USE VEHICLE NETWORK DATA TO TRAIN A ML MODEL AND PREDICT SPECIFIC CONDITIONS ON AUTOMOTIVE VEHICLES

DISSERTATION

Submitted in partial fulfillment of the requirements of the

MTech Data Science and Engineering Degree programme.

By

Saurabh Kumar

2021FC04327

Under the supervision of

Kiran Bachhav

Manager

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

Pilani (Rajasthan) INDIA

(September 2023)

Table of Contents

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

ii. Certificate

This is to certify that the Dissertation entitled USE VEHICLE NETWORK DATA TO TRAIN A ML MODEL AND PREDICT SPECIFIC CONDITIONS ON AUTOMOTIVE VEHICLES and submitted by Mr. Saurabh Kumar ID No. 2021FC04327 in partial fulfillment of the requirements of DSECLZG628T Dissertation, embodies the work done by him under my supervision.

Place: Pune	Name: Kiran Bachhav
Date: 25-Sep-2023	Designation: Manager

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI II SEMESTER 22-23

DSE CL ZG628T DISSERTATION

iii. Dissertation Final Report

BITS ID No. 2021FC04327 **Name of Student**: Saurabh Kumar

Name of Supervisor: Kiran Bachhav

Designation of Supervisor: Manager

Qualification, Experience: Executive Business Management & B.E. Electronics, 20+ years

E- mail ID of Supervisor: BachhavKiran@JohnDeere.com

Topic of Dissertation: Use vehicle network data to train a ML model and predict specific conditions on Automotive vehicles.

Name of First Examiner: C V Krishnaveni

Designation of First Examiner:

Qualification and Experience:

E- mail ID of First Examiner: krishnaveni@wilp.bits-pilani.ac.in

Name of Second Examiner:

Designation of Second Examiner:

Qualification and Experience:

E- mail ID of Second Examiner:

Saurabh Kumar
Date: 25-Sep-2023

Kiran Bachhav
Date: 25-Sep-2023

iv. ACKNOWLEDGEMENTS

I would like to thank my supervisor Kiran Bachhav for his guidance and wholehearted support. His valuable comments and feedback have been immensely helpful in enhancing the quality of work. I would like to express my sincere gratitude towards my project mates for the support and cooperation they have provided so far.

Special thanks to my examiner C V Krishnaveni for her support and guidance.

I am also thankful to John Deere & Company for giving me this opportunity and also making available all the resources required for this dissertation. I am also thankful to BITS, Pilani for their efforts in organizing this course.

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE,

PILANI SECOND SEMESTER2022-23

DSECLZG628T DISSERTATION

Dissertation Title : USE VEHICLE NETWORK DATA TO TRAIN A ML MODEL AND PREDICT SPECIFIC CONDITIONS ON AUTOMOTIVE VEHICLES.

Name of Supervisor : Kiran Bachhav

Name of Student : Saurabh Kumar

ID No. of Student : 2021FC04327

v. Abstract

Key Words: Automotive, Machine Learning, Embedded, Data Science, CAN J1939, Sensors, Control Algorithms

In Automotive vehicles, electronic controllers make use of various sensors which determines the surrounding conditions. There are also other data on the vehicle network such as calculated parameters from other controllers. These inputs obtained from sensors and controllers are used to calculate specific conditions on the machine. These conditions are derived by using Control Algorithms designed in the controllers. The goal is to use the data obtained from the sensors to train machine learning models to predict the conditions on the

machine.

For example, there are sensors on machine which help determine specific parameters. These observed parameters are used in detecting specific conditions on the machine. These specific conditions are currently determined using the control algorithms. Sensor data can be used to train a machine learning model which can predict these specific conditions. If we can predict these conditions successfully, we may be able to reduce the number of sensors on machine and thus reduce the overall cost.

The first step is to extract the data. The data comes on the vehicle network in CAN J1939 protocol. This data shall be filtered and cleaned. Then proper method should be found to convert this data to excel format. Then using the excel format the data can be divided in training and test set. Training dataset can be used to train a machine learning model and test data can be used to test the model. Both the calculated data from Control Algorithm and the learned data shall be compared to verify the accuracy of the result.

vi. List of Figures

vii. List of Tables

viii. List of Abbreviations

CAN – Controller Area Net=work

VT – Virtual Terminal

ISO - International Organization for Standardization

.asc – ASCII format

DF – Data Frame

RFC – Random Forest Classifier

LSTM – Long Short Term Memory

RNN – Recurrent Neural Network

1. Introduction

Today automotive vehicles have wide variety of functions such as transportation, harvesting, forestry, entertainment etc. These functions involve mechanical, electrical, electronics and software fields. Electronics and software together are also called Embedded programming. The automotive vehicles have electronic controllers which holds complex algorithms written in Embedded programming to do complex functions using sensor and actuators. These controllers also depend on huge amount of real-time data to process these complex control algorithms.

These data are generated from various sources. Such as Sensors, output of other electronic controllers, external devices connected etc. Also, the data is generated using different protocols such J1939, VT-ISO 11783-6 etc. therefore the type of data is different. Since this data is generated to process at real time, the nature of these data is log time-series data.

Data Science holds potential to use the data to replace with existing control algorithm and much more. Using machine learning we can train models to predict the output which can help us reduce the number of sensors and other parts. Thus, reducing the overall cost of the machines.

The goal is to use the data obtained from the sensors to train machine learning models to predict the conditions on the machine.

2. Data Description & Conversion

The vehicle network data was taken from the last year (2022) harvesting period. As mentioned earlier, the data is in CAN J1939 protocol in the log format with .asc extension. This protocol defines data in hexadecimal system.

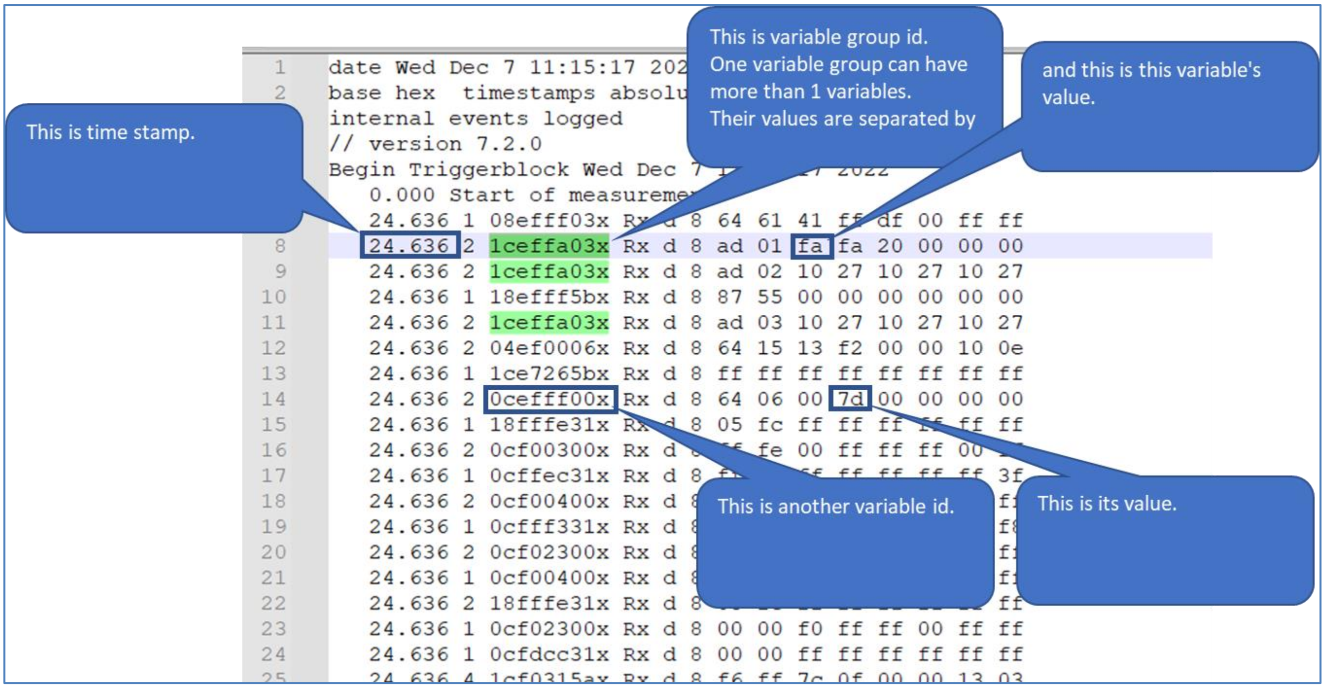

Figure 1Sample CAN J1939 log data

There are 50 such CAN logs in .asc format equal to 3.26GB. Hence this data is required to convert to decimal format in column format. A Deere & Company internal tool was used to do this conversion. All the required features were extracted and put in the column format in .csv files. 50 .csv files were created.

	A	B	C	D	E	F	G	H	I	J	K
1	Time (s)	Switch1	Switch2	Speed1	State2	Speed2	State1	State3	State4		
2	1923.136	1	1	1901.75	2	7.421875	1	1229	0		
3	1923.136	1	1	1901.75	2	7.421875	1	1229	0		
4	1923.136	1	1	904.25	11	7.421875	1	1229	0		
5	1923.136	1	1	904.25	11	7.421875	1	1229	0		
6	1923.146	1	1	904.25	11	7.421875	1	0	0		
7	1923.146	1	1	903.75	11	7.421875	1	0	0		
8	1923.146	1	1	903.75	11	7.421875	1	0	0		
9	1923.156	1	1	903.5	11	7.421875	1	0	0		
10	1923.156	1	1	903.5	11	7.421875	1	0	0		
11	1923.166	1	1	903.5	11	0	1	0	0		
12	1923.166	1	1	903.5	11	0	1	0	0		
13	1923.166	1	1	903	11	0	1	0	0		
14	1923.166	1	1	903	11	0	1	0	0		
15	1923.177	1	1	902.25	11	0	1	0	0		
16	1923.177	1	1	902.25	11	0	1	0	0		
17	1923.188	1	1	901.5	11	0	1	0	0		
18	1923.188	1	1	901.5	11	0	1	0	0		
19	1923.193	0	0	901.5	11	0	1	0	0		
20	1923.198	0	0	901.5	11	0	1	0	0		
21	1923.198	0	0	901.5	11	0	1	0	0		
22	1923.208	0	0	901.25	11	0	1	0	0		
23	1923.208	0	0	901.25	11	0	1	0	0		
24	1923.217	0	0	900.75	11	0	1	0	0		

Figure 2Converted .csv data

The 50 .csv files were read using Pandas library and combined into a single Data Frame.

Importing Libraries

Import CAN csv

```
In [1]: #Basic libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import glob    #to read csv files from directory
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split


#For accuracy, precision, recall and fscore metrics
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import mean_squared_error
import math
#To scale the values for lstm model
from sklearn.preprocessing import MinMaxScaler
#import sklearn.ensemble.HistGradientBoostingClassifier

import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM,Flatten
from keras.layers import ConvLSTM2D


#to save model
import pickle
```

Figure 3 Importing Basic Libraries

Multiple csv files are read as Dataframes using Pandas, then all the Dataframes are merged into single combined Dataframe.

This is how the dataframe looks like.

```
In [4]: # Print the combined dataframe
        print(df)
```

```
            Time  State1  State2   Speed1  State4    Speed2  State3  Speed5  \
0       1923.136       1       1  1901.75       2  7.421875       1    4240
1       1923.136       1       1  1901.75       2  7.421875       1    4240
2       1923.136       1       1   904.25      11  7.421875       1    4240
3       1923.136       1       1   904.25      11  7.421875       1    4240
4       1923.146       1       1   904.25      11  7.421875       1    4240
...          ...     ...     ...      ...     ...       ...     ...     ...
118001  5369.861       1       1  1895.25       1  6.195312       3    4180
118002  5369.866       1       1  1895.25       1  6.195312       3    4180
118003  5369.866       1       1  1899.25       3  6.195312       3    4180
118004  5369.874       1       1  1899.25       3  6.195312       3    4180
118005  5369.881       1       1  1900.50       3  6.195312       3    4180

        State5  Speed6  ...  BoolStateC3  BoolStateA4  BoolStateB4  \
0            3     184  ...          NaN          NaN          NaN
1            3     184  ...          NaN          NaN          NaN
2            3     184  ...          NaN          NaN          NaN
3            3     184  ...          NaN          NaN          NaN
4            3       0  ...          NaN          NaN          NaN
...        ...     ...  ...          ...          ...          ...
118001       3     164  ...          0.0          0.0          0.0
118002       3     164  ...          0.0          0.0          0.0
118003       3     164  ...          0.0          0.0          0.0
118004       3     164  ...          0.0          0.0          0.0
118005       3     164  ...          0.0          0.0          0.0

        BoolStateC4  BoolStateA5  BoolStateB5  BoolStateC5  BoolStateA6  \
0               NaN          NaN          NaN          NaN          NaN
1               NaN          NaN          NaN          NaN          NaN
2               NaN          NaN          NaN          NaN          NaN
3               NaN          NaN          NaN          NaN          NaN
4               NaN          NaN          NaN          NaN          NaN
...             ...          ...          ...          ...          ...
118001          0.0          0.0          0.0          0.0          0.0
118002          0.0          0.0          0.0          0.0          0.0
```

Figure 4 Dataframe

3. Exploratory Data Analysis

Using EDA, the type of data is observed, missing values, relationship between variables is observed.

```
In [5]: df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1148064 entries, 0 to 118005
Data columns (total 45 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   Time         1148064 non-null  float64
 1   State1       1148064 non-null  int64
 2   State2       1148064 non-null  int64
 3   Speed1       1148064 non-null  float64
 4   State4       1148064 non-null  int64
 5   Speed2       1148064 non-null  float64
 6   State3       1148064 non-null  int64
 7   Speed5       1148064 non-null  int64
 8   State5       1148064 non-null  int64
 9   Speed6       1148064 non-null  int64
 10  Speed7       1148064 non-null  int64
 11  Speed3       1148064 non-null  int64
 12  Speed4       1148064 non-null  int64
 13  Speed8       555210 non-null   float64
 14  SpeedA1      555210 non-null   float64
 15  SpeedA2      555210 non-null   float64
 16  SpeedA3      555210 non-null   float64
 17  SpeedA4      555210 non-null   float64
 18  SpeedA5      555210 non-null   float64
 19  SpeedA6      555210 non-null   float64
 20  SpeedB1      555210 non-null   float64
 21  SpeedB2      555210 non-null   float64
 22  SpeedB3      555210 non-null   float64
 23  SpeedB4      555210 non-null   float64
 24  SpeedB5      555210 non-null   float64
 25  SpeedB6      555210 non-null   float64
 26  Mode         555210 non-null   float64
 27  BoolStateA1  1057180 non-null  float64
 28  BoolStateB1  1057180 non-null  float64
 29  BoolStateC1  1057180 non-null  float64
```

Figure 5 Data Info

Data Cleaning

Unnecessary columns are omitted from the data.

```
In [ ]: # Drop unnecessary columns
        df=df.drop(['Time','ClnrDriveSpeed'],axis=1)
```

Figure 6 Data Cleaning

For missing values, there are 2 cases:

- Intermediate missing values are replaced with previous values. This is due to nature of CAN network, the data is sometimes missed on the network due to heavy traffic or other conditions.
- Top missing values are replaced with 0 as all the data taken here has initial value of 0.

Clean Data

```
In [8]: #Handle Missing values
        #Replace missing values with previous values
        #Replace top missing values with 0

        df['State1'] = df['State1'].fillna(method='ffill')
        df['State1'] = df['State1'].fillna(value=0)
        df['State2'] = df['State2'].fillna(method='ffill')
        df['State2'] = df['State2'].fillna(value=0)
        df['Speed1'] = df['Speed1'].fillna(method='ffill')
        df['Speed1'] = df['Speed1'].fillna(value=0)
        df['State3'] = df['State3'].fillna(method='ffill')
        df['State3'] = df['State3'].fillna(value=0)
        df['Speed2'] = df['Speed2'].fillna(method='ffill')
        df['Speed2'] = df['Speed2'].fillna(value=0)
        df['State3'] = df['State3'].fillna(method='ffill')
        df['State3'] = df['State3'].fillna(value=0)
        df['Speed5'] = df['Speed5'].fillna(method='ffill')
        df['Speed5'] = df['Speed5'].fillna(value=0)
        df['State5'] = df['State5'].fillna(method='ffill')
        df['State5'] = df['State5'].fillna(value=0)
        df['Speed6'] = df['Speed6'].fillna(method='ffill')
        df['Speed6'] = df['Speed6'].fillna(value=0)
        df['Speed7'] = df['Speed7'].fillna(method='ffill')
        df['Speed7'] = df['Speed7'].fillna(value=0)
        df['Speed3'] = df['Speed3'].fillna(method='ffill')
        df['Speed3'] = df['Speed3'].fillna(value=0)
        df['Speed4'] = df['Speed4'].fillna(method='ffill')
        df['Speed4'] = df['Speed4'].fillna(value=0)
        df['Speed8'] = df['Speed8'].fillna(method='ffill')
        df['Speed8'] = df['Speed8'].fillna(value=0)
        df['Mode'] = df['Mode'].fillna(method='ffill')
        df['Mode'] = df['Mode'].fillna(value=0)
        df['SpeedA1'] = df['SpeedA1'].fillna(value=0)
        df['SpeedA2'] = df['SpeedA2'].fillna(value=0)
        df['SpeedA3'] = df['SpeedA3'].fillna(value=0)
```

Figure 7 Data Cleaning 2

4. Model Selection

This is a problem binary classification. As we have to predict whether the specific condition occurred in the data or not.

We also want to train the model to detect the condition, hence Supervised learning has been implemented with the available data.

Second part of the problem is to predict the specific condition without the directly dependent inputs. These identified inputs are A1, A2, A3, A4, A5, A6, B1, B2, B3, B4, B5 and B6. Hence these will be removed from the data and another set of models will be trained with this data to predict the specific condition.

Here 2 models are considered:

- Random Forest Classifier
- LSTM

And their performance is compared with and without the directly dependent inputs A1 to A6 and B1 to B6. So, four models' were trained and performance compared:

- Random Forest Classifier with AB inputs
- Random Forest Classifier without AB inputs
- LSTM with AB inputs
- LSTM without AB inputs

Random Forest Classifier with directly dependent AB inputs

First the data was taken to train with a basic model. Random Forest Classifier was taken for this purpose. All the 26 inputs and 1 output were taken. This data was divided into training and test dataset with 0.2 test dataset. The model was trained with 50 estimators and max depth of 6. The model only predicted 0(not occurred) in the data. The model could not predict when the specific condition occurred in the data.

The number of estimators was changed to 100 and another model was trained, but that also could not predict when the specific condition occurred in the data.

	A	B	C	D	E
1		Y_Actual	Y_Predicted	Y_Predicted sum =	0
9163	9161	1	0		
9164	9162	1	0		
9165	9163	1	0		
9166	9164	1	0		
9167	9165	1	0		
9168	9166	1	0		
9169	9167	1	0		
9170	9168	1	0		
9171	9169	1	0		
9172	9170	1	0		
9173	9171	1	0		
9174	9172	1	0		
9175	9173	1	0		
9176	9174	1	0		
9177	9175	1	0		
9178	9176	1	0		
9179	9177	1	0		
9180	9178	1	0		
9181	9179	1	0		
9182	9180	1	0		
9183	9181	1	0		
9184	9182	1	0		
9185	9183	1	0		
9186	9184	1	0		
9187	9185	1	0		
9188	9186	1	0		
9189	9187	1	0		

Figure 8 RFC predition

Validation with trained dataset:

```
Accuracy: 0.9543492746490835
Precision: 0.9564332724502799
Recall: 0.9543492746490835
```

```
F1-score: 0.9320614355825253
```

Validation with untrained dataset:

```
Accuracy: 0.9956238410596027
Precision: 0.9564332724502799
Recall: 0.9543492746490835
F1-score: 0.9320614355825253
```

Random Forest Classifier WITH Speed AB Inputs

In [15]:
```
#Train model WITH Drum speed inputs
# Split the data into training and testing sets
X = df[['State1', 'State2', 'Speed1', 'State3',
       'Speed2', 'State4', 'Speed5',
       'State5', 'Speed6', 'Speed7', 'Speed3',
       'Speed4', 'Speed8', 'SpeedA1',
       'SpeedA2', 'SpeedA3',
       'SpeedA4', 'SpeedA5',
       'SpeedA6', 'SpeedB1',
       'SpeedB2', 'SpeedB3',
       'SpeedB4', 'SpeedB5',
       'SpeedB6', 'Mode']].values
y = df['BoolState1'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [18]:
```
type(X_test)
```

Out[18]:
```
numpy.ndarray
```

In [16]:
```
# Train the random forest model WITH Speed AB inputs
rf_model_WITH_SpeedAB = RandomForestClassifier(n_estimators=50, max_depth=6)
rf_model_WITH_SpeedAB.fit(X_train, y_train)

#Save model
# save the RandomForestClassifier model as a pickle file
model_pkl_file_WITH_SpeedAB = "RrandomForestClassifier_model_WITH_SpeedABInputs.pkl"
with open(model_pkl_file_WITH_SpeedAB, 'wb') as file:
    pickle.dump(rf_model_WITH_SpeedAB, file)
```

Figure 9 RFC with AB inputs

Random Forest Classifier without directly dependent AB inputs

Then the directly dependent inputs which are A1 to A6 and B1 to B6 were reduced from the input data and given to the RFC model to train. Total inputs were 14 and output 1. This data was divided into training and test dataset with 0.2 test dataset. The model was trained with 50 estimators and max depth of 6. The model predicted only the 0(condition not occurred). The model couldn’t predict when the specific condition occurred in the data.

Validation with trained dataset:

```
Accuracy: 0.9540879654026557
Precision: 0.9102838457261793
Recall: 0.9540879654026557
F1-score: 0.9316713083984506
```

Validation with untrained dataset:

```
Accuracy: 0.9956238410596027
Precision: 0.9102838457261793
Recall: 0.9540879654026557
F1-score: 0.9316713083984506
```

Result same as “RFC with AB inputs”.

Random Forest Classifier WITHOUT Speed AB Inputs

In [23]:
```python
#Train model WITHOUT speed AB inputs
# Split the data into training and testing sets
X = df[['State1', 'State2', 'Speed1', 'State3',
       'Speed2', 'State4', 'Speed5',
       'State5', 'Speed6', 'Speed7', 'Speed3',
       'Speed4', 'Speed8', 'Mode']].values
y = df['BoolState1'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [24]:
```python
# Train the random forest model
rf_model_WITHOUT_SpeedAB = RandomForestClassifier(n_estimators=100, max_depth=6)
rf_model_WITHOUT_SpeedAB.fit(X_train, y_train)

#Save model
# save the RandomForestClassifier model as a pickle file
model_pkl_file_WITHOUT_SpeedAB = "RrandomForestClassifier_model_WITHOUT_SpeedAB.pkl"
with open(model_pkl_file_WITHOUT_SpeedAB, 'wb') as file:
    pickle.dump(rf_model_WITHOUT_SpeedAB, file)
```

In [25]:
```python
#Load model
model_pkl_file_WITHOUT_SpeedAB = "RrandomForestClassifier_model_WITHOUT_SpeedAB.pkl"
# load model from pickle file
with open(model_pkl_file_WITHOUT_SpeedAB, 'rb') as file:
    rf_model_WITHOUT_SpeedAB = pickle.load(file)
```

Figure 10 RFC without AB inputs

LSTM with directly dependent AB inputs

Second model considered for training was LSTM(Long Short Term Memory) recurrent neural network model. The reason for considering this model is because the dataset is time-series nature. The specific condition occurs in the data after a sequence of events. This model is known to work well with time-series data, hence this model was selected.

All the 26 inputs and 1 output were given to the LSTM model to train. 1 Dense layer was selected. And loss function binary_crossentropy was selected because we want to train the model for binary classification problem of whether the specific condition occurred or not.

Train data was prepared considering window length of 500 and 26 features with 1 hidden LSTM layer of 32 units and 1 Dense output layer of 1 neuron with ‘Softmax’ activation. 1 epoch took ~40 minutes to train the data.

The model was verified with untrained data, but the model predicted values was always 1 which was not true.

Activation was changed to ‘Sigmoid’ but the model couldn’t predict the specific condition in the data.

Validation with trained dataset:
`Loss = 0.0915, Accuracy = 0.9596`

Validation with untrained dataset:
`Loss = 0.1143, Accuracy = 0.9765`

```
In [38]: #Train model WITH Drum speed inputs
         # Split the data into training and testing sets
         X_lstm_w = df1[['State1', 'State2', 'Speed1', 'State3',
                'Speed2', 'State4', 'Speed5',
                'State5', 'Speed6', 'Speed7', 'Speed3',
                'Speed4', 'Speed8', 'SpeedA1',
                'SpeedA2', 'SpeedA3',
                'SpeedA4', 'SpeedA5',
                'SpeedA6', 'SpeedB1',
                'SpeedB2', 'SpeedB3',
                'SpeedB4', 'SpeedB5',
                'SpeedB6', 'Mode']].values
         y_lstm_w = df1['BoolState1'].values
         X_train_lstm_w, X_test_lstm_w, y_train_lstm_w, y_test_lstm_w = train_test_split(X_lstm_w, y_lstm_w, test_size=0.3)

In [39]: #Shape of inputs
         print(np.shape(X_train_lstm_w))
         print(np.shape(y_train_lstm_w))
         print(np.shape(X_test_lstm_w))
         print(np.shape(y_test_lstm_w))

         X_train_lstm_w = np.reshape(X_train_lstm_w, (-1, 26, 1))
         X_test_lstm_w = np.reshape(X_test_lstm_w, (-1, 26, 1))

         #y_test_lstm_w = np.reshape(y_test_lstm_w, (-1, 1, 1))

         print(np.shape(X_train_lstm_w))
         print(np.shape(X_test_lstm_w))
```

Figure 11 LSTM with AB inputs

```
In [40]: # Define the LSTM model
lstm_model_WITH_SpeedAB = Sequential()
lstm_model_WITH_SpeedAB.add(LSTM(32, input_shape=(26, 1)))
lstm_model_WITH_SpeedAB.add(Dense(1, activation='sigmoid'))

# Compile the model
lstm_model_WITH_SpeedAB.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
lstm_model_WITH_SpeedAB.fit(X_train_lstm_w, y_train_lstm_w, epochs=10, batch_size=32, validation_data=(X_test_lstm_w, y_test_lstr

#Save model
# save the LSTM  model as a pickle file
model_pkl_file_LSTM_WITH_SpeedAB = "LSTM_model_With_SpeedAB.pkl"
with open(model_pkl_file_LSTM_WITH_SpeedAB, 'wb') as file:
    pickle.dump(lstm_model_WITH_SpeedAB, file)
```

Figure 12 LSTM model definition

```
Epoch 1/10
25114/25114 [==============================] - 441s 17ms/step - loss: 0.1084 - accuracy: 0.9542 - val_loss: 0.1009 - val_accura
cy: 0.9561
Epoch 2/10
25114/25114 [==============================] - 437s 17ms/step - loss: 0.1003 - accuracy: 0.9562 - val_loss: 0.0978 - val_accura
cy: 0.9577
Epoch 3/10
25114/25114 [==============================] - 441s 18ms/step - loss: 0.0979 - accuracy: 0.9575 - val_loss: 0.0971 - val_accura
cy: 0.9576
Epoch 4/10
25114/25114 [==============================] - 442s 18ms/step - loss: 0.0964 - accuracy: 0.9580 - val_loss: 0.0959 - val_accura
cy: 0.9583
Epoch 5/10
25114/25114 [==============================] - 442s 18ms/step - loss: 0.0955 - accuracy: 0.9582 - val_loss: 0.0954 - val_accura
cy: 0.9584
Epoch 6/10
25114/25114 [==============================] - 444s 18ms/step - loss: 0.0946 - accuracy: 0.9586 - val_loss: 0.0936 - val_accura
cy: 0.9585
Epoch 7/10
25114/25114 [==============================] - 441s 18ms/step - loss: 0.0935 - accuracy: 0.9589 - val_loss: 0.0931 - val_accura
cy: 0.9592
Epoch 8/10
25114/25114 [==============================] - 440s 18ms/step - loss: 0.0935 - accuracy: 0.9589 - val_loss: 0.0924 - val_accura
cy: 0.9593
Epoch 9/10
25114/25114 [==============================] - 445s 18ms/step - loss: 0.0932 - accuracy: 0.9589 - val_loss: 0.0918 - val_accura
cy: 0.9597
Epoch 10/10
25114/25114 [==============================] - 453s 18ms/step - loss: 0.0928 - accuracy: 0.9590 - val_loss: 0.0915 - val_accura
cy: 0.9596
```

Figure 13 LSTM model training

```
10764/10764 [==============================] - 67s 6ms/step - loss: 0.0915 - accuracy: 0.9596
```

LSTM without directly dependent AB inputs

Then all the directly dependent inputs were removed from the input dataset. Total 14 in puts and 1 output were given to the LSTM model to train. 1 Dense layer was selected. And loss function binary_crossentropy was selected for binary classification.

Model was verified with the untrained data. Model prediction was same as LSTM with AB inputs and model couldn't correctly predict the specific condition in the data.

Validation with trained dataset:
`Loss = 0.0935, Accuracy= 0.9584`

Validation with untrained dataset:
`Loss = 0.1124, Accuracy= 0.9770`

LSTM Classifier WITHOUT Speed AB Inputs

In [44]:
```
num_time_steps = 10
num_features = 14

#X_train_lstm_w = np.reshape(X_train_lstm_w, (-1, num_time_steps, num_features))
#X_test_lstm_w = np.reshape(X_test_lstm_w, (-1, num_time_steps, num_features))

#Train model WITHOUT speed AB inputs
# Split the data into training and testing sets
X_lstm_w = df1[['State1', 'State2', 'Speed1', 'State3',
       'Speed2', 'State4', 'Speed5',
       'State5', 'State6', 'Speed7', 'Speed3',
       'Speed4', 'Speed8', 'Mode']].values
y_lstm_w = df1['BoolState1'].values

X_train_lstm_w, X_test_lstm_w, y_train_lstm_w, y_test_lstm_w = train_test_split(X_lstm_w, y_lstm_w, test_size=0.3)
```

In [45]:
```
#Shape of inputs
print(np.shape(X_train_lstm_w))
print(np.shape(y_train_lstm_w))
print(np.shape(X_test_lstm_w))
print(np.shape(y_test_lstm_w))

X_train_lstm_w = np.reshape(X_train_lstm_w, (-1, 14, 1))
X_test_lstm_w = np.reshape(X_test_lstm_w, (-1, 14, 1))

#y_test_lstm_w = np.reshape(y_test_lstm_w, (-1, 1, 1))

print(np.shape(X_train_lstm_w))
print(np.shape(X_test_lstm_w))
```

Figure 14 LSTM model without AB inputs

5. Model Comparisons

	With AB inputs	Without AB inputs
RFC	Accuracy: 0.9956238410596027 Precision: 0.9564332724502799 Recall: 0.9543492746490835 F1-score: 0.9320614355825253	Accuracy: 0.9956238410596027 Precision: 0.9102838457261793 Recall: 0.9540879654026557 F1-score: 0.9316713083984506
LSTM	Softmax: Loss: 0.3330, Accuracy: 0.0044 Sigmoid: Loss: 1.0622, Accuracy: 0.2912	Softmax: Loss: 0.1114, Accuracy: 0.0044 Sigmoid: Loss: 0.1193 - Accuracy: 0.9956

Table 1 Model Comparison

Time Comparison	Dataset	Config	Time
RFC	50 can csv	Estimators 50	~25 mins
LSTM	6 can csv	1 LSTM with 32 nodes 1 dense with 1 node activation sigmoid	2 hours per epoch
LSTM	6 can csv	1 LSTM with 32 nodes 1 dense with 1 node activation softmax	40 mins per epoch

Table 2 Model Time Comparision

Observations:

- RFC model couldn’t predict the specific condition in the data.
- LSTM model also couldn’t predict the specific condition.

6. Conclusion

- CAN logs in J1939 format were successfully pipelined to Python as Dataframe.
- Exploratory Data Analysis was performed on the data.
- Data was cleaned and prepared for model.
- RFC and LSTM models were trained with multiple configurations.
- The RFC model predicted 0(not observed) but couldn’t predict 1(observed).
- LSTM models couldn’t predict the specific condition in the data.
- In the whole dataset, availability of true value for specific condition was in 3 instances.
- More dataset is needed with true instances for specific condition.
- Model is needed which can truly predict the specific condition in the machine with atlease > 60%.

Future work

- Find a machine learning model which can truly predict the specific condition with at least > 60%.
- More dataset is needed with true instances for specific condition.

7. Bibliography/References

1. Time series classification models
   https://omdena.com/blog/time-series-classification-model-tutorial/

2. Machine Learning for Time Series Data in Python
   https://www.youtube.com/watch?v=ZgHGCfwExw0&list=PLkKdVy_oHuMN_dDzAgIzW9niOUJFBQFfP

3. LSTM-binary-cross-entropy
   https://arize.com/blog-course/binary-cross-entropy-log-loss/#:~:text=Binary%20cross%20entropy%20(also%20known,equate%20to%20high%20accuracy%20values.

4. Tiny ML
   https://www.allaboutcircuits.com/technical-articles/what-is-tinyml/

8. Completed Dissertation Checklist

a)	Check list of items for the Final report	
b)	Is the Cover page in proper format?	Y
c)	Is the Certificate from the Supervisor in proper format? Has it been signed?	Y
d)	Is Abstract included in the Report? Is it properly written?	Y
e)	Does the Table of Contents page include chapter page numbers?	Y
f)	Are the Pages numbered properly?	Y
g)	Are the Figures numbered properly?	Y
h)	Are the Tables numbered properly?	Y
i)	Are the Captions for the Figures and Tables proper?	Y
j)	Does the Report have Conclusion / Recommendations of the work?	Y
k)	Are References/Bibliography given in the Report?	Y
l)	Have the References been cited in the Report?	Y