

Project report

Predicting autonomous vehicle straight line collision attributes with detected traffic object and recommending vehicle features to prevent collision using artificial neural network

Jasvir Dhillon

July 05 , 2021

Table of Contents

Keywords	3
1. Definition	4
Project overview	4
Problem statement	5
Metrics	5
2. Analysis	9
EDA (Exploratory Data analysis)	9
PCA (Principal Component Analysis)	19
Benchmark	21
3. Methodology	23
Algorithm, Technique, and Implementation	23
Data pre-processing	23
Model training	27
Model evaluation and validation (benchmark model)	27
Refinement	28
Model evaluation and validation (refined model)	29
Retraining the model on updated dataset	30
Model evaluation and validation (refined model with updated dataset)	31
4. Results	33
Justification	33
Conclusion	36
Improvements / Further suggested work	36

Keywords

AWS, SageMaker, S3, data processing, data analysis, visualization, machine learning, AI, neural network, PCA, ADAS, edgecase, collision, prediction, recommendation

1. Definition

Project overview

Domain Background

Virtual vehicle engineering is where digital models of real-world vehicle prototypes are developed and deployed to replicate the behaviour of the entire real-world vehicle or its given subsystem(s) (e.g. Electric vehicle powertrain) in a pre-defined virtual setting (e.g. a digital racetrack) to evaluate certain criteria or KPI of interest. A key advantage of this is that it reduces the vehicle development time compared with traditional real-world prototype-based development and testing and hence, reduces costs. The enabling fundamental concept is that digital models can be built, scaled, swiftly tweaked and tested orders of magnitude faster than physical systems. Take for example, the time required to replace the battery and motor assembly of a real-world vehicle prototype (several days) compared with changing the digital setup of a powertrain in a digital vehicle model (few seconds).

With the advent of complex ADAS and AV driving challenges, virtual vehicle simulation has been under the spotlight of global OEMs (from Ford in the West to VW in Europe to Geely in the east). This is because vehicle simulation software today provides capability not only to create accurate virtual prototypes, but also to create detailed surrounding environments (e.g., road and traffic) which the virtual prototypes can interact with. This implies that before a vehicle system (e.g., Emergency Brake system) is placed under real-world test, it can be accurately evaluated and modified by a system's development engineer without having to leave her desk.

This is where a software like CarMaker comes into play. Although, enabling thousands (even millions) of simulations to evaluate KPIs (especially in Autonomous driving scenarios) is advantageous over traditional systems, it also means generation of equal amounts of data which contain hidden value which if left untapped, is lost. My capstone project is derived from this motivation to unlock the potential for delivering value to customers from collected data using Machine Learning and making the process of creating virtual prototypes more intuitive for non-technical users.

Overview

This project considers a particular ADAS simulation case of emergency braking in a straight line when the vehicle sensor detects an unexpected traffic object appear in its course. The key KPI here is whether a collision occurs or not. Another variant of this KPI is to measure the distance to collision at the end of the emergency braking event – if positive, it implies no collision, if negative it implies collision will occur.

The objective of the responsible systems developer/engineer in such a case is to find or evaluate vehicle parameter configurations such that the KPI is fulfilled or not.

With help of the simulation data which was created specifically for this project, the analytics and machine learning model applied to it reveal important relationships between vehicle parameters (the features) and KPI (the output) and help recommend parameter values to the systems developer/engineer to avoid a collision.

Different statistical analysis methods have been used to visualize useful patterns in the data. While a neural network has been used to predict collision probability and accurate distance to collision to recommend safe vehicle parameters for the given use case.

The entire development work has been done using AWS (managed services) including SageMaker, S3, CloudWatch.

Problem statement

Despite the enormous number of simulations and system validations performed during the development of autonomous vehicles (AVs), the end users have still experienced an undesired amount of real-world failure cases. This is largely because those AVs have been primarily trained to tackle obvious and most common driving challenges. But real-world failures have happened in scenarios where those AVs were not trained to navigate through, also called 'edge-cases'. These are cases which AV development engineers had not thought of or not had designed algorithms unable to predict and evaluate such scenarios for safety, which is why the vehicle had failed to react. Hence, the new focus is more on quality rather than the quantity of test-cases for better safety.

While enabling easier virtual vehicle prototyping and testing for a user (here an autonomous vehicle developer) she faces two key challenges:

1. While initiating the vehicle system design process, the configuration of the virtual prototype and autonomous driving scenarios can be quite overwhelming and time inefficient, given the sheer number of interacting systems and configurable parameters for different vehicle variants. Additionally, existing simulation tools do not indicate the degree of impact of parameters on a given KPI especially in the initial stages of vehicle development
2. Due to the complexity of the integrated software, the simulations can be quite computation and resource heavy to test every new vehicle variant for the same repetitive test variations

Metrics

The accuracy of the models will be measured to evaluate the performance of the networks. By using the same metric, we are able to quantify and compare both the benchmark and the final models.

The dataset will be split into a training set and a test set. The test set will be untouched, and it will be new information to the trained neural network model. There will be no priority in reserving a certain set of datapoints as test data. The entire dataset will be shuffled, and a certain percentage will be randomly taken out as test data.

Our aim is to achieve an accuracy higher than 95%. This is because vehicle collisions with traffic objects is a sensitive challenge and the aim is to be as accurate as possible, given the input data.

An accuracy of 100% on a large dataset would mean there is overfitting and is not desirable as it may lead to false belief that the model may perform equally well in new test data.

$$Accuracy = \frac{No. of correct predictions [True positives + True negatives]}{Total no. of predictions [False positives + False negatives]}$$

Getting a high accuracy would mean that the neural network has captured the dynamics of the presented system with the given data well. But we also aim to find areas where the model has failed to capture it well enough.

We will also calculate the precision and recall to check the performance of the predictions. Precision attempts to answer the question: What proportion of positive predictions was actually correct? Recall attempts to answer the questions: What proportion of actual positives was predicted correct?

$$Precision = \frac{[True positives]}{[True positives + False positives]}$$

$$Recall = \frac{[True positives]}{[True positives + False negatives]}$$

Along with the test data, there is a small separate dataset set created for validation. This set is obtained by using virtual vehicle models that are different to what was used to get the training data. This should tell us how well the model is able to make predictions for new vehicle models. Here, it will be ideal to get all collision predictions correct and all predictions for 'distance to collision' withing the 95% accuracy range.

To check the performance of 'distance to collision' predictions, we will use the aggregated accuracy

$$Agg. Accuracy = \frac{\sum |predicted distance|}{\sum |validation distance|}$$

Note, that we use the modulus for absolute values to avoid possible false aggregations in the total sum.

Workflow

The stages in which the project was covered was as follows:

0. Create and collect data [Local PC/Web]

- This is where the dataset was created using CarMaker, which is an industry standard high-fidelity simulation software for virtual vehicle development
- This included creating the scenario, configuring a generic virtual prototype, running the simulations, and storing the raw results for each simulation

1. Pre-process data [on AWS]

- This stage involved finding a systematic way of reading the unstructured results data (text files with mixed text and numbers), extracting the required information for training, and converting it into a structured format (dataframe and CSV)
- We check the data for inconsistencies like missing values and otherwise unexpected errors since we are processing over 194,400 files

2. Explore and transform data [on AWS]

- Here we check the distribution of feature and output data. Some features are equally distributed across values, and some are not. We also check the distribution of the key output value – ‘distance to collision’. This value in other terms gives us the braking distance of the vehicle but measured w.r.t the position of the traffic object in the simulation scenario
- We also check the number of collision and non-collision cases that have been recorded which will give us an indication of balanced this data is. However, an imbalance here is not critical to us since we do not aim to create a classification model. The main goal is to accurately predict the ‘distance to collision’ – which if is done well all collision predictions will be automatically accurate

3. PCA [on AWS]

- Principal Component Analysis will be used to understand and reveal the relationships (dependencies and magnitude of dependencies) of features on the outputs. This analysis will help us discover relations between vehicle parameters and the KPI visually and in an intuitive manner
- We could choose to use only the principal components (relations > 80%) for training the model which will reduce the complexity of the training job. However, in our case we have chosen not to reduce the feature size and instead use PCA solely for analysis purposes since, collisions with traffic objects are a sensitive subject we want to get high model accuracy. Though, this may be considered in the future when the number of features increase

4. Decide the type of Machine learning model, Train the model [on AWS]

- Our goal is to create a regression model so we can predict the ‘distance to collision’ and with the help of that also determine how well have we been able to predict whether a collision will occur or not (the KPI). This has been done using a sequential artificial neural network

5. Deploy/Test the model [AWS]

- Once the model has been trained on the training data, its performance is evaluated with test data which is an untouched, separated part of the original dataset

- The model's performance is evaluated with validation data
- When the test results are not as desired or if the accuracy is lesser than 95%, the neural network architecture is updated, the model is retrained and tested again. The target accuracy is finally surpassed

2. Analysis

EDA (Exploratory Data analysis)

Dataset and inputs

Before we dive into the exploratory data results, let's understand the scenario and input data.

The dataset has been generated by me using the software CarMaker. A relatively simple yet critical case has been chosen. A generic vehicle model (Demo Tesla) with a single simplified LIDAR (ideal Object) sensor used for the surrounding environment's perception. This implies that LIDAR ray tracing and signal processing are not required to detect the traffic object.

The considered case: A vehicle cruising at a constant speed on a country-side road suddenly come across a traffic object (a wild boar) which is detected by the LIDAR sensor and emergency brakes are suddenly applied with the objective of not colliding with the traffic object.

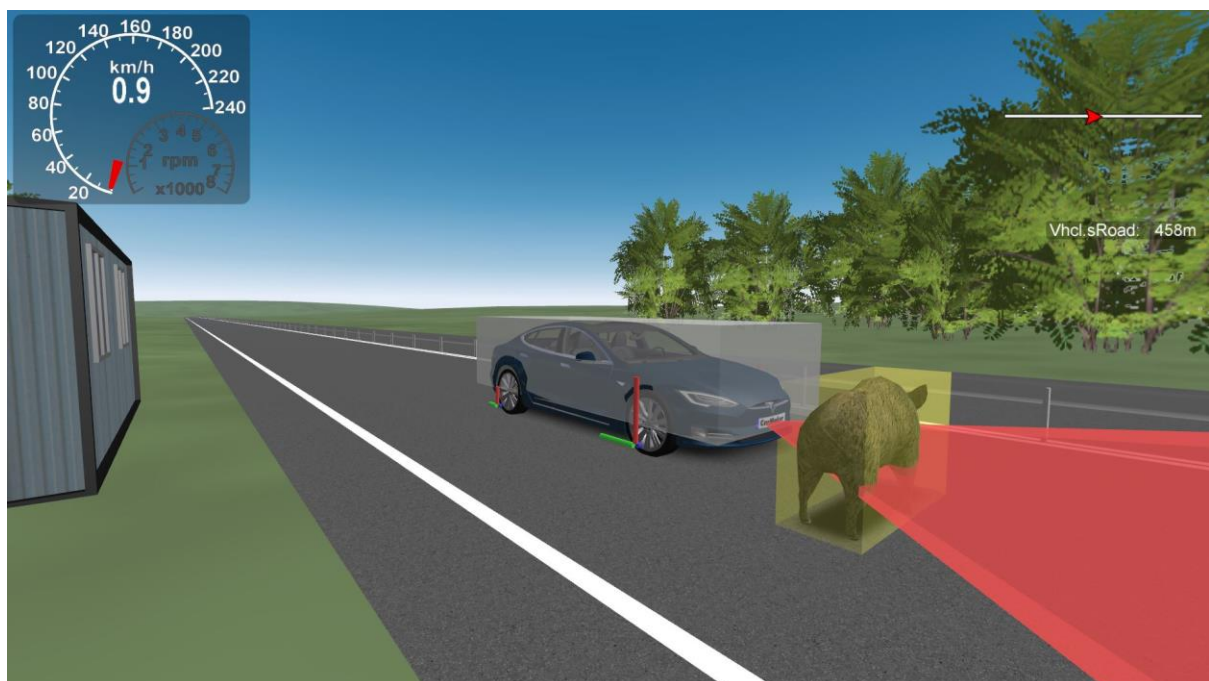


Fig: scenario: unexpected traffic object (wild-boar) appearing in-front of autonomous vehicle (CarMaker IPGMovie)

The following parameters (features) have been considered and all combinations have been used to run a corresponding simulation test:

Parameters (features)

Parameter	Description	Values
speed	Vehicle cruise speed (kph)	80, 100, 1520, 140

body_mass	Vehicle body mass (kg)	1200, 1650, 2100, 2550, 3000
cogx	Centre of Gravity in x-axis measured from rear hitch point (m)	2, 2.4, 2.8
tire	Standard tire model with varying rolling resistance factor	0.008 (tire_1), 0.015 (tire_2), 0.03 (tire_3)
road_mu	Road Coefficient of friction	0.4, 0.7, 1
react_time	Reaction time of driver to brake (s)	0.1, 0.2, 0.4
obj_dist	Distance of traffic object (wild boar) in-front of vehicle when it suddenly appears on the road	40, 60, 100, 140
update_rate	LIDAR sensor update rate (Hz)	10, 40, 60
pedal.ratio	Brake pedal ratio	3, 1.5, 2.5, 4, 3
boo.ampli	Brake booster amplification factor	5, 6, 5, 3, 5
mc.area	Brake master cylinder area (cm ²)	4.5, 3.5, 4.5, 6, 3.5
pf.area	Brake front piston area (cm ²)	23, 30, 15, 12, 30
pf.rbrake	Brake front radius (m)	0.1, 0.07, 0.1, 0.14, 0.1
pr.area	Brake rear piston area (cm ²)	11, 14, 15, 8, 14
pr.rbrake	Brake rear radius(m)	0.1, 0.07, 0.1, 0.14, 0.1

Out of the above parameters, all brake parameters are combined into sets of 5 'Brake' parameter files as below:

Hydraulic Brake file	Brake parameter variations [pedal.ratio, boo.ampli, mc.area, pf.area, pf.rbrake, pr.area, pr.rbrake]
HydESP_1	[3, 5, 4.5, 23, 0.1, 11, 0.1]
HydESP_2	[1.5, 6, 3.5, 30, 0.07, 14, 0.07]
HydESP_3	[2.5, 5, 4.5, 15, 0.1, 15, 0.1]
HydESP_4	[4, 3, 6, 12, 0.14, 8, 0.14]
HydESP_5	[3, 5, 3.5, 30, 0.1, 14, 0.1]

Along with these features, two output channels have been logged in the simulation data which will be used for training the model and also used to evaluate the trained model:

Output channel	Description
collision	A Boolean 'labelled' value saying whether a collision between the ego vehicle and the traffic object has been detected or not
dist_to_col	The distance between the ego's vehicles front bumper and the traffic object

Total no. of variations: 97,203

Total no. of data points: $97,203 \times 17 = 1,652,451$

Total no. of data files: 194,406

The idea behind this arrangement is to cover all possible variations of critical vehicle parameters such that each variation could represent a potentially new vehicle configuration within the parameter ranges. Hence, when the model that is trained on this data is tested on a new vehicle model configuration, we expect it be close or within the trained parameter data range.

These simulation tests have been generated using an automated test configurator available within the CarMaker software ran for over a 30-hour period with parallel computing.

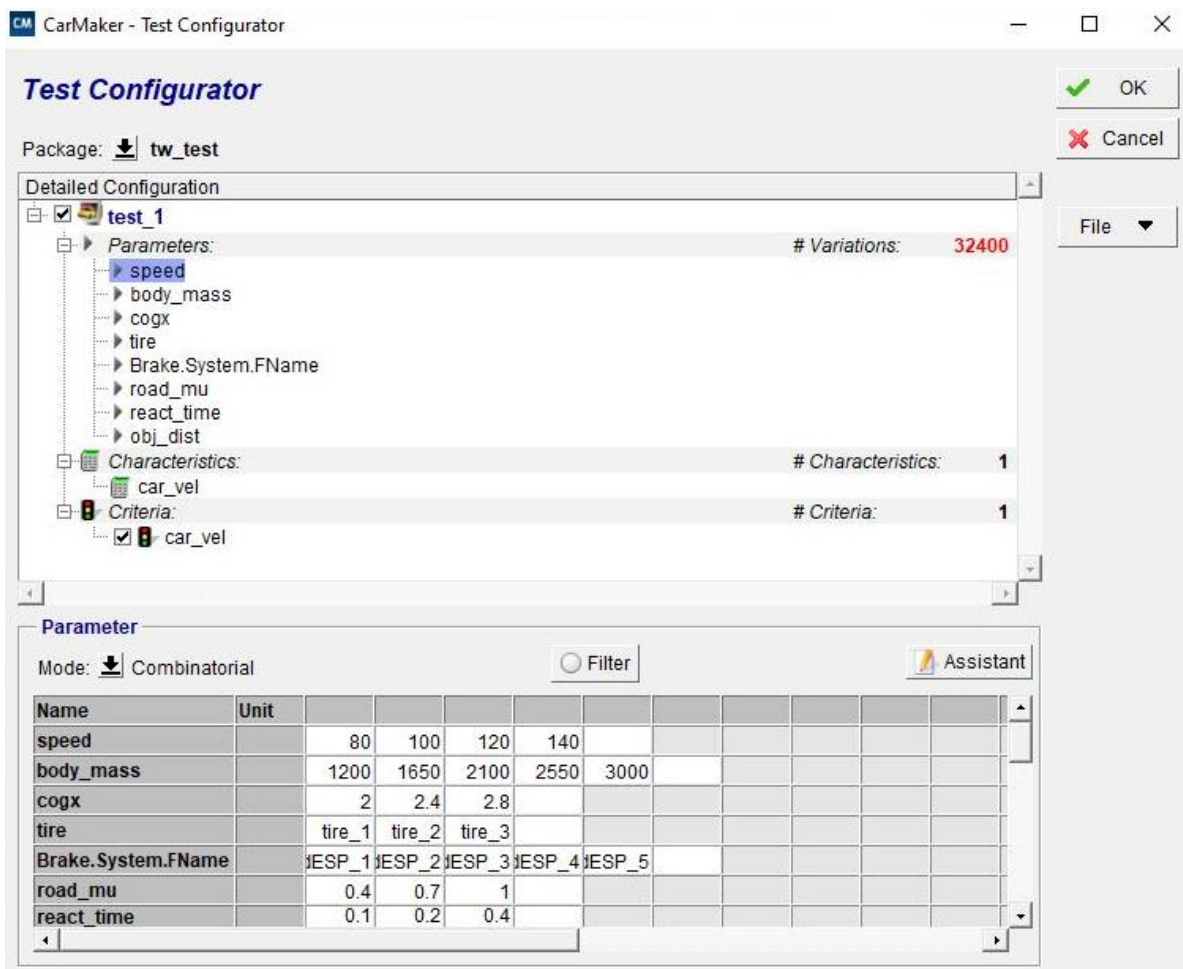


Fig: Test Configurator used to create all the CarMaker test variations

Data exploration and visualization

In the histograms below, the y-axis represents the counts, x-axis the feature and the data is characterized by colours that represent the different sensor update frequency values.

- a. The features: 'cogx', 'obj_dist', 'road_mu', 'react_time', 'tire_rr', 'speed' and 'body_mass' are in equal counts for the three different sensor update rates used.

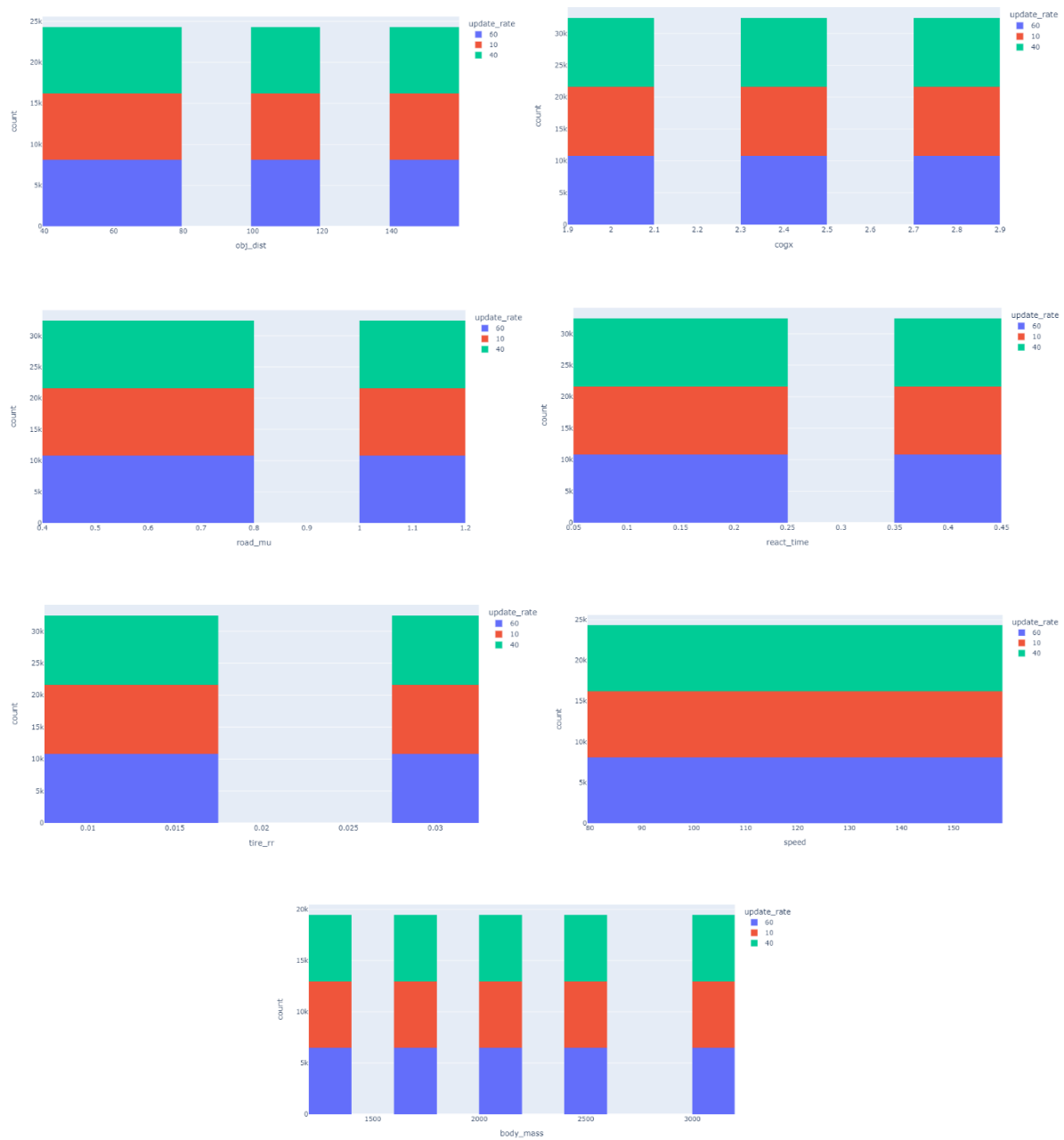


Fig: data distribution ('cogx', 'obj_dist', 'road_mu', 'react_time', 'tire_rr', 'speed', 'body_mass')

- b. Vehicle hydraulic brake system features: 'boo.ampli', 'mc.area', 'pf.area', 'pf.rbrake', 'pr.area', 'pr.rbrake', 'pedal.ratio'.
- These are unequally distributed. It has been kept this way because all the combinations of these parameters would create a very large number of variations which would require a much larger compute capacity than that currently available to finish the simulations in good time
 - Due to the unequal distribution the dataset doesn't cover the full array of Brake parameter variations. Hence, we can expect lower prediction accuracy when it comes to brake parameters specifically.

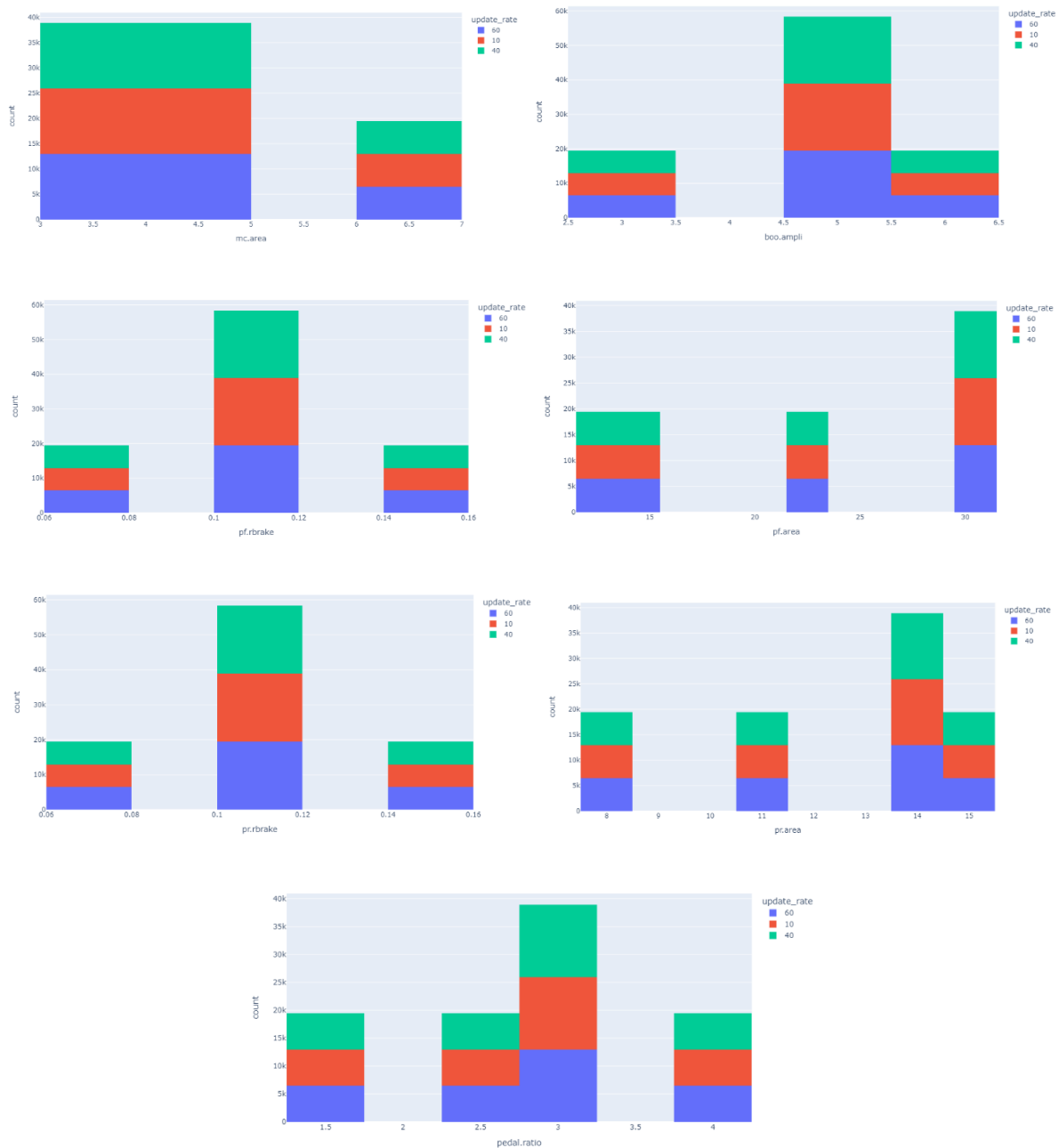


Fig: data distribution ('boo.ampli', 'mc.area', 'pf.area', 'pf.rbrake', 'pr.area', 'pr.rbrake', 'pedal.ratio')

c. Feature relations with output (collisions)

- These plots are complementary to PCA (Principal Component Analysis) plots and insightful. By plotting the average of collision (0 or 1) for datapoints for a given feature, it reveals trends on how the features (vehicle parameters) affect the output value (collision)
- For example- 'body_mass': As it increases the probability of collision increases dramatically according to the plot. It makes sense since braking distances for heavier vehicles is higher compared to lighter vehicles with the same tyre and brake properties

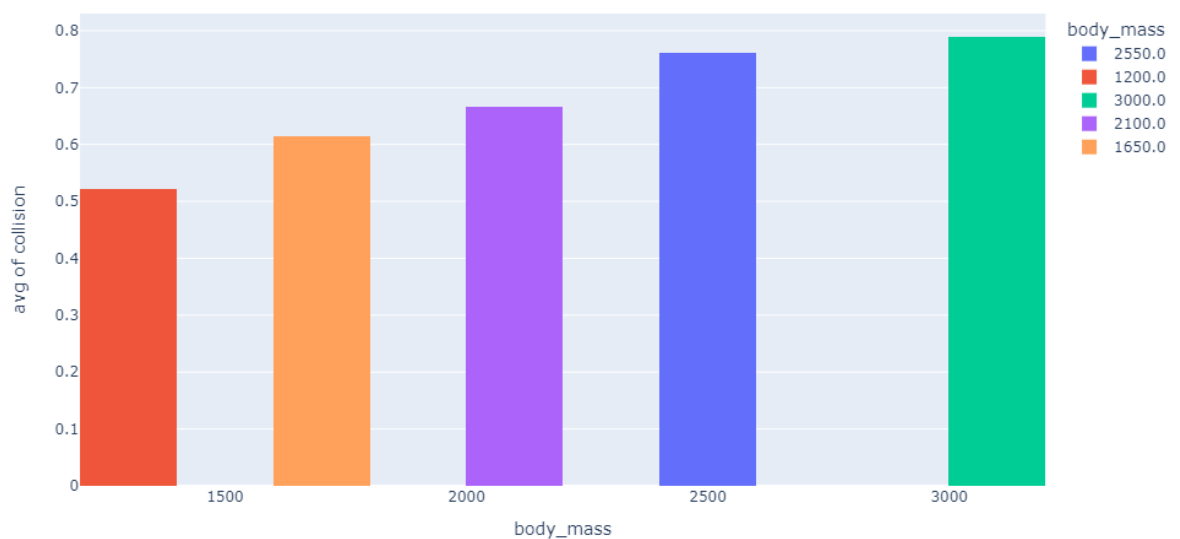


Fig: Data showing high influence of variation in vehicle mass to avoid collision with traffic object in the given scenario

- For another case - cogx: Based on the plot, the changes in centre of gravity location on the longitudinal axis of the vehicle body does not have significant influence on collision occurring or not. Increasing this parameter increases the weight transfer onto the front tyres but is also takes weight off from the rear tyres. The overall effect is not drastic, which can be seen from the data plots

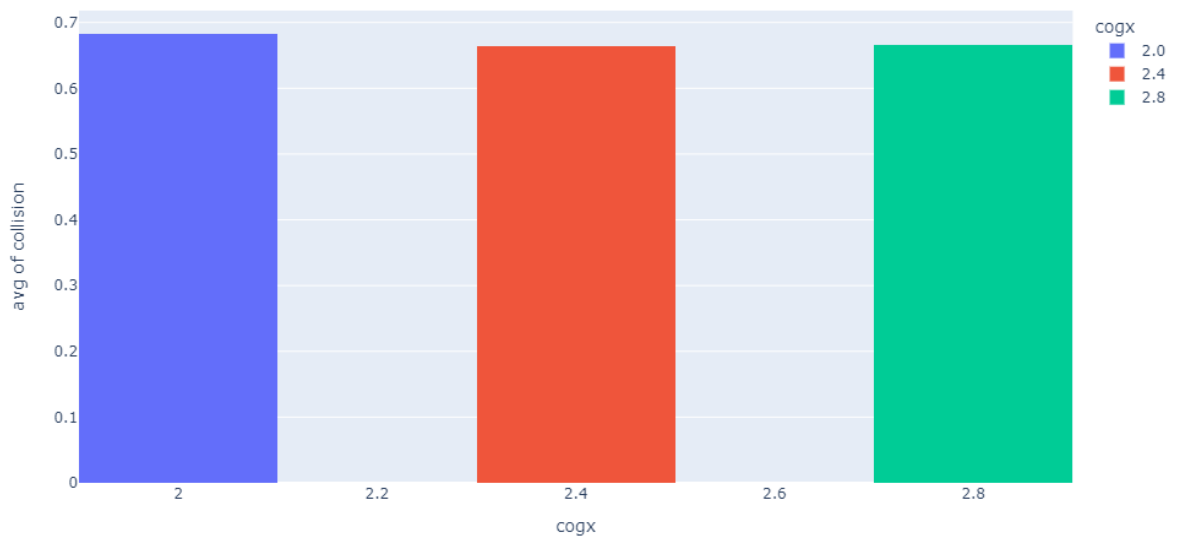
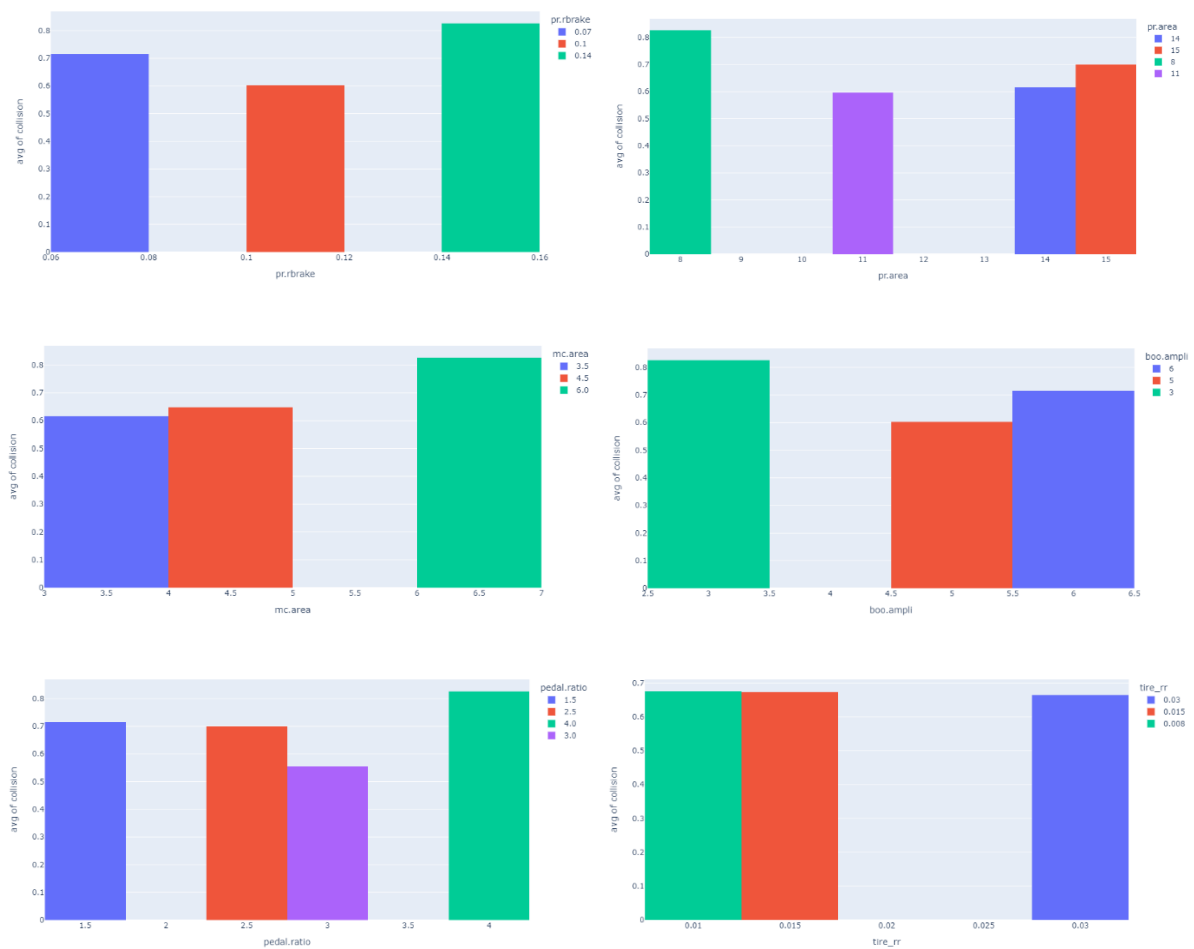


Fig: Data showing low influence of variation in vehicle centre of gravity (longitudinal position) to avoid collision with traffic object in the given scenario

Similarly, the remaining plots give us valuable insights and help us make sense of the data



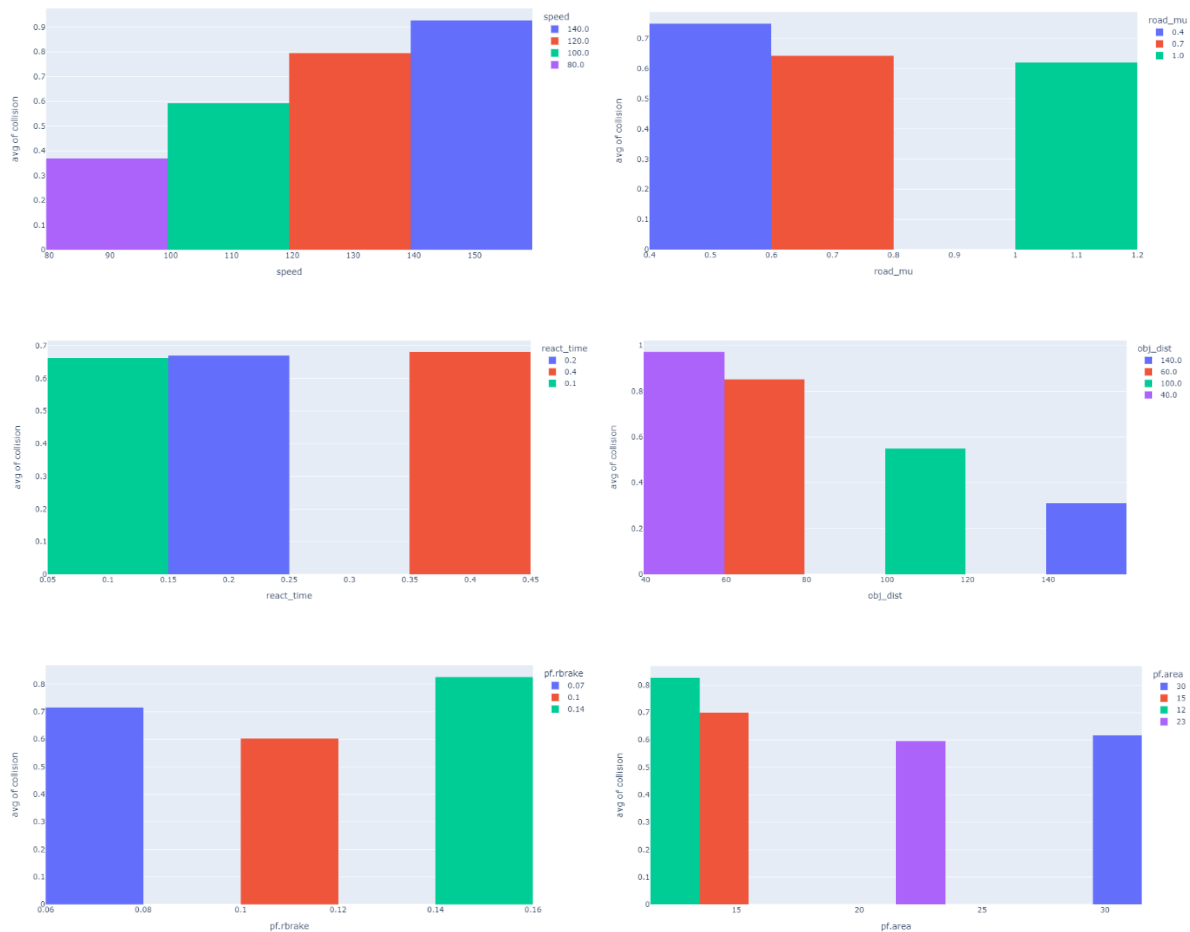


Fig: Trends in vehicle parameter effects on collision occurrence

Output data distribution

The graphs below show histograms plotted for the two recorded output values: 'collisions', 'dist_to_col'. The colours represent a third dimension visually hence it makes the more intuitive to understand.

- Plots with 'obj_dist' as the coloured dimension. 'obj_dist' represents the straight line distance in front of the vehicle when the traffic object is detected (m).
 - With the graphs below we can get a very good understanding of the how a certain 'feature' relates to the output (collision or no collision or distance to collision). In the plotted case, we see that collision cases are lower for higher 'obj_dist' and higher for lower 'obj_dist'
 - The plot following after gives a more detailed view of the first plot. It shows that the distance to collision is higher and lot more in the positive range when the vehicle is far away from the detected traffic object and vice versa



Fig: Data distribution for 'collision' instances and 'distance to collision' values classified by 'obj_dist'

b. Plots with 'speed' as the coloured dimension. 'speed' represents the cruising vehicle speed when the traffic object is detected on collision course (kph).

- From the graphs below we see that collision cases are higher for higher vehicle speeds and lower for lower speeds.
- The second plot gives a more detailed view of the first plot. The distance to collision is higher in magnitude on the negative side (implying beyond collision point) as the cruising speed increases. The histogram skew for different speeds suggest that collision cases are lesser at lower speeds which is expected.

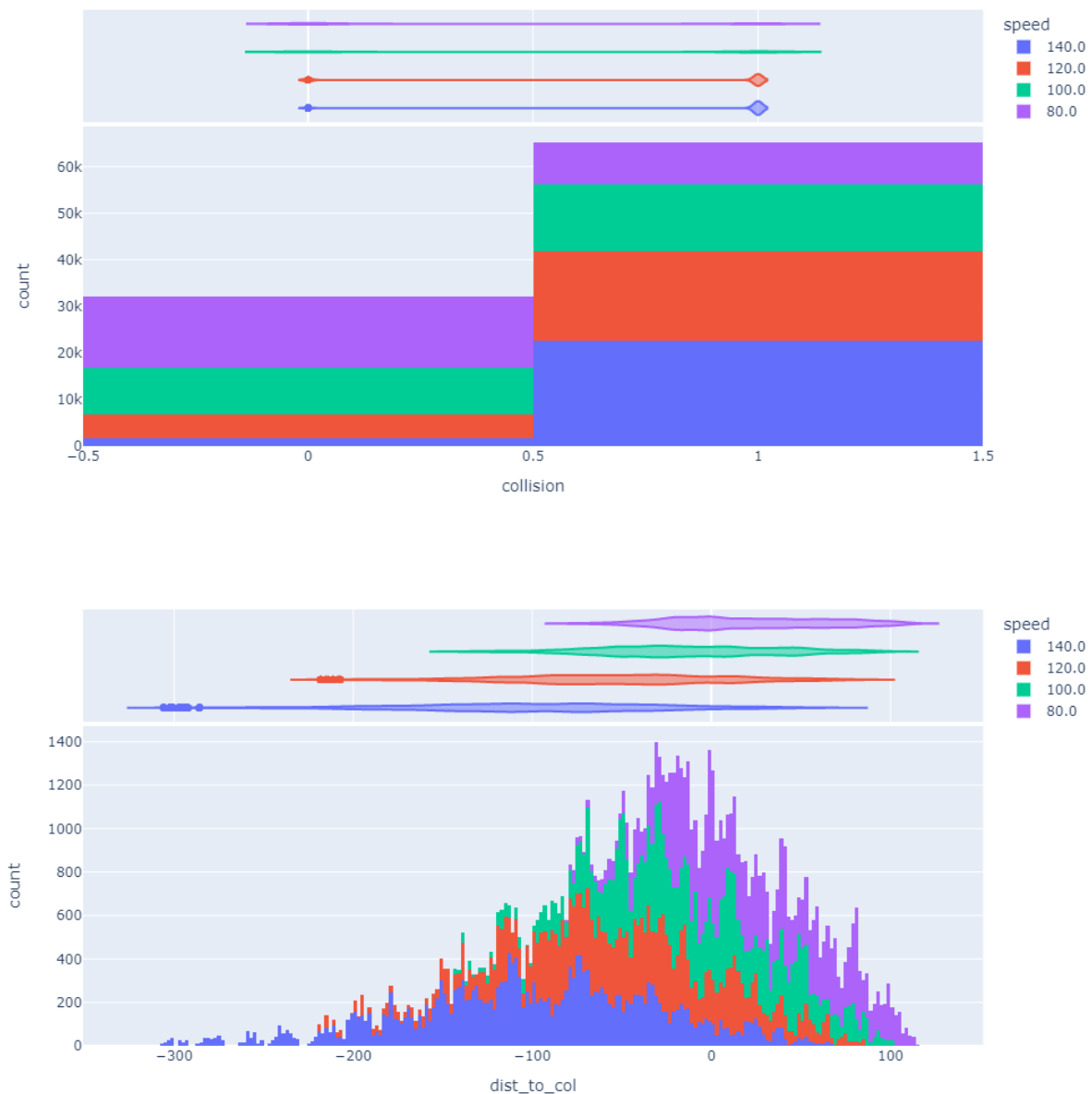


Fig: Data distribution for 'collision' instances and 'distance to collision' values classified by vehicle cruising speed

c. 'Distance to collision' relation with features

- The plots above show a different way of representing data. The coloured heat bar helps us find out visually obvious trends in the data, just like we saw with histograms.
- For example, we can see that for a given distance from traffic object, the distance to collision at the end of braking increases with increasing speed (bright yellow points) and decreases with decreasing speed (dark blue points)

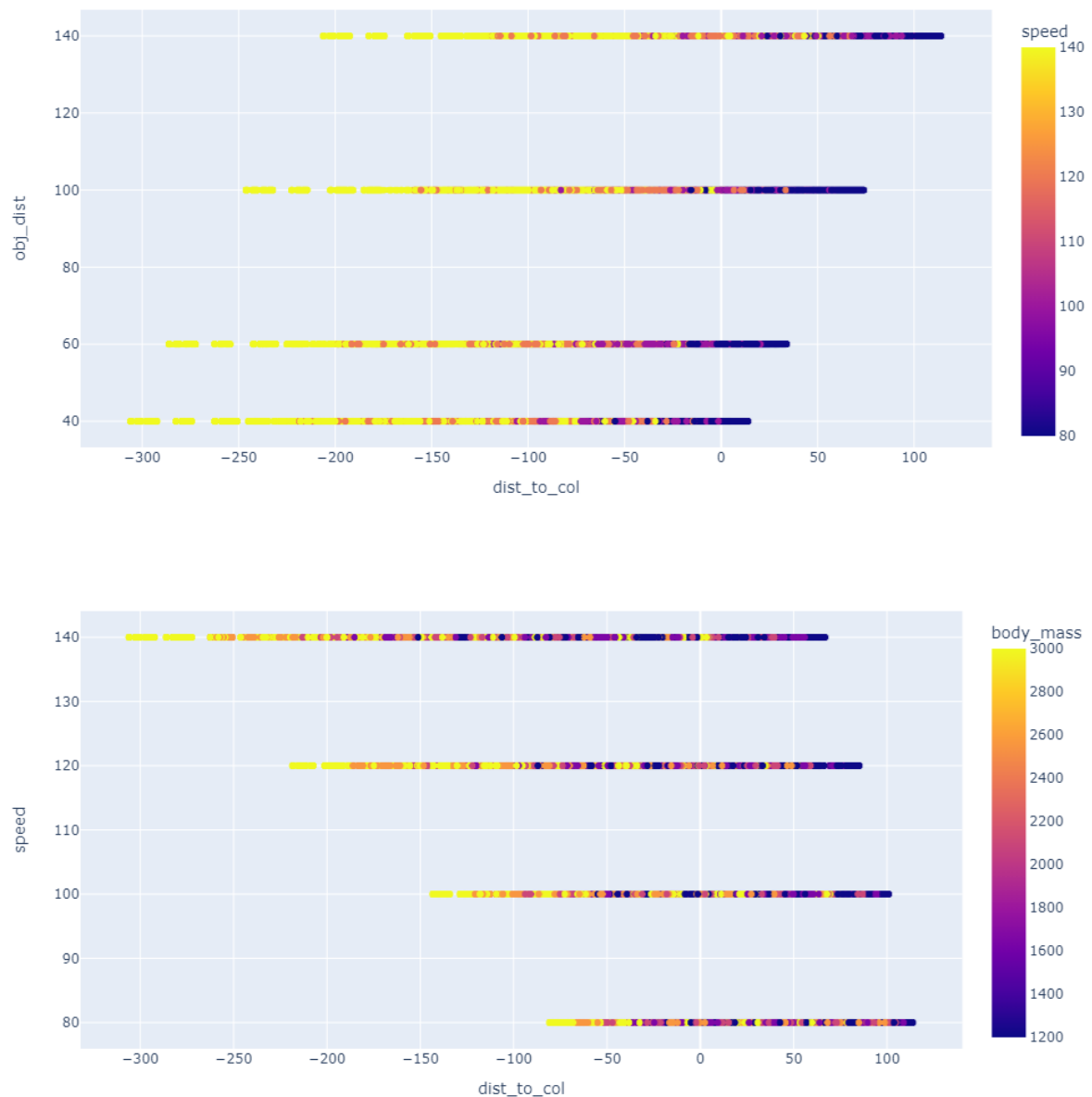


Fig: Data distribution for 'distance to collision' values classified by vehicle cruising speed and mass

PCA (Principal Component Analysis)

PCA is a statistical technique typically used to reduce the dimensionality of the data matrix. This helps in reducing the training complexity and required computation power. It exploits variability in data to create weighted 'principal components' (each composed as a mix of original features) which have the strongest relations with the given output.

Here, though, we aim to use PCA only to give us insights into key relationships between features themselves and outputs. The observations were as below.

1st principal component

The first principal component which has the highest variability among all components shows the strong relationship between all the brake parameters. Note that negative-positive weight relations mean, if the former decreases the latter increases in value and vice-versa for same-sign weighted features in that principal component.

For example, the feature dataset in our case suggests that as Master cylinder area (mc.area) of the brake system increases, the Piston areas (pf.area/pr.area) decrease. This reveals a design decision for the Brake system for the vehicle(s) in the dataset involved!

Note: The features that contribute to this component as shown on the y-axis

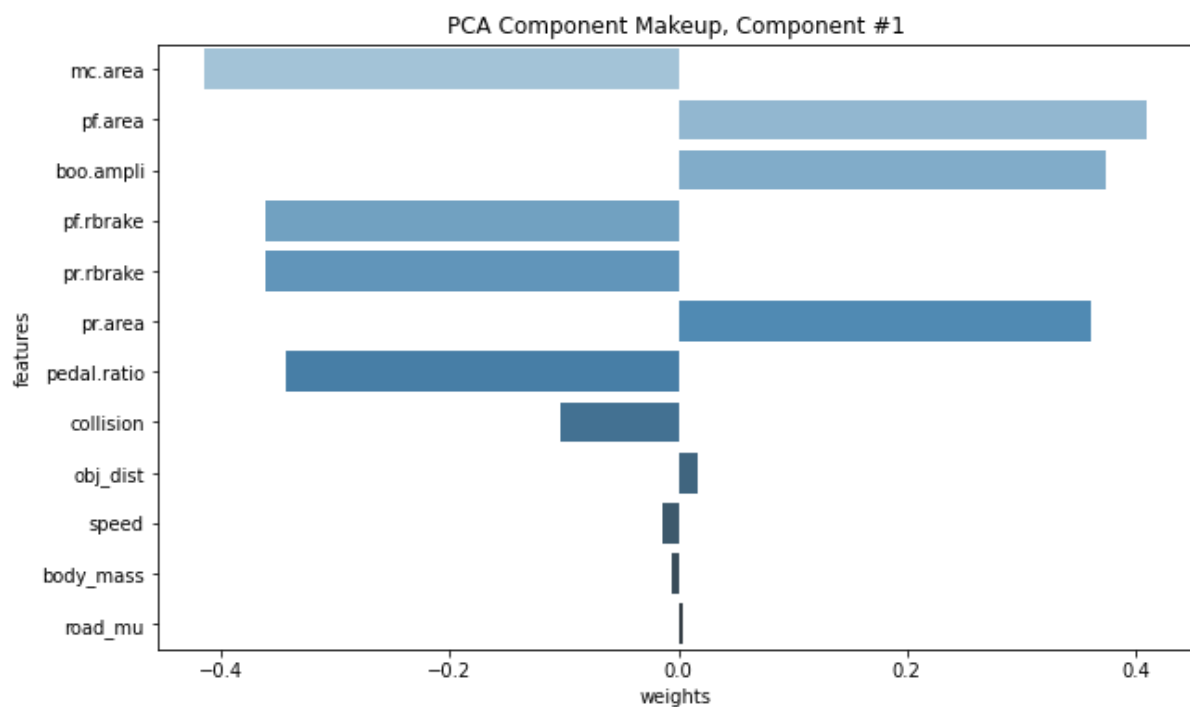


Fig: Principal component #1 feature composition

2nd principal component

The second component shows the relation between the output collision and all the features

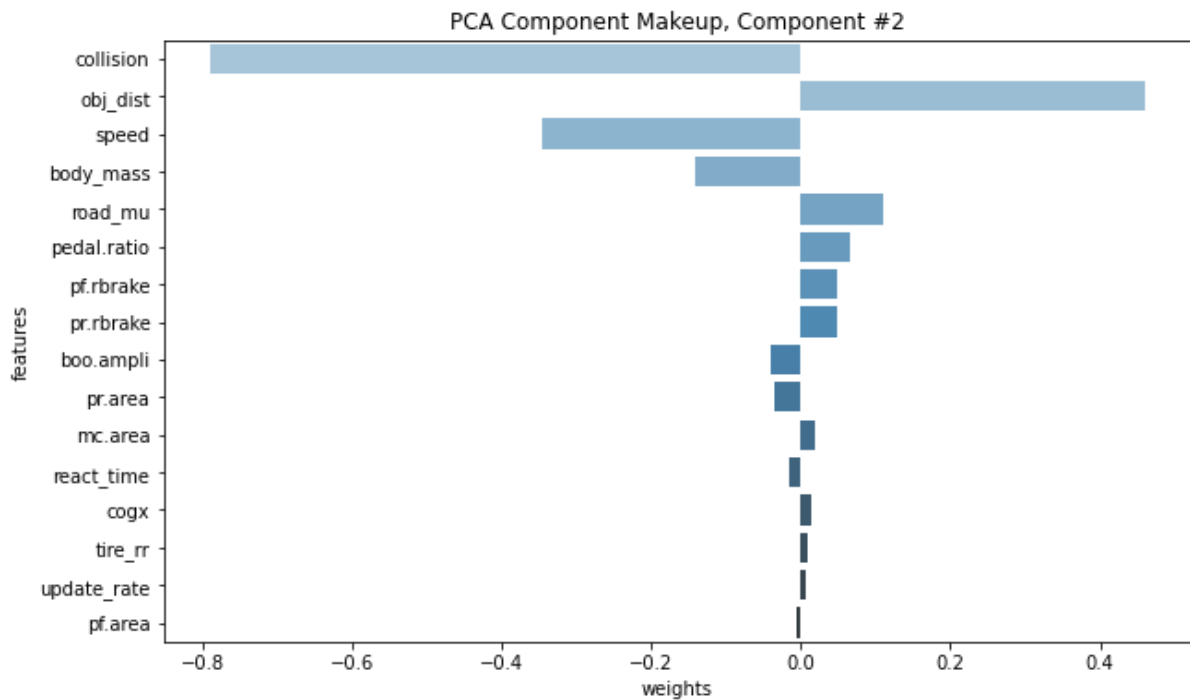


Fig: Principal component #2 feature composition

It is clear that 'obj_dist' (distance between the vehicle and the traffic object) has influence on whether a collision will occur or not - as 'obj_dist' increases the probability of collision decreases which is visible by the opposite weight signs. Also, we can see if the speed decreases, the probability of collision also decreases.

Body mass is the next vehicle parameters which has high influence on collision probability. Road friction coefficient is an environment parameter after 'obj_dist' which has a strong correlation with collision. Out of all the Brake system parameters, Pedal.ratio has the highest influence on collision probability. Interesting to note that front piston area has a much lesser effect compared to rear piston area

Braking reaction times and the centre of gravity of the vehicle have very low effect. Tire rolling resistance and the update rate of the sensor have negligible effect on the collision probability

To summarize the exploratory data analysis, we have gained valuable insights and an intuitive understanding from the dataset on how the different vehicle parameters affect the output or the KPI via the different visualizations. Also, we observe no abnormalities in the data which need to be paid due attention to.

Benchmark

It has already been decided to use a neural network. However, we do not know which neural network (nn) architecture the best is to begin with. Hence, as a benchmark we use a simple

sequential nn with 2 hidden layers with 25 neurons each (few more than no. of input features) and 10 epochs as an initial estimate.

We will then evaluate the performance of this architecture and implement changes to improve the model's performance. The performance of the final model is compared with the benchmark model.

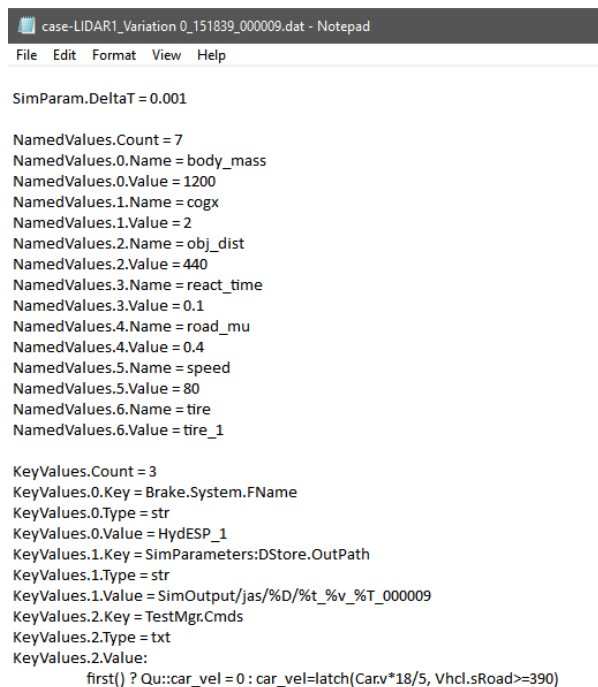
3.Methodology

Algorithm, Technique, and Implementation

Data pre-processing

The original data that we have is in a raw, unstructured format inside a zipped folder stored on the cloud. This raw dataset is first downloaded into the AWS notebook. Each simulation (variation) result stores two files: -

- .dat.info: A text file which contains the feature names and data that we need to extract



```
case-LIDAR1_Variation 0_151839_000009.dat - Notepad
File Edit Format View Help

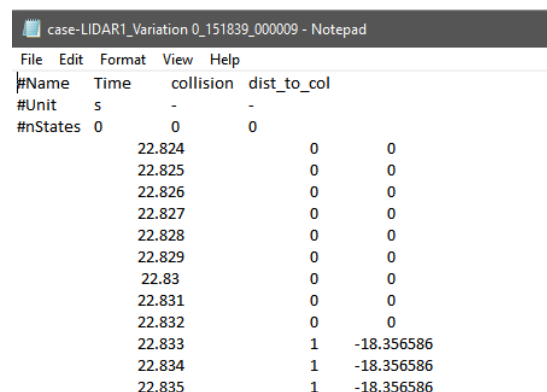
SimParam.DeltaT = 0.001

NamedValues.Count = 7
NamedValues.0.Name = body_mass
NamedValues.0.Value = 1200
NamedValues.1.Name = cogx
NamedValues.1.Value = 2
NamedValues.2.Name = obj_dist
NamedValues.2.Value = 440
NamedValues.3.Name = react_time
NamedValues.3.Value = 0.1
NamedValues.4.Name = road_mu
NamedValues.4.Value = 0.4
NamedValues.5.Name = speed
NamedValues.5.Value = 80
NamedValues.6.Name = tire
NamedValues.6.Value = tire_1

KeyValues.Count = 3
KeyValues.0.Key = Brake.System.FName
KeyValues.0.Type = str
KeyValues.0.Value = HydESP_1
KeyValues.1.Key = SimParameters.DStore.OutPath
KeyValues.1.Type = str
KeyValues.1.Value = SimOutput/jas/%D/%t_%v_%T_000009
KeyValues.2.Key = TestMgr.Cmds
KeyValues.2.Type = txt
KeyValues.2.Value:
    first() ? Qu::car_vel = 0 : car_vel=latch(Car.v*18/5, Vhcl.sRoad>=390)
```

Fig: Raw '.dat.info' data file with feature names and values to be extracted

- .dat: An ascii file which contains the output values (binary for collision- Yes:1, No:0 & distance to collision logged at the end of each simulation)



```
case-LIDAR1_Variation 0_151839_000009 - Notepad
File Edit Format View Help

#Name    Time    collision  dist_to_col
#Unit     s        -          -
#nStates  0         0          0

22.824           0          0
22.825           0          0
22.826           0          0
22.827           0          0
22.828           0          0
22.829           0          0
22.83           0          0
22.831           0          0
22.832           0          0
22.833           1 -18.356586
22.834           1 -18.356586
22.835           1 -18.356586
```

Fig: Raw '.dat' data file with output names and values to be extracted

Reading data files

To read the file names and sort them in a systematic order, 'glob' and 'natsort' libraries have been used in a custom defined 'read_org_data()' method

The read filenames are stored in two separate lists- one containing the '.dat.info' and one containing the '.dat' filenames along with the local file storage path. This method is applied on all the three data source folders. However, the filenames are not read in the desired chronological order. This is important because each simulation result has two corresponding files, and we want them arranged in the same order in both the file lists which is done using the 'natsort' library method.

```
['data/case-LIDAR-1/case-LIDAR1_Variation 32399_012801_032408.dat',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32398_012800_032407.dat',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32397_012759_032406.dat',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32396_012758_032405.dat',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32395_012757_032404.dat',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32394_012756_032403.dat',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32393_012755_032402.dat',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32392_012754_032401.dat',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32391_012753_032400.dat',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32390_012752_032399.dat']
```

Fig: Organized .dat file path and names

```
['data/case-LIDAR-1/case-LIDAR1_Variation 32399_012801_032408.dat.info',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32398_012800_032407.dat.info',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32397_012759_032406.dat.info',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32396_012758_032405.dat.info',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32395_012757_032404.dat.info',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32394_012756_032403.dat.info',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32393_012755_032402.dat.info',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32392_012754_032401.dat.info',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32391_012753_032400.dat.info',  
'data/case-LIDAR-1/case-LIDAR1_Variation 32390_012752_032399.dat.info']
```

Fig: Organized .dat.info file path and names

Extract required data

Then, the required data (feature and output data) is extracted from each individual file with the help of the 'StringIO' module and then placed into a 'pandas' dataframe with the help of the custom built method called 'get_data()'. This method reads the files using the file list created before, extracts targeted text information into a python list, converts it into a pandas series and then appends it to a pandas dataframe.

Some feature values are names of text files carried over from the simulation data. These string value are replaced with the corresponding numeric values inside the same 'get_data()' method.

Error debugging

For the first data folder, using this 'get_data()' method returned a 'list index out of range' error message, indicating one or more files are do not have the intended content format. A short script

has been created to identify the rogue file(s). This is done because it is a laborious process to spot corrupt files in a folder of 64,800 files. Once the rogue file(s) has been identified, it is deleted and the 'get_data()' method is run again successfully.

Finally, we obtain 3 dataframes corresponding to each data folder (where all result files belong to the same sensor configuration)

Special feature transformations

All the 3 dataframes are concatenated into one dataframe so it can be converted into a single CSV file for training and testing with the ML model. We make sure the dataframe data is shuffled well before we go further to avoid, specific classes of data only grouped together in the test dataset.

One of the columns, 'obj_dist' contains values of the traffic object in the global coordinates of the virtual test environment. However, we prefer this distance to be measured w.r.t the ego vehicle. Hence, we perform a single mathematical operation on the entire 'obj_dist' column to do this.

	body_mass	cogx	obj_dist	react_time	road_mu	speed	tire_rr	pedal_ratio	boo.ampli	mc.area	pf.area	pf.brake	pr.area	pr.brake	update_rate	colli:
0	2550.0	2.0	140.0	0.2	0.4	140.0	0.030	1.5	6	3.5	30	0.07	14	0.07	60	
1	2550.0	2.4	60.0	0.4	0.7	140.0	0.015	2.5	5	4.5	15	0.10	15	0.10	60	
2	1200.0	2.4	60.0	0.4	0.7	120.0	0.008	4.0	3	6.0	12	0.14	8	0.14	10	
3	3000.0	2.8	100.0	0.2	1.0	120.0	0.030	3.0	5	4.5	23	0.10	11	0.10	60	
4	2100.0	2.8	40.0	0.1	0.4	100.0	0.015	3.0	5	4.5	23	0.10	11	0.10	40	
...
97197	1200.0	2.4	100.0	0.1	0.4	80.0	0.008	1.5	6	3.5	30	0.07	14	0.07	40	
97198	2550.0	2.4	40.0	0.1	0.4	100.0	0.030	2.5	5	4.5	15	0.10	15	0.10	40	
97199	3000.0	2.4	60.0	0.2	1.0	120.0	0.030	2.5	5	4.5	15	0.10	15	0.10	10	
97200	2100.0	2.4	100.0	0.4	0.7	80.0	0.030	4.0	3	6.0	12	0.14	8	0.14	40	
97201	3000.0	2.0	140.0	0.1	0.4	120.0	0.008	4.0	3	6.0	12	0.14	8	0.14	60	

97202 rows × 17 columns

Fig: A snapshot of the final dataframe containing all features and outputs data

In order to store the dataframe for later use the 'pickle' library has been used.

In order to visualize the data, the 'plotly' library has been used over the standard 'matplotlib' library since it provides with a better plotting experience (but requires larger storage space and is graphics-heavy). The visual results have been discussed in a previous section already.

Splitting to train and test set

The final dataframe is split into two sets: train and test. Given the size of the dataset, it is decided to do a 90%-10% train-test split.

	body_mass	cogx	obj_dist	react_time	road_mu	speed	tire_rr	pedal.ratio	boo.ampli	mc.area	pf.area	pf.brake	pr.area	pr.brake	update_rate	colli
0	2550.0	2.0	140.0	0.2	0.4	140.0	0.030	1.5	6	3.5	30	0.07	14	0.07	60	
1	2550.0	2.4	60.0	0.4	0.7	140.0	0.015	2.5	5	4.5	15	0.10	15	0.10	60	
2	1200.0	2.4	60.0	0.4	0.7	120.0	0.008	4.0	3	6.0	12	0.14	8	0.14	10	
3	3000.0	2.8	100.0	0.2	1.0	120.0	0.030	3.0	5	4.5	23	0.10	11	0.10	60	
4	2100.0	2.8	40.0	0.1	0.4	100.0	0.015	3.0	5	4.5	23	0.10	11	0.10	40	
...
87475	1200.0	2.4	100.0	0.1	0.4	80.0	0.008	1.5	6	3.5	30	0.07	14	0.07	40	
87476	2550.0	2.4	40.0	0.1	0.4	100.0	0.030	2.5	5	4.5	15	0.10	15	0.10	40	
87477	3000.0	2.4	60.0	0.2	1.0	120.0	0.030	2.5	5	4.5	15	0.10	15	0.10	10	
87478	2100.0	2.4	100.0	0.4	0.7	80.0	0.030	4.0	3	6.0	12	0.14	8	0.14	40	
87479	3000.0	2.0	140.0	0.1	0.4	120.0	0.008	4.0	3	6.0	12	0.14	8	0.14	60	

87480 rows × 17 columns

Fig: Train data split (90%)

	body_mass	cogx	obj_dist	react_time	road_mu	speed	tire_rr	pedal.ratio	boo.ampli	mc.area	pf.area	pf.brake	pr.area	pr.brake	update_rate	colli
0	2100.0	2.4	60.0	0.4	0.4	80.0	0.008	2.5	5	4.5	15	0.10	15	0.10	60	
1	2550.0	2.0	60.0	0.2	0.7	100.0	0.015	2.5	5	4.5	15	0.10	15	0.10	10	
2	1200.0	2.0	40.0	0.1	0.4	100.0	0.008	4.0	3	6.0	12	0.14	8	0.14	60	
3	2550.0	2.4	100.0	0.2	0.4	120.0	0.030	3.0	5	3.5	30	0.10	14	0.10	60	
4	2100.0	2.4	100.0	0.2	0.7	140.0	0.015	1.5	6	3.5	30	0.07	14	0.07	60	
...
9717	1200.0	2.0	140.0	0.4	0.4	80.0	0.015	3.0	5	4.5	23	0.10	11	0.10	10	
9718	1650.0	2.8	40.0	0.2	1.0	140.0	0.015	4.0	3	6.0	12	0.14	8	0.14	60	
9719	2100.0	2.4	60.0	0.1	0.4	100.0	0.030	3.0	5	3.5	30	0.10	14	0.10	10	
9720	2100.0	2.0	140.0	0.4	0.4	80.0	0.008	2.5	5	4.5	15	0.10	15	0.10	60	
9721	2550.0	2.0	100.0	0.1	1.0	80.0	0.008	3.0	5	4.5	23	0.10	11	0.10	60	

9722 rows × 17 columns

Fig: Test data split (10%)

PCA

PCA is also part of the data analysis. Before feeding data into the PCA algorithm the 'MinMaxScaler' class from SKLearn pre-processing library is used to scale and transform it. This is done so that the feature values can be compared consistently which makes it also easier to plot them on a relative relational scale.

	body_mass	cogx	obj_dist	react_time	road_mu	speed	tire_rr	pedal.ratio	boo.ampli	mc.area	pf.area	pf.brake	pr.area	pr.brake	update_rate	colli
0	0.75	0.0	1.0	0.333333	0.0	1.000000	1.000000	0.0	1.000000	0.0	1.000000	0.000000	0.857143	0.000000		↑
1	0.75	0.5	0.2	1.000000	0.5	1.000000	0.318182	0.4	0.666667	0.4	0.166667	0.428571	1.000000	0.428571		↑
2	0.00	0.5	0.2	1.000000	0.5	0.666667	0.000000	1.0	0.000000	1.0	0.000000	1.000000	0.000000	1.000000		(
3	1.00	1.0	0.6	0.333333	1.0	0.666667	1.000000	0.6	0.666667	0.4	0.611111	0.428571	0.428571	0.428571		↑
4	0.50	1.0	0.0	0.000000	0.0	0.333333	0.318182	0.6	0.666667	0.4	0.611111	0.428571	0.428571	0.428571		(
...
97197	0.00	0.5	0.6	0.000000	0.0	0.000000	0.000000	0.0	1.000000	0.0	1.000000	0.000000	0.857143	0.000000		(
97198	0.75	0.5	0.0	0.000000	0.0	0.333333	1.000000	0.4	0.666667	0.4	0.166667	0.428571	1.000000	0.428571		(
97199	1.00	0.5	0.2	0.333333	1.0	0.666667	1.000000	0.4	0.666667	0.4	0.166667	0.428571	1.000000	0.428571		(
97200	0.50	0.5	0.6	1.000000	0.5	0.000000	1.000000	1.0	0.000000	1.0	0.000000	1.000000	0.000000	1.000000		(
97201	1.00	0.0	1.0	0.000000	0.0	0.666667	0.000000	1.0	0.000000	1.0	0.000000	1.000000	0.000000	1.000000		↑

97202 rows × 16 columns

Fig: Scaled data for Principal Component Analysis

The PCA model itself is built using a built-in SageMaker estimator. The 'MXNet' is used to load the model artifacts for use/deployment. To display the principal components, the 'matplotlib' and 'seaborn' libs have been used.

Model training

There were options to use simple lesser compute heavy regression algorithms: linear, multiple linear, non-linear (SVR, Decision Tree, Random Forest) for example. However, a sequential neural network model with automatic backpropagation has been chosen to do the job. Based on prior experience, compared to the mentioned algorithms, neural networks are better able to learn system dynamics and is more robust if the hyperparameters are optimized well. Although they tend to be computationally more demanding.

To implement the neural network the 'keras' library from TensorFlow has been used. The script to build, compile, train and save the model has been done in a separate python file named 'train.py'. This python file is called by the SageMaker estimator during the training process. In order to minimize the cost function a stochastic gradient descent method 'adam' optimization has been used. Since, we this is regression challenge, we use 'mean squared error' loss function to measure the loss in each epoch.

Benchmark model architecture: To begin with we start with 2 hidden neuron layers and 10 epochs. The number of neurons is taken as 25 (making sure it is greater than the number of input nodes) and the model is tested.

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	multiple	400
dense_1 (Dense)	multiple	650
dense_2 (Dense)	multiple	26

```
Total params: 1,076  
Trainable params: 1,076  
Non-trainable params: 0
```

Fig: Benchmark ANN saved model summary

Model evaluation and validation (benchmark model)

Benchmark model performance on test data

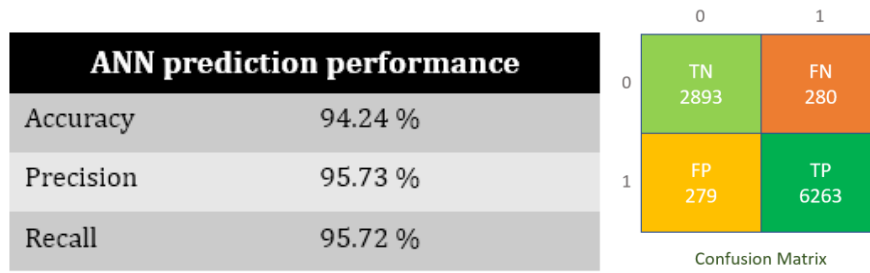


Fig: Benchmark model performance and confusion matrix for test data

Benchmark model performance on validation data

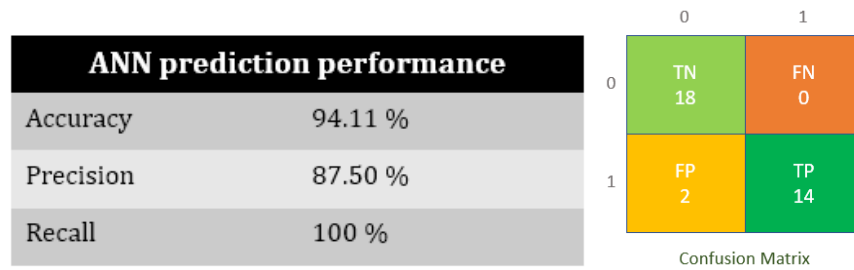


Fig: Benchmark model performance and confusion matrix for validation data

Next, we also check the aggregated accuracy for the 'distance to collision' predictions for the validation data set. As also mentioned earlier, the equation for it is as follows:

$$Agg. Accuracy = \frac{\sum |predicted distance|}{\sum |validation distance|}$$

The aggregate accuracy is 138.85%. This implies an aggregate error margin of 38.85%. The evaluation numbers indicate the performance of the benchmark model not as desired and that there is scope for improvement. This is when we refine the benchmark model.

Refinement

The benchmark model performance suggests that the neural network does not capture the dynamics as expected. Hence, we introduce two new hidden layers into the architecture. Also, we will increase the number of epochs to 200 to allow the nn to reduce the loss much further and will also reduce the batch size from 64 to 32. This will increase the training job time but should improve the model's performance.

The refined model's summary looks as follows:

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	multiple	400
dense_1 (Dense)	multiple	650
dense_2 (Dense)	multiple	650
dense_3 (Dense)	multiple	650
dense_4 (Dense)	multiple	26
Total params: 2,376		
Trainable params: 2,376		
Non-trainable params: 0		

Fig: Refined ANN saved model summary

Model evaluation and validation (refined model)

Refined model performance on test data

ANN prediction performance		0	1
Accuracy	98.62 %	TN 3083	FN 45
Precision	98.64 %	FP 89	TP 6498
Recall	99.31 %		

Confusion Matrix

Fig: Refined model performance and confusion matrix for test data

Refined model performance on validation data

ANN prediction performance		0	1
Accuracy	97.05 %	TN 20	FN 1
Precision	100 %	FP 0	TP 13
Recall	92.85 %		

Confusion Matrix

Fig: Refined model performance and confusion matrix for validation data

Next, we also check the aggregated accuracy for the 'distance to collision' predictions for the validation data set. As also mentioned earlier, the equation for it is as follows:

$$Agg. Accuracy = \frac{\sum |predicted distance|}{\sum |validation distance|}$$

The aggregate accuracy falls at 105.71 %. This implies an aggregate error margin of 5.71% only. As the numbers suggest, the refined model's performance has drastically improved with accuracy, precision and recall all showing promising results for both the test and validation data. 1 out of 34 collision predictions is wrong for the validation data.

An interesting observation is that though the error in prediction has drastically reduced (in other words improved), the highest error occurs for features where vehicle speed falls outside the range of the training feature set. Where the prediction is wrong in this case, the vehicle speed was 150kph, whereas the training speed range was between 80kph - 140kph only!

At this point, we create a new set of test variations (~1300) with higher vehicle speeds and different variations of brake parameters and vehicle centre of gravity heights. Then this small dataset will be added to the original dataset and the model will be retrained. Then we will check the accuracy of the model again.

Update training dataset

The following parameters (features) have been considered and all combinations have been used to run a corresponding simulation test to create the updated training dataset:

Vehicle params			Tire	Brakes	Road	Driver	Traffic	Sensor
speed	body_mass	CoG_x	Tire RR	Brake file	road_mu	react_time	object distance	update_rate
130	1700	2.4	0.015	HydESP_1	0.4	0.1	100	40
140	2000	2.6		HydESP_test	0.7		120	60
160	2500	3			1		140	
180								

Fig: New data parameter variations

The parameters for the new 'HydESP_test' file are as follows:

Hydraulic Brake file	Brake parameter variations [pedal.ratio, boo.ampli, mc.area, pf.area, pf.rbrake, pr.area, pr.rbrake]
HydESP_test	[3.5, 5.5, 3.5, 25, 0.12, 14, 0.12]

With this we obtain a total of 1296 new variations with new parameter ranges, and we merge this data with the original dataset using all the data pre-processing steps discussed before

Retraining the model on updated dataset

The aforementioned refined model is now re-trained on the new dataset and is evaluated. To summarize the model architecture looks as below:

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	multiple	400
dense_1 (Dense)	multiple	650
dense_2 (Dense)	multiple	650
dense_3 (Dense)	multiple	650
dense_4 (Dense)	multiple	26
Total params: 2,376		
Trainable params: 2,376		
Non-trainable params: 0		

Fig: Refined ANN model summary for the updated dataset

Model evaluation and validation (refined model with updated dataset)

Refined model performance on test data from the updated dataset

ANN prediction performance		0	1
Accuracy	98.66 %	TN 1607	FN 58
Precision	99.75 %	FP 8	TP 3276
Recall	98.26 %		

Confusion Matrix

Fig: Refined model performance and confusion matrix for test data (updated dataset)

Refined model performance on test data from the updated dataset

ANN prediction performance		0	1
Accuracy	100 %	TN 20	FN 0
Precision	100 %	FP 0	TP 14
Recall	100 %		

Confusion Matrix

Fig: Refined model performance and confusion matrix for validation data (updated dataset)

Next, we also check the aggregated accuracy for the 'distance to collision' predictions for the validation data set. As also mentioned earlier, the equation for it is as follows:

$$Agg. Accuracy = \frac{\sum |predicted distance|}{\sum |validation distance|}$$

The aggregate accuracy now stands closer at 96.88 %. This implies an aggregate error margin of 3.12% only. As the numbers suggest, the refined model works much better compared with the benchmark model, especially now when we have introduced a new targeted training data. All collision predictions for the validation data are correct and the error in predicting distance to collision is desirably low. This is the final trained model.

4. Results

Justification

Our goal was not only to be able to predict collision possibility or the distance to collision for new vehicle models, but it was also to use the trained neural network to recommend feature values (vehicle parameters) such that a collision does not occur in the given scenario.

We have done this for three sample features, and we compare a range of feature predictions and compare it with simulation data to determine how well the trained model performs here. Some features are linearly related to the output variable (distance to collision) while some are non-linearly related, and we will see examples of both.

Considered case: The vehicle (Demo Audi R8) is cruising at a speed of 150 kph on a road surface with friction coefficient 0.7

Feature 1: 'obj_dist' (distance of traffic object (wild boar) in-front of vehicle when it suddenly appears on the road)

Task: Predict traffic object distance at which collision does not occur in the considered case with remaining features unchanged.

The plot below shows the predicted and simulation data (referenced here as ground truth) and we can see that the correlation is incredibly good and satisfactory. The algorithm predicts a collision would not occur if the object distance @150 kph vehicle speed was 113m and beyond. The simulation shows an actual collision when that distance is 115m. An error of 1.73%

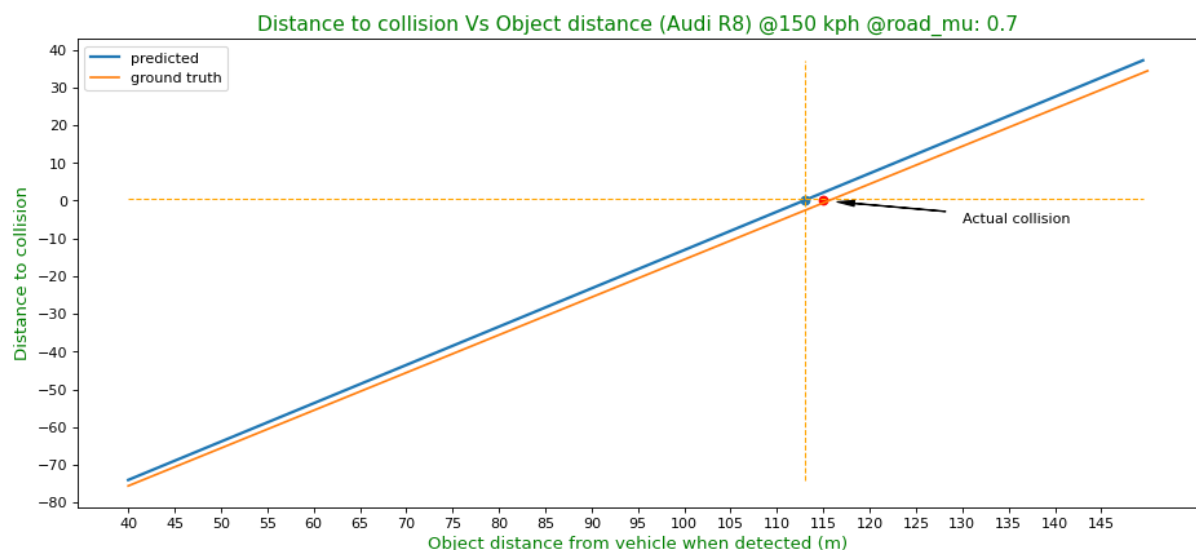


Fig: Feature value recommendation to avoid collision – 'obj_dist'

Feature 2: 'pf.area' (Brake front piston area (cm²))

Task: Predict front brake piston area for which collision does not occur in the considered case with remaining features unchanged.

The algorithm predicts a collision would not occur if the front brake piston area was kept at 17.99 cm² or more. The simulation shows a collision just occurring for a front brake piston value of 15.5 cm² already. In this case, the error rate is undesirably high at 16%. However, this is expected because the dataset covers only a short range of brake parameters in the training data.

This means, for a better recommendation accuracy here, we will require more data points covering a broader range of brake parameters. Interestingly, the shape of the two curves suggests the model has captured the brake system dynamics fairly well, which is a good sign.

As we can observe from the plot below, feature-output relation is very non-linear.

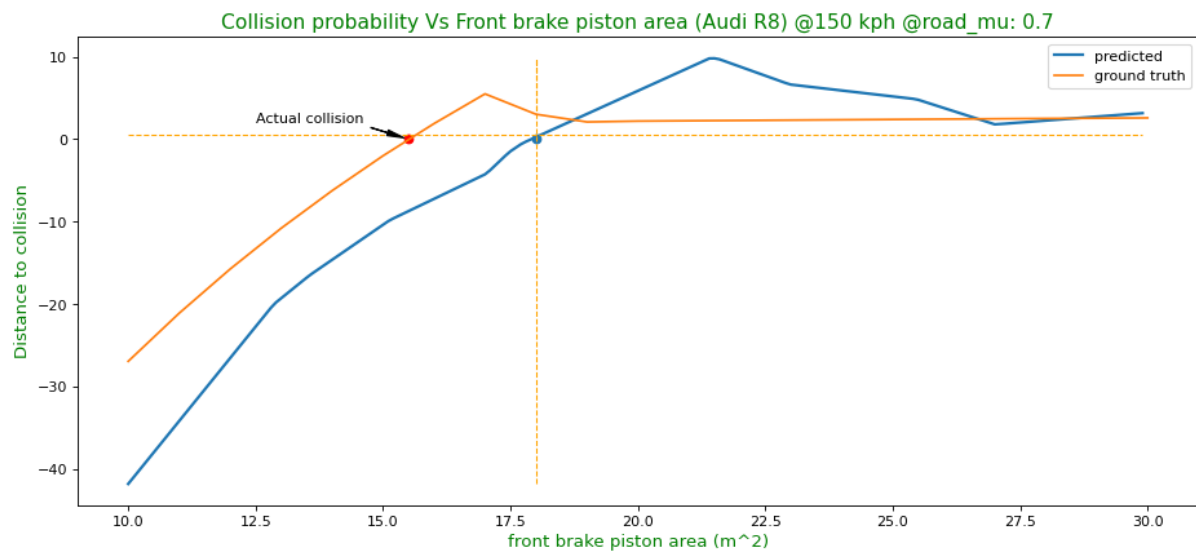


Fig: Feature value recommendation to avoid collision – 'pf.area'

Feature 3: 'road_mu' (Road Coefficient of friction)

Task: Predict road friction coefficient for which collision does not occur in the considered case with remaining features unchanged.

Here, the algorithm predicts a collision would not occur if the road friction coefficient were 0.61 or higher. The simulation suggests the exact same value for the collision just occurs at the very same road_mu value. The error rate in this feature case is 0% which is remarkable.

As seen from the plots below the feature-output relation this time is also very non-linear, and the model performs very well. This can be partially credited to the broad range of 'road-mu' covered in the training data.

It is worth noting that below road_mu: 0.4 (outside the training data range) the prediction accuracy starts deviating heavily. To capture system's behaviour below road_mu:0.4, we will require to train the model on new data covering that feature range.

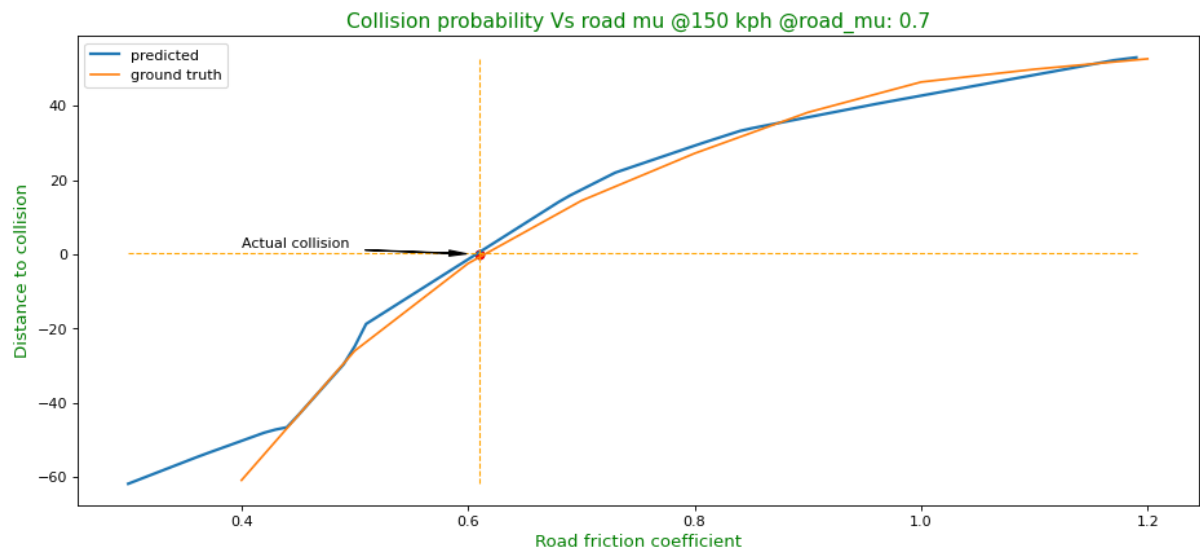


Fig: Feature value recommendation to avoid collision – 'road_mu'

Conclusion

This machine learning project has successfully implemented the following steps

- Data collection and retrieving
- Data pre-processing (edgecase_data_processing.ipynb)
 - Cleaning and transforming
 - Data analysis and insights
 - PCA (PCA.ipynb)
- Training a neural network on input data (Training_prediction.ipynb)
 - Evaluating and refining the model
- Injecting new data and retraining the model (edgecase_newdata_processing.ipynb)
- Evaluating the final model (Visualizing_model_predictions.ipynb)
 - Accurately predict collision possibility
 - Recommend feature values to avoid collision

All development work has been conducted with AWS SageMaker. Data analysis has revealed interesting relationships between the various features and output variables. The final refined neural network model is a much-improved version of the benchmark model with remarkably high accuracy, precision and recall (the metrics) proven on test and validation data.

Improvements / Further suggested work

Even though the neural network model performs with very high accuracy, it falls behind when we dive deeper into individual feature investigation with the model. It is important to note that the current input dataset considers only 15 features or vehicle parameters. In reality, a vehicle or even a high-fidelity simulation model has several times a greater number of parameters which influence the vehicles behaviour in different magnitudes.

In order to increase the robustness of the neural network it is suggested to expand the training dataset to more parameters and cover a broader value range. To make this process more effective, PCA performed in this project can hint towards features of high interest.

The data pre-processing code can be made more robust by incorporating automatic feature extractions by name tagging. Currently, there is small manual process used to identify the location of information of interest inside the raw data files. This method would not directly work if new features were considered, for example.

Furthermore, new and complex autonomous driving scenarios can be considered. The current case considered a longitudinal case. Physics get more complicated when a vehicle wishes to perform evasive manoeuvres by steering. Also, the sensor model considered here is an ideal sensor. In reality, a high-fidelity or Raw Signal Interface (RSI) sensor models have a greater number of parameters affecting the quality of traffic object detection which is suggested to be taken into account.