Cars 24 Case Study documentation

Overview of Data:

- The data contain some features that can be directly given to model (after some basic preprocessing). Example: odometer reading, Fuel type
- Some of the features need understanding and proper transformation as it is in raw form(text and NULL values). Example: `engineTransmission_engineOil` and its related variables.
- In Raw form, It contain 26307 datapoints and 72 variables and 1 target variable.
- Target variable(rating) is unevenly distributed between 0.5 to 5:

```
4.0    10152
3.5     5944
3.0     4379
4.5     2209
5.0     1623
1.0     1456
2.5      420
2.0      117
1.5        6
0.5        1
```

Problem Approach:

- Check Distribution of Dependent and independent variables.
- Calculation of Age feature from the registration year and the inspection date.
- Understanding of inspection parameter features of engine:
    - All inspection parameters are divided into several features:
        - The first of them describe if the engine is ok(yes) or not ok(no) wrt that particular parameter.
        - If it is not ok (NO) then each issue/s are described in the following consequent features:

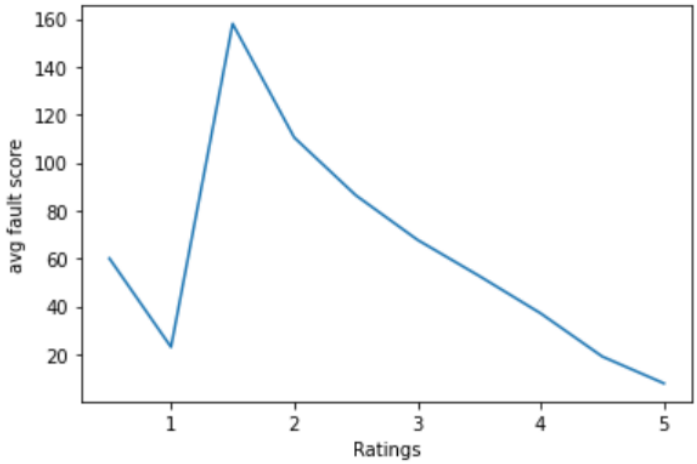
```
'engineTransmission_battery_value',  → Yes/ NO
'engineTransmission_battery_cc_value_0',  → problem(if first is no)
'engineTransmission_battery_cc_value_1',
'engineTransmission_battery_cc_value_2',
'engineTransmission_battery_cc_value_3',
'engineTransmission_battery_cc_value_4
```

- All these features can be replaced with ISSUE/Problems with values 0&1 giving us a sparse feature matrix(as vectorization in Text Processing).

- Standardization of age and odometer reading.
- Split data first into features and label and then into train & test(8:2)
- Populating target label in train data using Random Oversampler technique.

Intermediate step (To check for wrongly rated engine):

- Created a ‘score’ variable which is essentially the sum of all fault(i.e fault score).
- Plotted a graph between ratings and average fault score:



- As we can see from the graph,for rating1.5 - 5 average fault score is decreasing as the rating is increasing(which makes sense)
- For rating 1, average fault score is very low. This can be due to wrongly labelled ratings. Same goes for 0.5 rating.
- We can further investigate this by running model on different sets of cross validation data. Model score metrics should not fluctuate unless there is some outliers or wrong label.

- Classification vs Regression: As for classification, we have 10 classes(0.5, 1, 1.5…..4.5, 5), for regression we have range(0,5). For now I will be going for classification.
- Creating base logistic regression model.

```
Accuracy on Train Data : 0.6325
Accuracy on Test Data: 0.4027

F1 score = 0.4
```

- Creating XGboost classification model:

```
Accuracy on Train Data : 0.6955
Accuracy on Test Data: 0.3968
F1 score = 0.4
```

- Xgboost Hyperparameter tuning using randomized search CV:

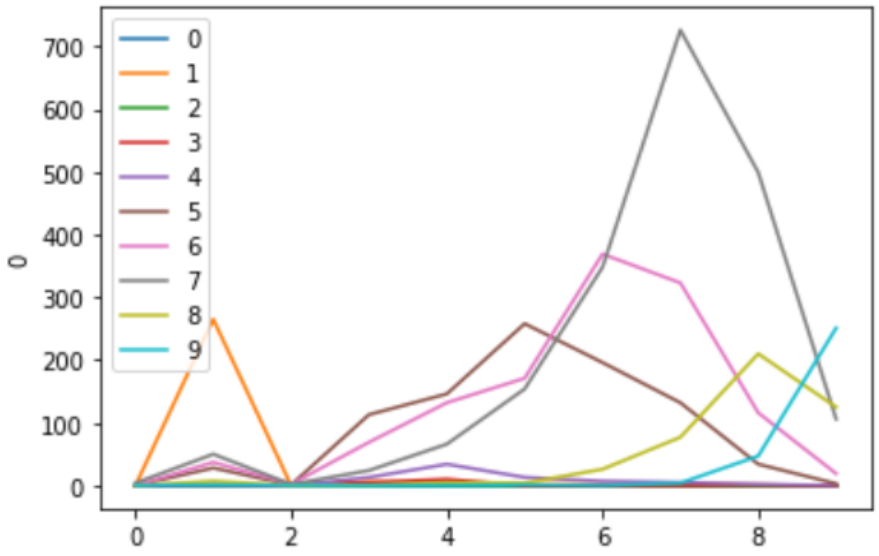
```
Accuracy on Train Data : 0.7210
Accuracy on Test Data: 0.4340
F1 score = 0.43
```

- Model Evaluation

  Basics:

  a. How well does your model work?
     - For the imbalanced data, model is quite underfit. But is able to predict in close range.

  b. How do you know for sure that's how well it works?
     - By plotting a confidence graph between actual rating distribution(y-axis) over predicted(X-axis: rating* 2 -1):



     We can observe that the actual rating distribution is concentrated around the predicted label with peak at the predicted = True for most of the case.

  c. What stats did you use to prove its predictive performance and why?
     - Base metrics:
       - Accuracy: to check overall performance of data. Worst accuracy possible for 10 classes is 10%(Random classification), so accuracy over 40% is good.
       - Roc score: To check the overall prediction capacity of model.
     - Advance metric:
       - Prediction distribution: Check the plot of prediction distribution. Prediction distribution should show its peak at actual rating.

  d. What issues did you encounter?
     - Imbalanced data: Data was imbalanced for most of the ratings with minimum of 3 datapoints.
     - Incorrect label: While it can be observed, but this often leads to wrong model learning.
     - Less knowledge on car specification: Since I have less knowledge of car and its fault occurrence, validating through each features was irregular.

       a. What other data (if any) would have been useful?

- Car type(SUV, Sedan)
- Registration place
- Ownership hand
- mileage

b. What are some other things you would have done if you had more time?

- Explored Comments: sentiment check
- Ranking each issue/problems
- Explored more on issues to get more insights
- Run a sequential model for sparse data

Determining wrong rating

Basic assumption:

- We can categorise data features broadly into two category: first is the age and km driven, Second is the issue/problems in the inspection.
- From the Second category, If we weigh every issue/problems same, then more the problems a car have, the less rating should it have(when only seeing these features)

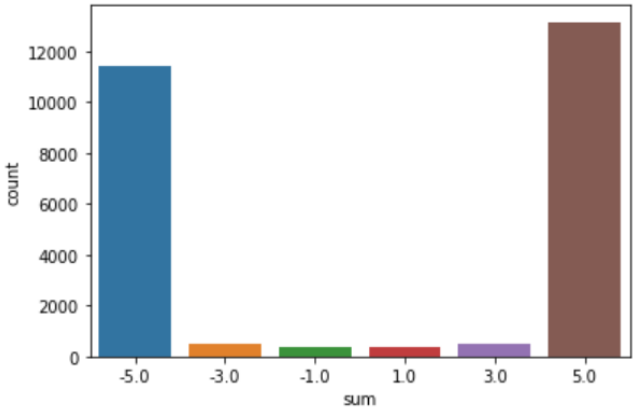
I have used two approach to check for the wrong rating:

- Intermediate steps as mentioned above
- Cross validation:
  - divided data into 10 splits, used 9 for training 1 for testing.
  - Repeated it 5 times with random initiation.
  - Mark the index of train and test as validation step.
  - Check for that validation steps when there is huge difference in train and test accuracy.
  - Observe the test data for that validation steps:

```
Validation: 0
Accuracy on Train Data : 0.5417300219631694
Accuracy on Test Data: 0.5218548080577727
Validation: 1
Accuracy on Train Data : 0.5410542321338064
Accuracy on Test Data: 0.5286963131889015
Validation: 2
Accuracy on Train Data : 0.5382665990876837
Accuracy on Test Data: 0.5458000760167236
Validation: 3
Accuracy on Train Data : 0.5377175198513262
Accuracy on Test Data: 0.5419992398327632
Validation: 4
Accuracy on Train Data : 0.5379709410373373
Accuracy on Test Data: 0.5408589889775751
Validation: 5
Accuracy on Train Data : 0.5370839668862983
Accuracy on Test Data: 0.5435195743063473
Validation: 6
Accuracy on Train Data : 0.5402094948471026
Accuracy on Test Data: 0.5313568985176739
Validation: 7
Accuracy on Train Data : 0.5395531528487562
Accuracy on Test Data: 0.5334600760456274
Validation: 8
```

For validation 1 & 2, there is a difference of 2% in train and test accuracy(while almost same for other validation set)

- Predict test data and compare the result from actual
- Store in a dataframe row as data points and columns as validation steps with values:
    - 0: not present in test data
    - 1: classified
    - -1: miss classified
- Mislabelled data will tends to show more noise in classification. We can plot to the the frequency.



    - The datapoints having sum can be hypothised as outliers/mislabelled.
    - This can be further solidified that, as we increase the cross validation steps, same set of datapoints are plotted against negative max value.

- Removing this datapoints, and re running the model, its performance increases drastically.
- `Accuracy on Train Data : 0.9592151675485009`
- `Accuracy on Test Data: 0.9084507042253521`