Capstone Project

Machine Learning Engineering Nano Degree

DISTINGUISH BAD DRIVERS FROM GOOD ONES

1. Definition

a. Project Overview

This project aims to identify divers which are aggressive in nature. Initially there were three level of divers but as the importance is put on identifying the aggressive drivers, we have modified this problem and converted it into a binary problem.

This is a case of classification problem which falls under the supervised learning problem. Driver style is the target variable whereas vehicle data related to the driver vehicle, preceding vehicles, the road type and weather conditions are the features. The data is available in the form of 3 csv files.

Data Source: I will be sharing the raw data files. As suggested in the course.

b. Problem Statement

Develop a supervised learning model using classification algorithm to predict which drivers are aggressive in nature using the captured features.

c. Metrics

Since this is a classification problem, metric used here will be accuracy and recall. The accuracy is to ensure overall model is good and to ensure we are predicting actual aggressive drivers with high accuracy; we want recall to be high.

Mathematically it is defined as:

Accuracy = (TP + TN)/ (TN + TP + FP + FN)

Mathematically, it is defined as:

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

2. Data Exploration And Preparation

We have three datasets and we did initial exploration to understand how many rows and columns each dataset contains. This led us to the following output

```
# Understanding the data dimensions
print("The rawdata has {} rows and  {} columns".format(rawdata.shape[0], rawdata.shape[1]))
print("The traveldata has {} rows and  {} columns".format(traveldata.shape[0], traveldata.shape[1]))
print("The weatherdata has {} rows and  {} columns".format(weatherdata.shape[0], weatherdata.shape[1]))
```

```
The rawdata has 12994 rows and  5 columns
The traveldata has 162566 rows and  10 columns
The weatherdata has 162566 rows and  9 columns
```

In total across three datasets we have 18 variables. We used these variables to create the new variable. These variables were then combined into one dataset.

```
V1  Date & time of data
V2  Length of vehicle in cm
V3  Lane of the road of the vehicle(1 &2)
V4  Speed of the vehicle (kph)
V5  weight of vehicle in kg
V6  Number of axles
V7  ID of the preceding vehicle
V8  Speed of the preceding vehicle (kph)
V9  Weight of the preceding vehicle (kg)
V10 Length of preceding vehicle (cm)
V11 Time gap with the preceeding vehicle in seconds
V12 Weather details-Air temperature
V13 Precipitation-(Clear,rain, snow)
V14 Precipitation intensity- none, low, moderate, high
V15 Relative humidity-
V16 Wind direction-0 to 360)
V17 Wind speed in m/s
V18 Condition of the road wrt weather
```

Here is the list of variables which we created from travelData.csv along with the code snippet.

1. SwitchCount – How many switches did the driver made during the travel distance.
2. TotalJourneyTime – Total time taken for the journey.
3. RateOfSwitch – What is the rate of overtalking the other vehicle.
4. AvgSpeed – Average speed of the vehicle
5. MaxSpeed – Maximum speed of the vehicle
6. AvgSpeedPrecedingVehicle – Average speed of the preceding vehicle
7. MaxSpeedPrecedingVehicle – Max speed of the preceding vehicle
8. AvgTimeGapPreceding – Average gap with the preceding vehicle in seconds
9. MaxTimeGapPreceding – Max gap with the preceding vehicle in seconds
10. MinTimeGapPreceding – Min gap with the preceding vehicle in seconds
11. MedianTimeGapPreceding – Median gap with the preceding vehicle in seconds
12. AvgLengthvehiclePreceding – Average length of the preceding vehicle in cm
13. MaxLengthvehiclePreceding – Max length of the preceding vehicle in cm
14. MinLengthvehiclePreceding – Min length of the preceding vehicle in cm
15. MedianLengthvehiclePreceding – Median length of the preceding vehicle in cm
16. AvgWeightvehiclePreceding – Average weight of the preceding vehicle in kg
17. MaxWeightvehiclePreceding – Max weight of the preceding vehicle in kg
18. MinWeightvehiclePreceding – Min weight of the preceding vehicle in kg
19. MedianWeightvehiclePreceding – Median weight of the preceding vehicle in kg

20. RoadCondition – Existing variable just took the first value of each row for the each driver data.

```
for i in unique_id:
    ID.append(i)
    dfTemp = traveldata[traveldata.ID == i]

    JourneyTime = pd.to_datetime(dfTemp.V1.iloc[dfTemp.shape[0]-1]) - pd.to_datetime(dfTemp.V1.iloc[0])
    TotalJourneyTime.append(JourneyTime)

    AvgSpeed.append(round(dfTemp.V4.mean(),0))
    MaxSpeed.append(round(dfTemp.V4.max(),0))

    AvgSpeedPrecedingVehicle.append(round(dfTemp.V8.mean(),0))
    MaxSpeedPrecedingVehicle.append(round(dfTemp.V8.max(),0))

    AvgTimeGapPreceding.append(round(dfTemp.V11.mean(),0))
    MaxTimeGapPreceding.append(round(dfTemp.V11.max(),0))
    MinTimeGapPreceding.append(round(dfTemp.V11.min(),0))
    MedianTimeGapPreceding.append(round(dfTemp.V11.median(),0))

    AvgLengthvehiclePreceding.append(round(dfTemp.V10.mean(),0))
    MaxLengthvehiclePreceding.append(round(dfTemp.V10.max(),0))
    MinLengthvehiclePreceding.append(round(dfTemp.V10.min(),0))
    MedianLengthvehiclePreceding.append(round(dfTemp.V10.median(),0))

    AvgWeightvehiclePreceding.append(round(dfTemp.V9.mean(),0))
    MaxWeightvehiclePreceding.append(round(dfTemp.V9.max(),0))
    MinWeightvehiclePreceding.append(round(dfTemp.V9.min(),0))
    MedianWeightvehiclePreceding.append(round(dfTemp.V9.median(),0))
```

```
    switch_count = 0
    lane = dfTemp["V3"]
    currentValue = []
    nextValue = []
    for j in range(0, len(lane)-1):
        if j < len(lane):
            #print(j)
            currentValue =  dfTemp.V3.iloc[j]
            nextValue = dfTemp.V3.iloc[j+1]
            if nextValue != currentValue:
                switch_count = switch_count + 1
    SwitchCount.append(switch_count)

SwitchCountDF = pd.DataFrame({'ID':ID, 'TotalSwitches': SwitchCount, 'TotalJourneyTime': TotalJourneyTime,
                              'AvgSpeed':AvgSpeed, 'MaxSpeed': MaxSpeed, 'AvgSpeedPrecedingVehicle':AvgSpeedPrecedingVehicle, 'Ma
                              'AvgTimeGapPreceding': AvgTimeGapPreceding, 'MaxTimeGapPreceding': MaxTimeGapPreceding,
                              'MinTimeGapPreceding': MinTimeGapPreceding, 'MedianTimeGapPreceding':MedianTimeGapPreceding,
                              'AvgLengthvehiclePreceding': AvgLengthvehiclePreceding, 'MaxLengthvehiclePreceding': MaxLengthvehic
                              'MinLengthvehiclePreceding': MinLengthvehiclePreceding, 'MedianLengthvehiclePreceding': MedianLengt
                              'AvgWeightvehiclePreceding': AvgWeightvehiclePreceding, 'MaxWeightvehiclePreceding': MaxWeightvehic
                              'MinWeightvehiclePreceding': MinWeightvehiclePreceding, 'MedianWeightvehiclePreceding': MedianWeigh
                              'RoadCondition': RoadCondition}, columns = ['ID', 'TotalSwitches', 'TotalJourneyTime', 'AvgSpeed',
                                        'AvgSpeedPrecedingVehicle', 'MaxSpeedPrecedingVehicle', 'AvgTimeGapPreceding',
                                        'MaxTimeGapPreceding', 'MinTimeGapPreceding', 'MedianTimeGapPreceding',
                                        'AvgLengthvehiclePreceding', 'MaxLengthvehiclePreceding', 'MinLengthvehiclePreceding',
                                        'MedianLengthvehiclePreceding', 'AvgWeightvehiclePreceding', 'MaxWeightvehiclePreceding'
                                        'MinWeightvehiclePreceding', 'MedianWeightvehiclePreceding', 'RoadCondition']
                             )
```

We noticed that some of the drivers did not made any switches between the lanes and this when we calculated the Rate of Switch some NaN were generated these were taken care by the following code.

```
SwitchCountDF['TotalJourneyTime'] = SwitchCountDF['TotalJourneyTime'].astype('timedelta64[m]')
SwitchCountDF['RateOfSwitch'] = round(SwitchCountDF['TotalSwitches']/SwitchCountDF['TotalJourneyTime'],2)
SwitchCountDF['RateOfSwitch'][SwitchCountDF['RateOfSwitch'] == -inf] = 0
```

The weather data had some blanks and missing values we wanted to know what percent of this data is missing. The following code helped us learn that only 2% of the data is missing. We decided to delete these rows as the total percentage of missing data is very low.

From the weather data we created the following new variables:

1. Weekday – It is a number suggesting which day of the week it is.

We then used the following code to get the most common values for each of the features.

```
ID = []
AirTemp = []
Precipitation = []
PrecipitationIntensity = []
RelHumidity = []
WindDirection = []
WindSpeed = []
Weekday = []
for i in unique_id:
    dfTemp = weatherdata[weatherdata.ID == i]
    if dfTemp.shape[0] != 0:
        ID.append(i)
        AirTemp.append(Counter(dfTemp['V12']).most_common(1)[0][0])
        Precipitation.append(Counter(dfTemp['V13']).most_common(1)[0][0])
        PrecipitationIntensity.append(Counter(dfTemp['V14']).most_common(1)[0][0])
        RelHumidity.append(Counter(dfTemp['V15']).most_common(1)[0][0])
        WindDirection.append(Counter(dfTemp['V16']).most_common(1)[0][0])
        WindSpeed.append(Counter(dfTemp['V17']).most_common(1)[0][0])
        Weekday.append(dfTemp.Weekday.iloc[0])

weatherFinal = pd.DataFrame({'ID': ID, 'AirTemp': AirTemp, 'Precipitation': Precipitation,
                             'PrecipitationIntensity': PrecipitationIntensity, 'RelHumidity': RelHumidity,
                             'WindDirection': WindDirection, 'WindSpeed': WindSpeed, 'Weekday': Weekday },
                            columns = ['ID','AirTemp', 'Precipitation', 'PrecipitationIntensity', 'RelHumidity',
                             'WindDirection', 'WindSpeed', 'Weekday'])

weatherFinal['PrecipitationIntensity'] = weatherFinal['PrecipitationIntensity'].apply(lambda x: "Missing" if x == " " else x)
```

The Precipitation Intensity is a categorical variable and it had some ~200 odd blank values we converted these blanks and marked them as new category as missing. These three new datasets were than merged by the ID variable to create a final raw data which had 12847 rows and 32 features. Finally, each data type was checked and based upon the type they were converted into the categorical variable.

```
for i in catVariableList:
    finalRawData[i] = finalRawData[i].astype('category')
```

```
Out[68]: ID                            object
         TotalSwitches                float64
         TotalJourneyTime             float64
         AvgSpeed                     float64
         MaxSpeed                     float64
         AvgSpeedPrecedingVehicle     float64
         MaxSpeedPrecedingVehicle     float64
         AvgTimeGapPreceding          float64
         MaxTimeGapPreceding          float64
         MinTimeGapPreceding          float64
         MedianTimeGapPreceding       float64
         AvgLengthvehiclePreceding    float64
         MaxLengthvehiclePreceding    float64
         MinLengthvehiclePreceding    float64
         MedianLengthvehiclePreceding float64
         AvgWeightvehiclePreceding    float64
         MaxWeightvehiclePreceding    float64
         MinWeightvehiclePreceding    float64
         MedianWeightvehiclePreceding float64
         RoadCondition                category
         RateOfSwitch                 float64
         AirTemp                      float64
         Precipitation                 object
         PrecipitationIntensity        object
         RelHumidity                  float64
         WindDirection                float64
         WindSpeed                    float64
         Weekday                        int64
         VehicleLength(cm)            float64
         VehicleWeight(kg)            float64
         NumAxles                     category
         DrivingStyle                   int64
         dtype: object
```

As part of the final data preparation deal, we converted the DriverStyle variable we marked the 2 & 3 types of drivers as normal drivers. Also, we used MinMaxScaler to scale all the variables in the dataset.

In [67]:
```
from sklearn.preprocessing import MinMaxScaler

# Initialize a scaler, then apply it to the features
scaler = MinMaxScaler() # default=(0, 1)

finalRawData[numerical] = scaler.fit_transform(finalRawData[numerical])

# Show an example of a record with scaling applied
finalRawData.head(n = 5)
```

Out[67]:

	ID	TotalSwitches	TotalJourneyTime	AvgSpeed	MaxSpeed	AvgSpeedPrecedingVehicle	MaxSpeedPrecedingVehicle	AvgTimeGapPreceding
0	DR_24526	0.133333	0.002678	0.500000	0.491379	0.576923	0.420168	0.039658
1	DR_30052	0.233333	0.002975	0.441860	0.396552	0.564103	0.504202	0.082367
2	DR_40928	0.133333	0.002678	0.476744	0.482759	0.512821	0.495798	0.095790
3	DR_66033	0.200000	0.002975	0.453488	0.379310	0.538462	0.445378	0.070165
4	DR_45266	0.166667	0.002678	0.453488	0.379310	0.525641	0.453782	0.065284

5 rows × 32 columns

We also created dummy variables using get_dummies() function from pandas package.

```
# TODO: One-hot encode the 'features_log_minmax_transform' data using pandas.get_dummies()
features_final = pd.get_dummies(features)
features_final.head()
```

3. Building Models

To build the model we ran some 10 different classification algorithms. We used two different approaches to decide on the final model.

a. Used Principal components to predict the DriverStyle.

We got a very good accuracy while we used the top 10 principal components to build the final model. This resulted into decent Recall for some of the shortlisted models. Best among all was GausianNB. The Naïve Bayes model beats the benchmark model by approximately ~6% in terms of the accuracy.

Out[52]:

	model_name	model_run_time	train_accuracy	test_accuracy	train_recall	test_recall	train_precision	test_precision	train_fscore	test_fscore
0	DecisionTreeClassifier	1.083747	1.000000	0.724065	1.000000	0.385287	1.000000	0.357639	1.000000	0.370948
1	GaussianNB	0.051456	0.761260	0.753555	0.453728	0.416459	0.442268	0.416459	0.447925	0.416459
2	Linear SVM	9.907897	0.786545	0.788836	0.000000	0.000000	0.000000	0.000000	NaN	NaN
3	RBF SVM	10.360768	0.830794	0.807267	0.276044	0.201995	0.800613	0.637795	0.410539	0.306818
4	LogisticRegression	0.050970	0.806863	0.805687	0.241142	0.245636	0.622951	0.596970	0.347693	0.348057
5	BaggingClassifier	6.028900	0.982617	0.791732	0.920677	0.269327	0.997708	0.513064	0.957646	0.353230
6	RandomForestClassifier	1.090039	0.982842	0.787783	0.923850	0.234414	0.995442	0.494737	0.958310	0.318105
7	AdaBoostClassifier	3.591107	0.812394	0.801738	0.269170	0.234414	0.645120	0.574924	0.379851	0.333038
8	GradientBoostingClassifier	4.279923	0.838356	0.806214	0.322581	0.240648	0.801577	0.603125	0.460030	0.344029
9	KNeighborsClassifier	14.140511	0.843097	0.779358	0.426758	0.241895	0.725067	0.457547	0.537284	0.316476

b. Used just the final rawdata set to predict the DriverStyle. This is the final approach which we selected. Again, GausianNB model performed the best. Albeit, we beat the benchmark model by only ~2% we see a ~65% recall and considering the nature of the problem I believe

that recall is sort of important.

Out[24]:

	model_name	model_run_time	train_accuracy	test_accuracy	train_recall	test_recall	train_precision	test_precision	train_fscore	test_fscore
0	DecisionTreeClassifier	0.511261	1.000000	0.754344	1.000000	0.407731	1.000000	0.416561	1.000000	0.412098
1	GaussianNB	0.080565	0.712609	0.710111	0.654680	0.642145	0.395401	0.387509	0.493031	0.483341
2	Linear SVM	10.130323	0.786545	0.788836	0.000000	0.000000	0.000000	0.000000	NaN	NaN
3	RBF SVM	13.778903	0.795124	0.796209	0.054469	0.048628	0.792308	0.780000	0.101930	0.091549
4	LogisticRegression	0.224538	0.811265	0.808057	0.259122	0.259352	0.643890	0.606414	0.369532	0.363319
5	BaggingClassifier	3.210542	0.982842	0.821485	0.924907	0.390274	0.994315	0.623506	0.958356	0.480061
6	RandomForestClassifier	0.515514	0.987245	0.808847	0.941830	0.310474	0.998318	0.590047	0.969252	0.406863
7	AdaBoostClassifier	1.913810	0.827859	0.807004	0.334744	0.285536	0.703333	0.588689	0.453601	0.384551
8	GradientBoostingClassifier	3.090343	0.860707	0.826488	0.468006	0.372818	0.795148	0.657143	0.589214	0.475736
9	KNeighborsClassifier	14.162890	0.844452	0.785150	0.454786	0.300499	0.712510	0.485887	0.555197	0.371341

4. Final Result

Finally, GaussianNB model is selected. The model as an accuracy of ~72% and Recall ~65%. The results of some of the other models are either overfit which very high difference or has a very low recall value.

Please Note: Other techniques were also used apart from the above like xgboost, gridsearch() for GradientBoostingClassifier() to see if we can improve the results. However, we did not see much of an improvement with the recall.