Driving declinometer and vehicle weight estimation CS116.O11.KHCL

Nguyen Thanh Lam
University of Information Technology (UIT)
HCM City, VietNam
20521517@gm.uit.edu.vn

I. Introduction

Knowing the weight of a vehicle and the slope of the road would be beneficial for many driver assistance systems. For example, the information could be used to improve power management or plan routes based on weight restrictions.

Unfortunately, installing additional sensors to measure weight and slope directly is expensive. However, vehicles already track properties like engine power and speed that depend strongly on unknown quantities. These could be used as input for a software "sensor" that is much cheaper than a hardware solution

So in this article I will present the analysis of a dataset of signals from the vehicle as well as the prediction of weights and gradients using the amount of signals already present on a given vehicle on the dataset. From there, solutions are proposed to help the driver assistance system be easier to choose and more convenient for users.

Step by step I will take to process the dataset from raw data to successful prediction.

- Data preprocessing.
- Decide to drop the column features or not.
- Propose algorithm to solve two main problems, Linear regression and classification.
- Propose metrics to evaluate algorithms.

II. Data preprocessing

A. Preliminary data survey

In this section, I will check the data size, whether the data has a null value or not, and the possible data types.

I used Pandas library to read the input data file, Fig 1 is the result after reading:

```
1 df.head()
```

	Epm_nEng_100ms	VehV_v_100ms	ActMod_trqInr_100ms	RngMod_trqCrSmin_100ms	CoVeh_trqAcs_100ms	Clth_st_100ms	CoEng_st_100ms
0	902.5	67.72	1971.9360	-140.0	9.999747	0	3
1	1241.0	63.87	2604.0000	-196.0	9.999747	0	3
2	903.0	67.28	2208.0700	-140.0	9.999747	0	3
3	934.5	68.34	0.0000	-140.0	9.999747	0	3
4	969.0	61.28	392.5794	-112.0	9.999747	0	3

Fig. 1. Original dataset

Using the **df.shape()** statement, I get the result as (8496, 11), which means there are 8496 rows of data with 9 feature columns and the remaining two columns are two label columns (one predicts linear regression, The remaining column predicts classification).

Next, I use the statement **df.isna().sum()** to check if the data has missing values or not. The result is as shown in Fig 2, there are no missing values on data.

```
1 df.isna().sum()
```

```
Epm_nEng_100ms                0
VehV_v_100ms                  0
ActMod_trqInr_100ms           0
RngMod_trqCrSmin_100ms        0
CoVeh_trqAcs_100ms            0
Clth_st_100ms                 0
CoEng_st_100ms                0
Com_rTSC1VRVCURtdrTq_100ms    0
Com_rTSC1VRRDTrqReq_100ms     0
RoadSlope_100ms               0
Vehicle_Mass                  0
dtype: int64
```

Fig. 2. Check null values

To avoid confusion while processing the data columns, I will separate the label columns (these two columns are not related or change during the data processing process).

Where variable **X** contains the data that needs preprocessing, **y_linear** is the label for the linear regression problem and variable **y_classification** is the label for the classification problem, Fig 3 illustrates data separation.

```
1 X = df.iloc[:,0:9]
2 y_linear = df.iloc[:,9]
3 y_classification = df.iloc[:,10]
```

Fig. 3. Separating label columns

In the next step, I will determine the type of each feature column, categorical, or numerical data type. Fig 4 and Fig 5 illustrates the mentioned process. It can be seen that all

columns in the dataset are numeric, and no columns are categories.

```
# find categorical variables
categorical = [var for var in X.columns if X[var].dtype=='O']

print('There are {} categorical variables\n'.format(len(categorical)))
print('The categorical variables are :', categorical)
```

```
There are 0 categorical variables

The categorical variables are : []
```

Fig. 4. Checking categorical data type

```
# find numerical variables
numerical = [var for var in X.columns if X[var].dtype!='O']

print('There are {} numerical variables\n'.format(len(numerical)))
print('The numerical variables are :', numerical)
```

```
There are 9 numerical variables

The numerical variables are : ['Epm_nEng_100ms', 'VehV_v_100ms', 'ActMod_trqInr_100ms',
```

Fig. 5. Checking numerical data type

B. Several graphs depict the trend of the data

1) Outliers in numerical variables: To find outliers in the data set, I will use box plot to check the central tendency of data points and which points are outside the range of the box. Fig 6 and Fig 7 show are the results after drawing graph.

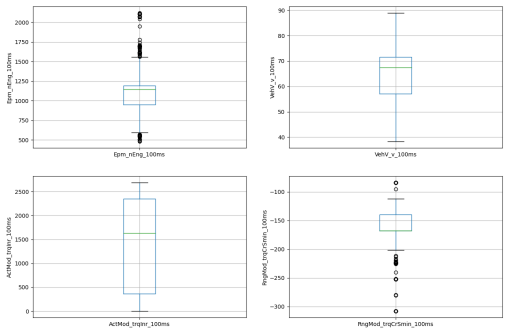

Fig. 6. Box plot for Epm_nEng_100ms, VehV_v_100ms, ActMod_trqInr_100ms and RngMod_trqCrSmin_100ms columns

It can be seen in Fig 6 that in the Epm_nEng_100ms and RngMod_trqCrSmin_100ms columns, there are a lot of outliers in these variables.

2) Check the distribution of variables: Now, I will plot the histograms to check distributions to find out if they are normal or skewed. If the variable follows normal distribution, then I will do Extreme Value Analysis otherwise if they are skewed, I will find IQR (Interquantile range).

We can see that all the *Epm_nEng_100ms*and *RngMod_trqCrSmin_100ms* variables are skewed. So, I will use interquartile range to find outliers.

For *Epm_nEng_100ms*, the minimum and maximum values are 482.0 and 2120.0. So, the outliers are values > 1924.875 or

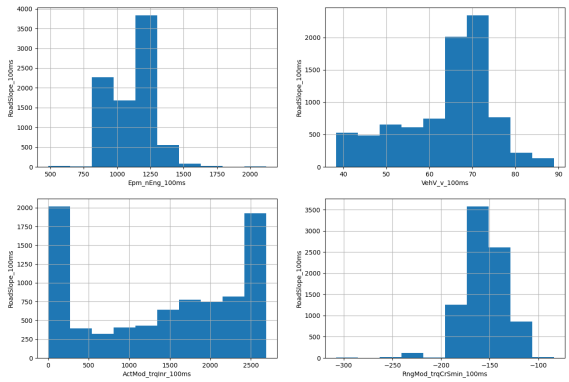

Fig. 7. Hist plot for Epm_nEng_100ms, VehV_v_100ms, ActMod_trqInr_100ms and RngMod_trqCrSmin_100ms columns

< 216.0 , and the same column's *RngMod_trqCrSmin_100ms* are values < -252.0 or > -56.0

In the remaining 5 columns are CoVeh_trqAcs_100ms, Clth_st_100ms, CoEng_st_100ms, Com_rTSC1VRVCURtdrTq_100ms and Com_rTSC1VRRDTrqReq_100ms respectively. The values in them mostly do not change or fluctuate too much, Fig 8 illustrate that. So I decided to remove the above 5 columns.

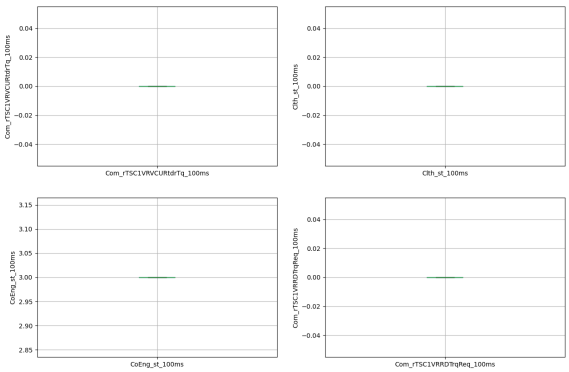

Fig. 8. Box plot for four non-fluctuation features

III. Training and Evaluating Data

In this section, I will do the following steps

- Split train and test data.
- Standardized train and test data.
- Training linear, classifier models and evaluate model performances.

A. Splitting training and testing data

I split the data set into two sets for training and testing with ratios of 0.8 and 0.2 respectively. Fig 9 describe a training set after spliting.

	Epm_nEng_100ms	VehV_v_100ms	ActMod_trqInr_100ms	RngMod_trqCrSmin_100ms
count	6796.000000	6796.000000	6796.000000	6796.000000
mean	1102.443754	64.438992	1408.981414	-158.968887
std	155.170728	10.642140	988.613374	26.298610
min	481.500000	38.310000	0.000000	-308.000000
25%	944.000000	57.337500	339.500000	-168.000000
50%	1144.750000	67.395000	1602.184000	-168.000000
75%	1191.000000	71.580000	2327.500000	-140.000000
max	1924.875000	88.930000	2688.000000	-84.000000

Fig. 9. Original training data

B. Standardized train and test data

I use MinmaxScaler, StandardScaler and RobustScaler to normalize the data train and test.
Formula for normalization for each scalers:

1) MinmaxScaler:

$$\text{MinMaxScaler}(x) = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Where x is the variable before normalize.

2) StandardScaler:

$$\text{StandardScaler}(x) = \frac{x - \text{mean}(x)}{\text{std}(x)}$$

Where mean and std are the expectation and standard deviation of that component on the entire training data, respectively.

3) RobustScaler:

$$\text{RobustScaler}(x) = \frac{x - \text{Q}_1(x)}{\text{Q}_3(x) - \text{Q}_1(x)}$$

Where x is the value of the variable that needs to be normalized, $\text{Q}_1(x)$ is is the first percentile (25%) of x, $\text{Q}_3(x)$ is the third percentile (75%) of x.
Fig 10 illutrates training set after using MinmaxScaler.

	Epm_nEng_100ms	VehV_v_100ms	ActMod_trqInr_100ms	RngMod_trqCrSmin_100ms
6260	0.466615	0.654484	0.813802	0.625
8472	0.469732	0.064006	0.979167	0.875
2067	0.467308	0.654089	0.824219	0.625
5632	0.402875	0.783682	0.000000	0.750
1047	0.471811	0.661004	0.449219	0.625
...	...	...	...	...
5734	0.284749	0.553536	0.199219	0.625
5191	0.387980	0.316871	0.763348	0.750
5390	0.281632	0.554326	0.042100	0.625
860	0.278860	0.555314	0.885417	0.750
7270	0.306573	0.601541	0.539107	0.750

Fig. 10. Training data after using MinmaxScaler

TABLE I
MAE VALUES OF 3 MODELS IN 3 SCALERS

	MinMaxScaler	StandardScaler	RobustScaler
LinearRegression	-0.42	-0.42	-0.424
Lasso	-0.44	-0.42	-0.428
DecisionTreeRegressor	0.505	0.09	0.1525

C. Metrics

a) Linear metric: This measure averages the absolute difference between the predicted value and the actual value across all samples in the data set. MAE provides an average measure of how much a prediction deviates from the actual value and is widely used in evaluating the performance of prediction models, especially in regression problems.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

The value returned by the evaluation function is in the range [-0.5, 1].

b) Classification metric: Recall is a measure used in classification problems to evaluate a model's ability to detect all positive cases. This measure measures the ratio between the number of positive cases correctly predicted by the model (true positives - TP) and the total number of actual positive cases (TP + false negatives - FN) learned from two values.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

After calculating the two recall values of labels 38 and 49, the evaluation weight will be calculated by taking the square root of the product of two recalls.

$$W = \sqrt{\text{Recall}_{38} \times \text{Recall}_{49}}$$

D. Training linear and classifier models

1) Linear models: I use the following machine learning models to solve the problem:

- DecisionTreeRegressor
- Lasso
- LinearRegression

DecisionTreeRegressor reached the highest value when tested with all three scalers, the values were 0.505, 0.09 and 0.1525 respectively. The two models LinearRegression and Lasso appear to be quite weak when encountering highly challenging data sets, the scores of both models are near the bottom (about -0.4). Table I is the result after evaluate the absolute error in three models (using testing data).

2) Classifier models: List models I used to train and evaluate data:

- Logistic Regression
- DecisionTreeClassifier

In general, both models have very reasonable accuracy when predicting, the model's lowest score when predicting on RobustScaler is about 0.94 and the highest when using

TABLE II
EXPERIMENT USE TWO MODELS WITH THREE SCALERS

	MinMaxScaler	StandardScaler	RobustScaler
LogisticRegression	0.996	0.94	0.994
DecisionTreeRegressor	0.996	0.945	0.994

MinmaxScaler (0.96). Table II is the results after the experiment.

E. Conclusion

In this project, I conducted processes such as data preprocessing, checking for outlier data, data division steps to prepare for testing and finally training and evaluating the model. Although the classifier models bring high results, in the opposite direction, the linear regression models do not bring the expected efficiency (the highest is 0.505 on the [-0.5,1] scale), in the future I will find New solutions that can enhance the performance of linear regression models so that they are more effective when used in Driving declinometer and vehicle weight estimation systems