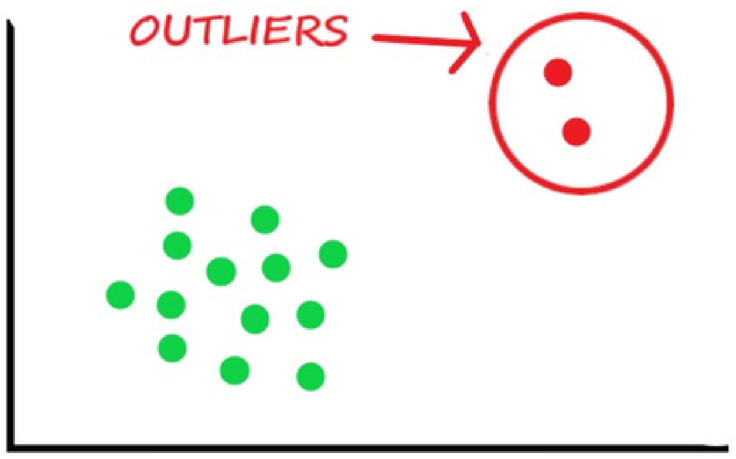
We also plot the correlation of true values and predict values:



2.3.3. Improve Linear Regression Model

To optimize the above results, we researched outlier detection in regression and apply on our project.

Outlier is known that values locate far from the expected distribution. They distort the model significantly, cause features distribution is less well-behaved, and they make the linear regression models produce worse results and more biased. There are several ways to solve this problem such that removing outliers from the observations, using algorithms, or treating them. By removing outliers from training data before modeling, we can get a better fit of the data and good performance. [2]

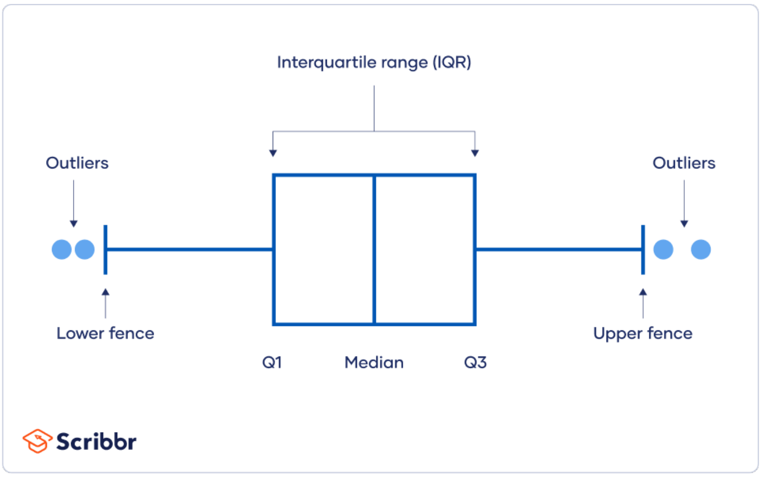


To identify outliers, we use the box plots technique which will be explained below.

Box plot is a useful technique to describe data in the middle and at the end of the distributions. It uses the median and the lower and upper quartiles (often 25th and 75th percentiles). The difference between lower and upper is called interquartile range (IQ).

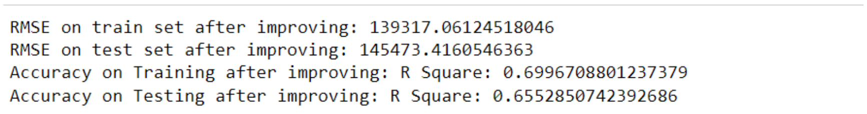
We follow some steps:

- Step 1: Calculating the median and the lower and upper quartiles.
- Step 2: Calculating the interquartile range which is the difference between the lower and upper quartile (denotes as IQ)
- Step 3: Denote Q1,Q3 are the lower and upper quartiles repectively. Calculating some points:
  - lower inner fence: $Q1 - 1.5IQ$
  - upper inner fence: $Q3 + 1.5IQ$
  - lower outer fence: $Q1 - 3IQ$
  - upper outer fence: $Q3 + 3IQ$
- Step 4: Plot a box that has the following form



- Step 5: Removing outliers. Points beyond the outer fence are called extreme outliers, and points beyond the inner fence are called mild outliers.

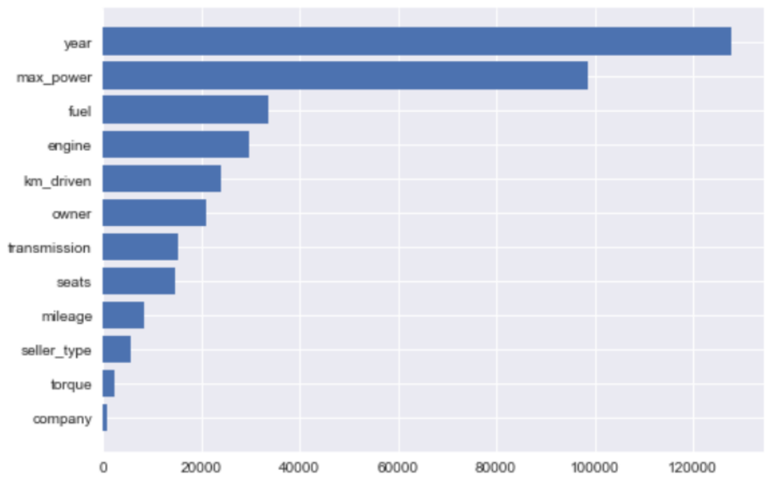
Outlier detection helps us to improve the accuracy of the linear regression model, from 64% to 69.96% for the train set and from 63.5% to 65.5% for the test set. Although it's insignificant, this explains why outliers impact on the model.


```
RMSE on train set after improving: 139317.06124518046
RMSE on test set after improving: 145473.4160546363
Accuracy on Training after improving: R Square: 0.6996708801237379
Accuracy on Testing after improving: R Square: 0.6552850742392686
```

2.3.4. Feature importance

Feature importance is a technique to evaluate the influence of input's features in the process of training model. Each observation contributes to the accuracy of the model by their features' values. Consequently, there will be many grade of importance marked by features' impact. These important points are dependent on the possible range of value that features produce.

To be more obvious, in this price-predicting model, we assume that we analyzed the data and knew each feature's standard deviation and its coefficient. Then, the importance of the feature is evaluated by its standard deviation and coefficient (importance = coefficient * standard deviation). This is exactly the impact range mentioned above. However, in our model, the whole feature values get scaled to one certain standard, so the standard deviations of each feature's new values are relatively equivalent. As a result, the coefficient is the major factor to determine whether the feature is important and how the importance is.



As we can see, **year** is the most important feature of Linear Regression model. If we remove this feature, the performance will be decrease roughly.

2.4. Boosting

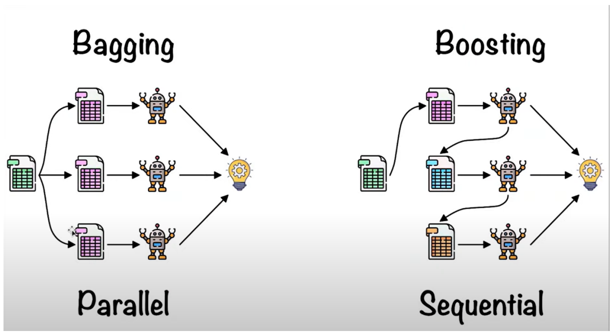
2.4.1. What is Ensemble Learning

Ensemble Learning is one of the Machine Learning methods that combines some base models to give a optimal model. Instead of making a model and deploying this model with expecting to bring the predictor accurately, Ensemble methods take a lot of models into account and compute average those models in order to produce the final results. [3]

There are three main classes in ensemble learning: bagging, boosting and stacking. In this report, we will focus on boosting method such that Gradient Boosting, XGBoost.

2.4.2. What is Boosting

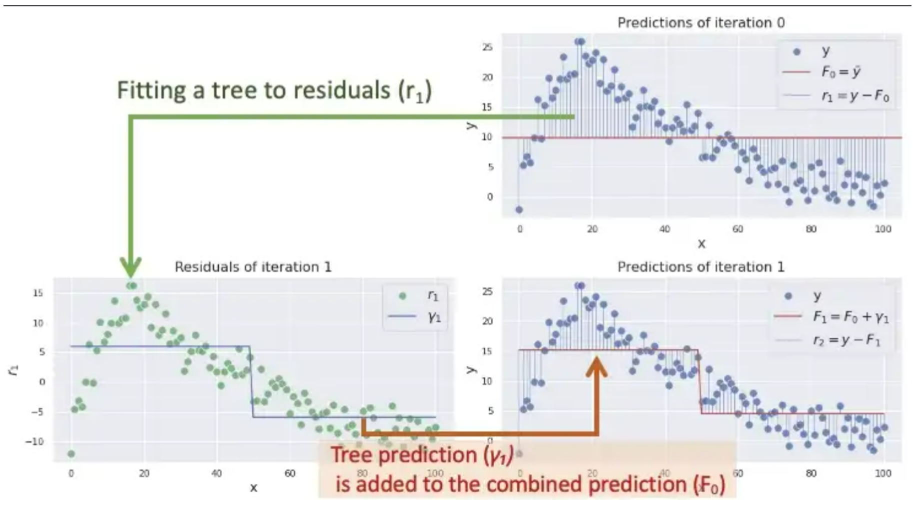
Boosting is one of the three main methods in Ensemble Learning. Different from bagging that training many decision trees parallelly on the different samples of the same dataset and produce prediction by averaging, boosting method combines a set of weak learners into strong learners to minimize training error. In boosting, a random sample of data is selected, fitted with a model and then train sequentially - that is, each model tries to compensate for the weakness of its predecessor. [4]

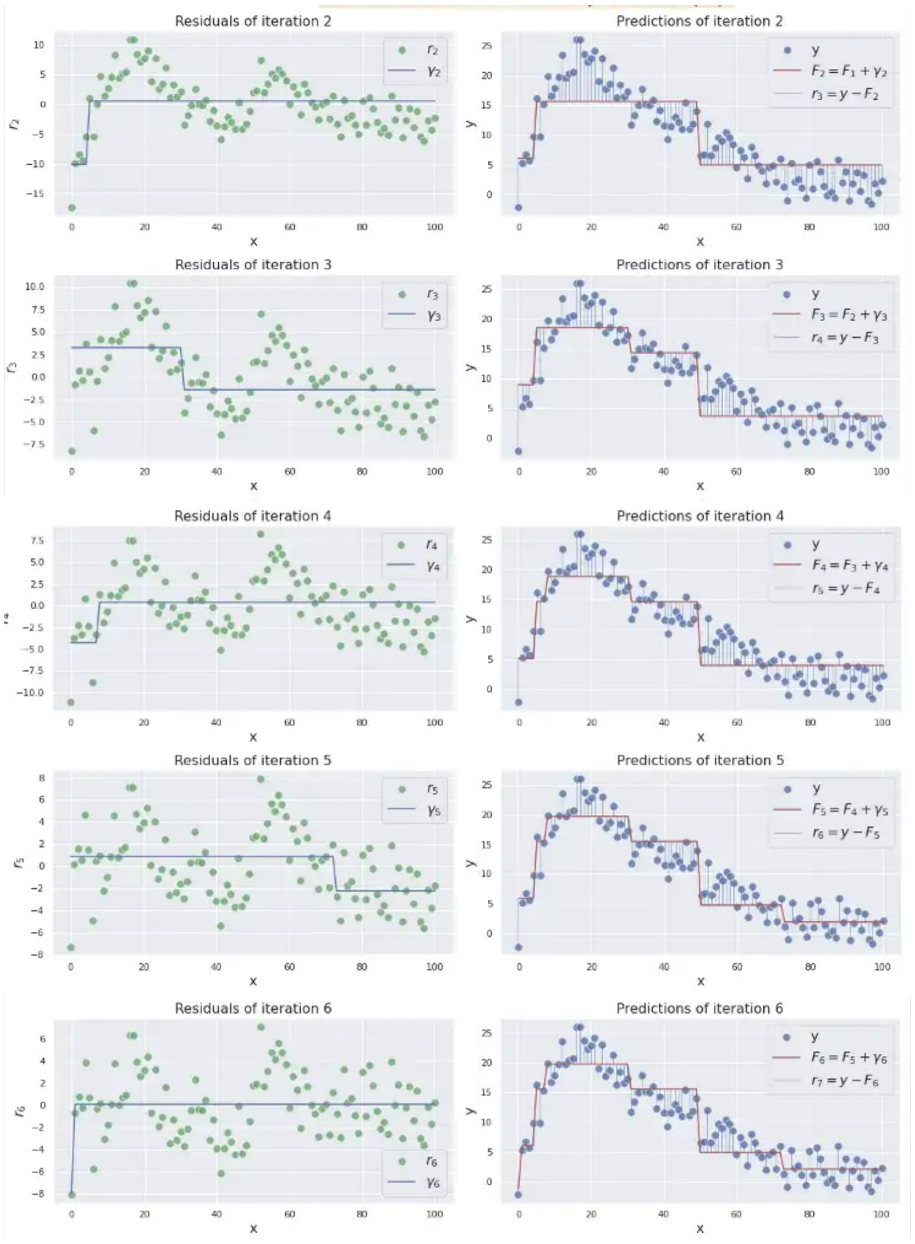


2.4.3. Gradient Boosting

One of the most popular methods in Ensemble Learning is Gradient Boosting Machine. It is a powerful technique for building predictive models for regression and classification tasks. In our project, we use Gradient Boosting Regressor to train model and give predictions.

In this project, we built gradient boosting regression trees step by step using the vehicle's data set and produce results well. To understand how it works intuitively, you could follow the image and some steps that we explain below.





We will explain Gradient Boosting Algorithm step by step:

- **Step 1**: Initialize model with a constant value

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

with L is the loss function: $L = (y_i - \gamma)^2$, this is squared error in regression case. We find γ value in order to minimize the loss function $\sum L(y_i, \gamma)$ by arg min function. Taking derivative $\sum L$ with respect to γ and let it equals to zero, we get:

$$\gamma = \frac{1}{n} \sum_{i=1}^{n} y_i = \bar{y} \rightarrow F_0(x) = \bar{y}$$

- **Step 2**: We will iterate this step M times

- Step 2.1: Compute the residual by taking derivative loss function with respect to previous prediction $F_{m-1}(x)$ and multiplying by -1. We have:

$$\begin{aligned} r_i m &= -[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}]_{F(x)=F_{m-1}(x)}, i = 1, ..., n \\ &= -\frac{\partial (y_i - F_{m-1})^2}{\partial F_{m-1}} \\ &= 2(y_i - F_{m-1}) \end{aligned}$$

  We can pop 2 out of it since it is a constant, so the residuals is $r_i m = y_i - F_{m-1}$
- Step 2.2: With features x, we train regression tree, then create terminal node R_{jm} for $j = 1, 2, ..., J_m$ including:
  * m: tree index
  * j: terminal node
  * J: total number of leaves
- Step 2.3: In this step, we go to find γ_{jm} so that loss function is minimize on each terminal node:

$$\begin{aligned} \gamma_{jm} &= \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma) \\ &= \arg\min_{\gamma} \sum_{x_i \in R_{jm}} (y_i - F_{m-1}(x_i) - \gamma)^2 \end{aligned}$$

  for $j = 1, 2, ..., J_m$.
  Similar to step 1, taking derivative with respect to γ and let it equal to zero, we have:

$$\frac{\partial}{\partial \gamma} \sum_{x_i \in R_{jm}} (y_i - F_{m-1}(x_i) - \gamma)^2 = 0$$

$$-2 \sum_{x_i \in R_{jm}} (y_i - F_{m-1}(x_i) - \gamma) = 0$$

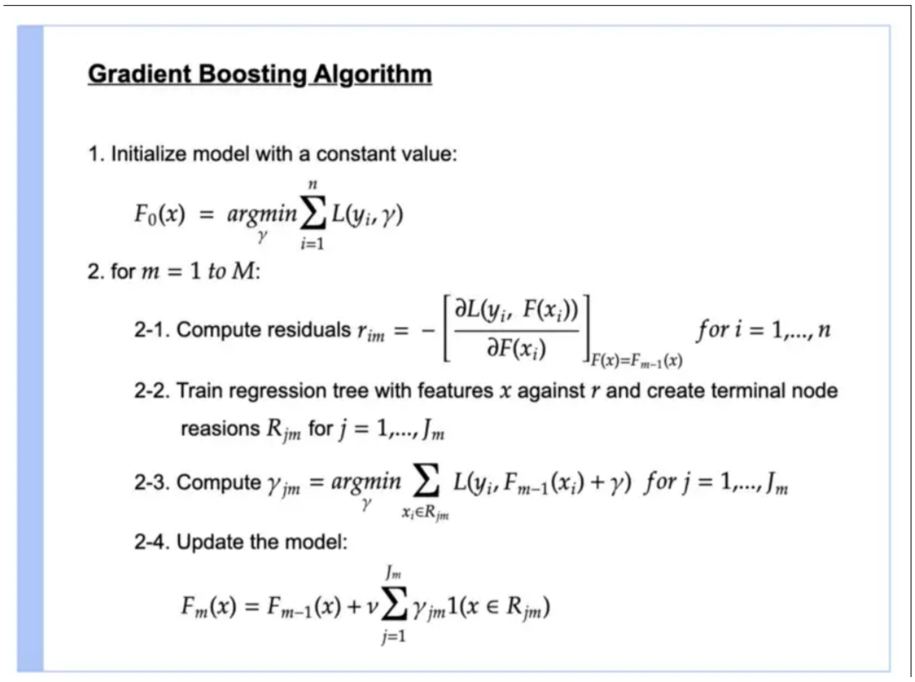
$$n_j \gamma = \sum_{x_i \in R_{jm}} (y_i - F_{m-1}(x_i))$$

$$\gamma = \frac{1}{n_j} \sum_{x_i \in R_{jm}} r_{im}$$

  Notes: n_j denotes number of samples in the terminal node j.
  In conclude, we will calculate the average of the target value (residuals) in each terminal node and produce the regular prediction values of regression tree γ_{jm}.
- Step 2.4: We update the model: $F_m(x) = F_{m-1}(x) + v \sum_{j=1}^{J_m} \gamma_{jm}$ with $x \in R_{jm}$.
  Using $v \in (0, 1)$ is learning rate which controls the degree of contribution of the additional tree prediction γ to the combined prediction F. Also model overfitting to be decrease if we get the small learning rate.

Remember that we need to iterate step 2 M times.

The brief algorithm is shown bellow:



Gradient Boosting Algorithm

1. Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{argmin} \sum_{i=1}^{n} L(y_i, \gamma)$$

2. for $m = 1$ to M:

2-1. Compute residuals $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ $for\ i = 1,\ldots,n$

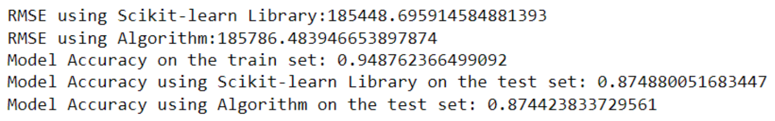
2-2. Train regression tree with features x against r and create terminal node reasions R_{jm} for $j = 1,\ldots,J_m$

2-3. Compute $\gamma_{jm} = \underset{\gamma}{argmin} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$ $for\ j = 1,\ldots,J_m$

2-4. Update the model:

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} 1(x \in R_{jm})$$

After deploying Gradient Boosting Regressor by two ways (one is using Scikit-learn Library and the other is above Algorithm) on the vehicle's dataset, we get quite good result with the accuracy of 94.88% for the train set and 87.48% for the test set (same results for 2 ways).

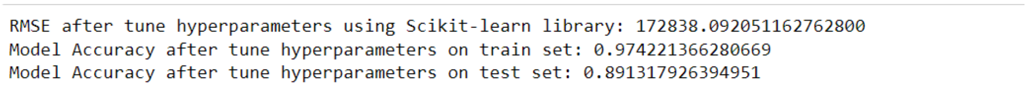


```
RMSE using Scikit-learn Library:185448.695914584881393
RMSE using Algorithm:185786.483946653897874
Model Accuracy on the train set: 0.948762366499092
Model Accuracy using Scikit-learn Library on the test set: 0.874880051683447
Model Accuracy using Algorithm on the test set: 0.874423833729561
```

There are some important parameters impact on model which we want to explain:

- n_estimators: the number of sequential trees to be modeled
- learning_rate: controls the degree of contribution of the additional tree prediction to the combined prediction. Small learning rates are popular because they make the model fit to the specific characteristics of tree, but they require higher number of trees.
- max_depth: the maximum depth of a tree. They help avoid overfitting since higher depth allow model to learn relations very specific to a particular sample.

In our code, we choose values for above parameters by intuitively. After using GridSearchCV function from Scikit-learn Library to tune hyperparameters, the result becomes better than the original which is 97.42% for train set and 89.13% for test set.



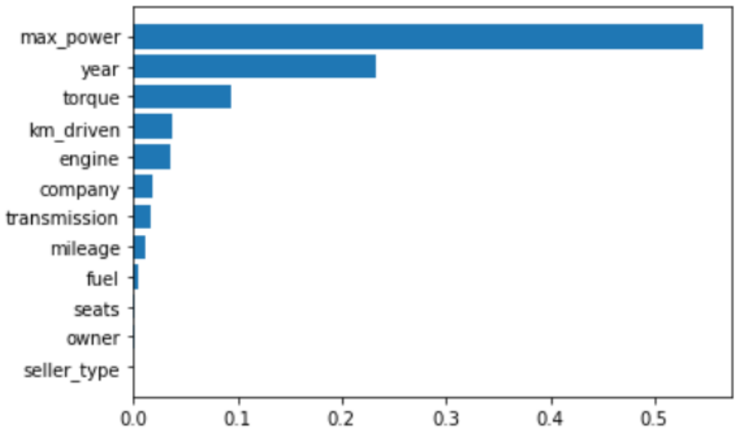
```
RMSE after tune hyperparameters using Scikit-learn library: 172838.092051162762800
Model Accuracy after tune hyperparameters on train set: 0.974221366280669
Model Accuracy after tune hyperparameters on test set: 0.891317926394951
```

2.4.4. Features Importance

Gradient Boosting model retrieve importance scores of each attribute after constructing the trees. It shows how useful each feature was in decision trees, the more an attribute is used to make key decisions with decision trees, the higher its relative importance.

Importance is computed for a decision tree by above principle, then averaged over all of the decision trees within the model. Features importance are calculated for each attribute in the dataset, allow to be compared each other. [5]

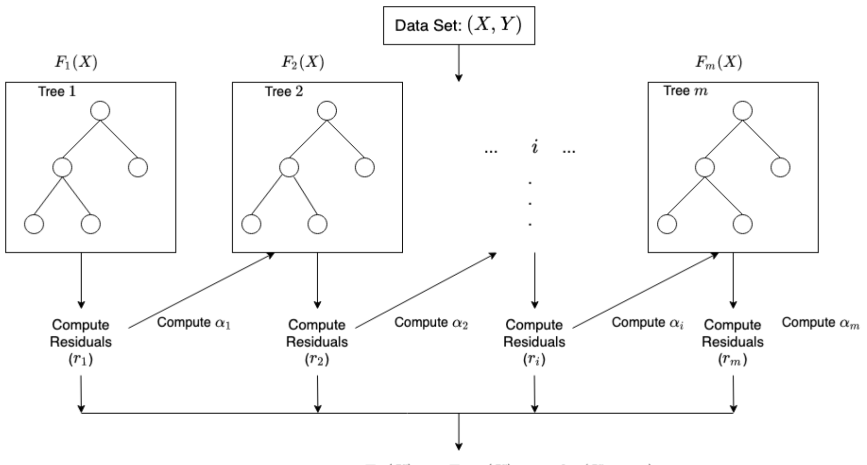
The image below shows the features importance of our GBM model in which **max_power** is the most important.



2.4.5. XGBoost

Extreme Gradient Boosting Machine (XGBM) is the latest version of gradient boosting machines that works very similar to GBM. In XGBM, trees are added sequentially (one at a time) that learn from the errors of previous trees and improve them. Although algorithm of XGBM and GBM are quite similar but still there are a few differences between them:

- XGBM reduces overfitting or underfitting, make model performance better by using regularization techniques.
- Instead of doing sequential process like GBM, XGBM follow parallel of each node. This makes speed becomes quicker than GBM.
- XGBM handling missing data automatically, so we can pop this step in data preprocessing step.



$$F_m(X) = F_{m-1}(X) + \alpha_m h_m(X, r_{m-1}),$$

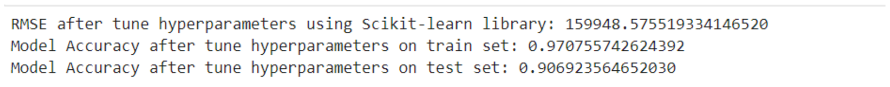
where α_i, and r_i are the regularization parameters and residuals computed with the i^{th} tree respectfully, and h_i is a function that is trained to predict residuals, r_i using X for the i^{th} tree. To compute α_i we use the residuals computed, r_i and compute the following: $arg \min_{\alpha} = \sum_{i=1}^{m} L(Y_i, F_{i-1}(X_i) + \alpha h_i(X_i, r_{i-1}))$ where $L(Y, F(X))$ is a differentiable loss function.

The initial accuracy on the train set of XGBoost is 98.89% and on the test set is 86.78% (with default parameters).

We try to tune some hyperparameters that we understand basically:

- n_estimators: the number of trees to be modeled, also the number of boosting rounds.
- learning_rate: step size at each iteration, is added to optimize the combined prediction, optimizes chance for getting optimum value. This value is between 0 and 1.
- max_depth: the maximum depth of tree. A high value for max_depth might increase the performance but also the chances of overfitting.
- colsample_bytree: the value is between 0 and 1. It represents the fraction of columns to be randomly sampled for each tree.

After using GridSearchCV to try to tune hyperparameters, we get the accuracy of the test set is 90.69%



```
RMSE after tune hyperparameters using Scikit-learn library: 159948.575519334146520
Model Accuracy after tune hyperparameters on train set: 0.970755742624392
Model Accuracy after tune hyperparameters on test set: 0.906923564652030
```

Because of lack of experience in hyperparameters tuning, we only try to tune them basically. Hope that you will give more feedback and experience about this topic.

2.5. Random Forest

2.5.1. Understanding basic Decision Tree

Decision Tree is one of the most powerful method in machine learning. It has a tree, hierarchical structure that includes root node, branches, internal nodes and leaf nodes.

The outcomes branch from root node and go into terminal node, by evaluating technique, all possible outcomes are represented in leaf nodes. To seek the tree which is the best in split the set of data, the data will be typically trained by Cart algorithm applying some evaluation metric such as Gini impurity.

Decision Tree using greedy algorithm to find the optimal result within a tree. One of the ways that decision tree get stable accuracy is forming an ensemble via random forest algorithm.

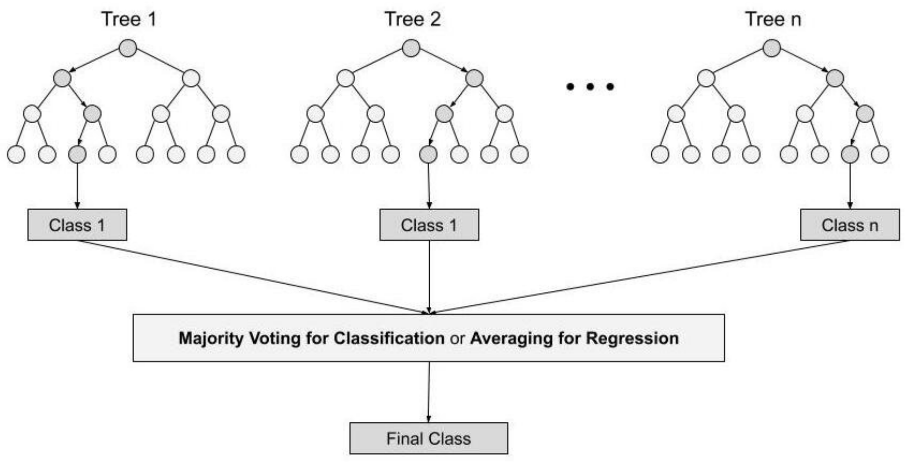
2.5.2. What is Random Forest

As mentioned above, Ensemble Learning includes Bagging method. And one of the most popular algorithm in machine learning is Random Forest which is a bagging technique, based on many Decision Trees that built on different samples and generalize all votes from those to find the majority for classification or average values in case of regression.

2.5.3. Random Forest Algorithm

This algorithm includes several steps:

- **Step 1**: Randomly choose m features from n given feature of the dataset input (m<n), the default value will be $\sqrt{n}$ if there is no reference
- **Step 2**: In each feature, we choose the best split point of this feature at node d
- **Step 3**: Continue splitting the node into several of nodes using best-split method
- **Step 4**: Repeat the above process to build completely individual Decision Tree
- **Step 5**: From n built Decision Tree, we form a Random Forest model


Tree 1
Tree 2
Tree n
Class 1
Class 1
Class n
Majority Voting for Classification or Averaging for Regression
Final Class

After applying Random Forest method, we get the accuracy of 98.60% for the train set and 88.05% for the test set.

```
RMSE on the train set: 181202.48326613262
RMSE on the test set: 181202.48326613262
Accuracy on the test set: R Square: 0.986073492313117
Accuracy on the test set: R Square: 0.8805441895369438
```

2.5.4. How to improve Random Forest model?

As mentioned above, Random Forest model is constructed based on small components. Therefore, in the process of train model, it has a lot of parameter that need to be considered carefully like (number of trees to divided, maximum depth of each tree,...).

Each parameter has a considerable effect on the predicting quality of the model. Caught the problem, a sotisphicated solution was come up with, that is *Hyperparameter tuning*.

2.5.5. Hyperparameter tuning

Hyperparameter tuning is primarly finding the set of parameters' values such that this contributes to the performance of the model, maximizes the efficient but still ensures the complexity within a dataset. This process is used to not only improve the predictive power of the model but also improve the speed, make the model runs faster.

- n_estimators: Number of trees set to be built by the algorithm. Generally, higher number of trees divied, higher the accuracy in predicting. Try to rise the number of trees in the forest will decrease the variance of the model, and does not make model get into overfitting. However, there are diminishing results, and continuously increasing trees leads to the worse and worse computational volume. Therefore, it is reasonable to get a large number of trees but the computational burden is still in suitable limits. Several hundred is typically a good idea to try, but it is largely relied on the model's situation

- tree size (maximum_depth, min_sample_split, min_sample_leaf): Determines the size of tree in detail.

  Larger tree results in heavy computation, complicated function. As a consequence, the model easily becomes overfitting. By default, there is not any restriction. Therefore, Tuning tree size can enhance the accuracy through balancing between overfitting and underfitting.

- max_features: Maximum number of features random forest are limited to be considered in a tree.

  Whenever the node starts splitting, a set of feature is considered to find randomly, and the best one is choose to perform the split. With more feature, we can do better performance at split job. But, it also increases the correlation between trees, the variance of model is also affected to increase. It is assumed that the default is suitable and balanced for any model (square root of total with classification, and 1/3 of total with regression). As tree size, it needs tuning to ensure the model's performance.

Choosing hyperparameters:
It's probably the hardest problem in tuning hyperparameter. The reason is that we cannot choose the most important one to tune and the hyperparameters have interaction and affect each other within the process. Therefore, the evaluation at one time doesn't work since when changing to another hyperparameter, the evalutaion will be completely different. In this model, we just choose some typical hyperparameter (the

ones mentioned above) for tuning test.

Hyperparameters tuning process:
Tuning random forest hyperparameters is as other typical models that need tuning. The first step is defining the space search (domain) of each hyperparameter with the form of set (include lists). Each list of parameters' values is trained and then estimated to determine its best-performance value. Finally, collect all optimal value and train the new model.

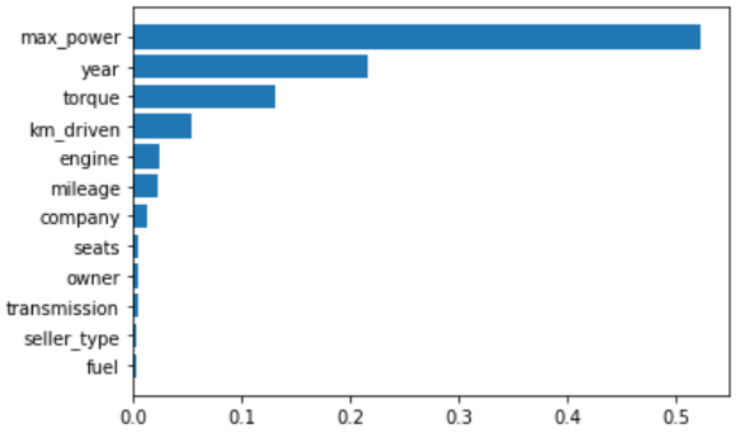
If we have prior experience with a model, we might know where the best values for the hyperparameters typically lie, or what a good search space is.In other way, we simply define a large space by random searching and hope that the best values are in there. Typically, when first using a method, I define a wide search space centered around the default values. Then, if I see that some values of hyperparameters tend to work better, I can concentrate the search around those values. [6]

Trying to tune hyperparameters, we get the accuracy of 88.41% for the test set, slightly increasing.

```
RMSE on the train set after tuning: 178483.81548974745
RMSE on the test set after tuning: 178483.81548974745
Accuracy on the test set after tuning: R Square: 0.9785227660921533
Accuracy on the test set after tuning: R Square: 0.8841018051589403
```

2.5.6. Features Importance

The important feature of Random Forest method is how much this feature is used in each tree of the forest. It equals to the total of the impurity decrease from all decision trees in forest. Let's look at the features importance of our Random Forest model:



max_power is still the most important feature, it impacts strongly on RF model. We can see features that have low cardinality are not as important as higher cardinality features, for example year feature is more crucial than the transmission feature.

2.6. Evaluation and Prediction

After deploying and improving model by several methods, we draw some useful experience and some conclusions. This is shown in the table below.

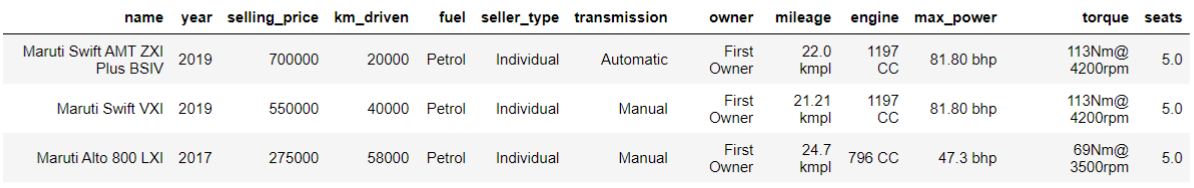
METHODS	Advantages	Disadvantages
Linear Regression	- Has an uniquely efficient performance for linear-separated data - The model implementation is simple, so it is easy to approach and able to interpret in detail - Handles overfitting pretty well by dimensionally reduction	- Too sensitive to outliers, so it easily leads to overfitting - Has weakly performance when deals with nonlinear-structured data
Gradient-Boosting and XGBoost	- Has a high accuracy compared to other models in general - Possible to deal with a extremely large and complicated dataset - Usually provides support in categocial features handling - XGBoost have the speed which is better than GBM and highly accurate implementation of GBM	- As other models with large computation volume, this model costs much of time to train and needs a large resource - Because of its complex in structure, the final model is hard to interpret - XGBoost is very sensitive to outliers as other classifiers are supposed to be constructed with base of predecessor learners
Random Forest	- Based on *bagging* and *ensemble learning*, the variance is minimized in decision trees and therefore the overfitting problem and accuracy are improved - Handles smoothly with both categorical and continuous variables - Opposite to Linear Regression, Random Forest is quite robust to outliers, it can treat outliers and use this to contribute the model automatically - Random Forest algorithm is very stable	- Enforces a giant sequence of computation and leaves large resource burden - Random forest requires huge amount of time to train as it produces trees with a lot of computations inside

The accuracy of the model with different methods also has deviance:

METHODS	Train Set	Test Set
Linear Regression	69.96%	65.53%
Gradient Boosting	97.42%	89.13%
XGBoost	97.07%	90.69%
Random Forest	97.85%	88.41%

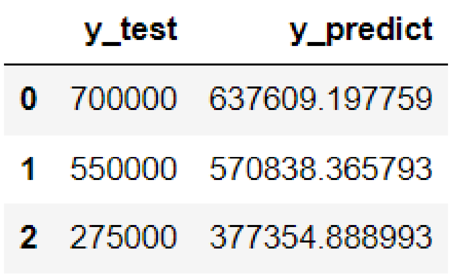
We can see that XGBoost is a powerful method to train machine learning models. That is the reason that XGBoost wins many competitions.

Now, we will try to predict some instances in the test set with the following features:



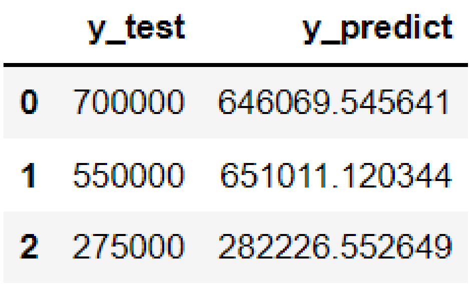
name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats
Maruti Swift AMT ZXI Plus BSIV	2019	700000	20000	Petrol	Individual	Automatic	First Owner	22.0 kmpl	1197 CC	81.80 bhp	113Nm@ 4200rpm	5.0
Maruti Swift VXI	2019	550000	40000	Petrol	Individual	Manual	First Owner	21.21 kmpl	1197 CC	81.80 bhp	113Nm@ 4200rpm	5.0
Maruti Alto 800 LXI	2017	275000	58000	Petrol	Individual	Manual	First Owner	24.7 kmpl	796 CC	47.3 bhp	69Nm@ 3500rpm	5.0

We get:



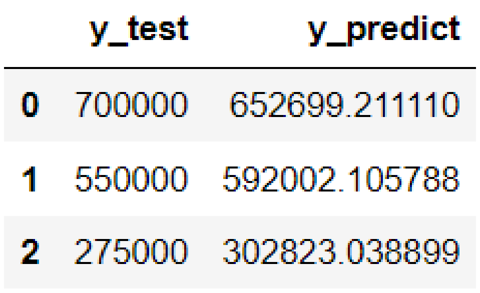
	y_test	y_predict
0	700000	637609.197759
1	550000	570838.365793
2	275000	377354.888993

(a) Linear Regression



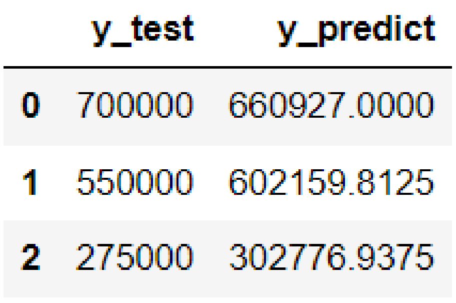
	y_test	y_predict
0	700000	646069.545641
1	550000	651011.120344
2	275000	282226.552649

(b) Random Forest



	y_test	y_predict
0	700000	652699.211110
1	550000	592002.105788
2	275000	302823.038899

(c) Gradient Boosting



	y_test	y_predict
0	700000	660927.0000
1	550000	602159.8125
2	275000	302776.9375

(d) XGBoost

Figure 1: Prediction

3. CONCLUSION

Price predicting (In particular here is vehicle) is gradually becoming indispensable, especilaly in a technology-based market with full of competition. In this report, we clarify core issues of a predicting model: what it is?, how it works?, and other kind of basic definition. Important models are also implemented and interpreted in detail, since then both general perspective and insight in each predicting model are brought out obviously.

For further information about ours works and submission, please visit our Github repository TruongTran/Project-Intro-AI

REFERENCES

[1] Weisberg, Sanford. Applied linear regression. Vol. 528. John Wiley Sons, 2005.

[2] Ben-Gal, Irad. "Outlier detection." Data mining and knowledge discovery handbook. Springer, Boston, MA, 2005. 131-146.

[3] Polikar, Robi. "Ensemble learning." Ensemble machine learning. Springer, Boston, MA, 2012. 1-34.

[4] Schapire, Robert E. "A brief introduction to boosting." Ijcai. Vol. 99. 1999.

[5] Section 10.13.1 “Relative Importance of Predictor Variables” of the book The Elements of Statistical Learning: Data Mining, Inference, and Prediction, page 367.

[6] Probst, Philipp, Marvin N. Wright, and Anne-Laure Boulesteix. "Hyperparameters and tuning strategies for random forest." Wiley Interdisciplinary Reviews: data mining and knowledge discovery 9.3 (2019): e1301.