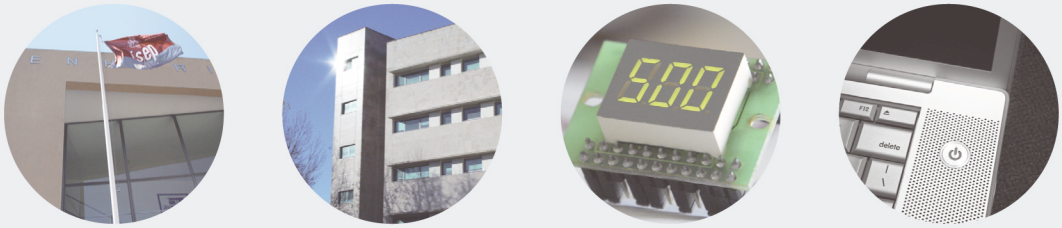
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



Deteção de Falha de Ignição

FRANCISCO JOSÉ MARAVILHA SANTOS
julho de 2021

POLITÉCNICO
DO PORTO

isep Instituto Superior de Engenharia do Porto

Misfire Detection

Master in Electrical and Computer Engineering - Automation and Systems

Francisco José Maravilha Santos

Orientation:
Ramiro Barbosa
André Roque
Pedro Pintado

Academic Year: 2020-2021

Instituto Superior de Engenharia do Porto
Department of Electrical Engineering
Rua Dr. António Bernardino de Almeida, 431, 4200-072 Porto

Acknowledgements

First of all, I want to thank the opportunity to do my thesis at CES, but especially to my two CES advisors, André Roque and Pedro Pintado, for their follow-up and availability throughout these months.

I thank my advisor, Professor Dr. Ramiro Barbosa, for his help, quick availability and guidance throughout this project.

My sincere thanks to all.

Abstract

This project analyzes some models of Machine Learning and Deep Learning in the context of misfire detection. Initially, the themes of misfire and Machine Learning are contextualized. The main objective of this project is to replace the existing ignition failure detection algorithm with a Machine Learning model. For this, the process was divided into several phases: problem formulation, data exploration, preparation and pre-processing of the data, building the model, and exporting it. For this, two versions of Matlab were used, 2020a and 2016b. The 2020a version was used to carry out all steps up to the export of the model. The 2016b version was used to perform the comparison with the detection algorithm already developed. Furthermore, dSpace TargetLink was used to generate the C code.

The construction of several models allows, through different metrics such as accuracy, precision, recall, and F1 score, to analyze and compare them and determine which model is the best.

With the completion of this project, we learned about the ignition failure event, but mainly about Machine Learning. All the necessary steps were learned, both in terms of data preparation and programming for the construction of the model and calculation of the respective metrics to evaluate the models. With this type of work, it was highlighted that Machine Learning is an iterative process, it can be present in the most diverse areas and with many different purposes. Machine Learning is already present in many industries and applications of our daily lives, but it is estimated that in the future its presence will be almost the majority.

Keywords: Misfire, Machine Learning, Deep Learning, MATLAB.

Resumo

Neste projeto é analisado alguns modelos de Machine Learning e Deep Learning no contexto da deteção de falha de ignição. Incialmente é contextualizado os temas falha de ignição e Machine Learning. O objetivo principal deste projeto é substituir o algoritmo de deteção da falha de ignição já existente, por um modelo de Machine Learning. Para isso dividiu-se o processo por várias fases: formulação do problema, exploração dos dados, preparação e pré-processamento dos dados, construção do modelo e exportação deste. Para isso, foram utilizadas duas versões de MATLAB, 2020a e 2016b. A versão 2020a é utilizada para realizar todas as estapas até à exportação do modelo. A versão 2016b é utilizada para realizar a comparação com o algoritmo de deteção já desenvolvido. Para além disso, foi utilizado o dSpace TargetLink para a gerar o código C.

A construção de vários modelos permite, através de diferentes métricas como a *accuracy*, *precision*, *recall* e *F1 score*, analisá-los e compará-los e aferir qual dos modelos é o melhor.

Com a realização deste projeto, aprendeu-se sobre o evento de falha de ingnição, mas sobretudo sobre Machine Learning. Aprendeu-se todos os passos necessários, tanto em termos de preparação de dados como a programação para a construção do modelo e cálculo das respetivas métricas para avaliar os modelos. Com este tipo de trabalho, realçou-se que o processo de Machine Learning é um processo iterativo, pode estar presente nas mais diversas áreas e com fins diferentes. O Machine Learning já está muito presente em diversas indústrias e aplicações do nosso quotidiano, mas estima-se que no futuro a sua presença seja quase maioritária.

Palavras-Chave: Falha de Ignição, Machine Learning, Deep Learning, MATLAB.

Contents

List of Figures

List of Tables

Glossary

Abbreviation	Description
AI	Artificial Intelligence
ANN	Artificial Neural Network
CPU	Central Processing Unit
CES	Continental Engineering Services
CNN	Convolutional Neural Networks
DT	Decision Trees
DNN	Deep Neural Network
ECU	Electronic Control Unit
EMS	Engine Management System
FNR	False Negative Rate
KNN	K Nearest Neighbour
LSTM	Long-Short-Term Memory
ML	Machine Learning
MLP	Multi-Layer Percepton
NB	Naive Bayes
OBD	On Board Diagnostic
PRNN	Pattern Recognition Neural Network
PCA	Principal Component Analysis
RF	Random Forest
SVM	Support Vector Machine
TPR	True Positive Rate

Chapter 1

Introduction

Humanity increasingly uses cars and the maintenance of vehicles has become a key aspect, as this allows to reduce both significant costs over the life of the vehicle and allows the prevention of a sudden breakdown. In addition, it can enhance reliability and increase the overall satisfaction of vehicle owners by reducing fuel consumption, emissions and improved comfort. The main concern of any vehicle in maintenance is the engine. Internal combustion engines require only three elements to operate: fuel, intake air and the ignition spark. A failure in one of these elements can lead to a misfire [16].

The misfire is very common in vehicles with an internal combustion engine, occurs in one or more cylinders of the engine and can arise due to different causes, such as: a defective spark plug, very high temperature, lack of air compression or even problems with the exhaust gas recirculation [1, 17]. As can be seen in Figure 1.1, according to carMD the main repairs made in 2018 were, both with the same percentage, the replacement of spark plugs and the oxygen sensor with 5.81% of repairs.

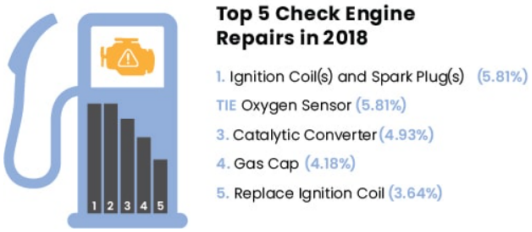


Figure 1.1: Top 5 checks engine repairs in 2018 [1]

Then, in the third place, the replacement of the catalytic converter, in the fourth, tighten or replace the tank cap, and in the fifth, the replacement of the ignition coil [1], thus reinforcing, as the main causes present in the engine, as stated before, that it can lead to misfire if not dealt with in time.

Currently, there are several methods to detect the misfire, where these can be divided into two categories, the direct and indirect methods [3].

One of the main objectives of detecting the misfire is to prevent the increase in order for the environment propagation by unburned fuel, loss of fuel economy, engine wear, among others [3, 16]. In accordance with European and American legal regulations, in order to reduce emissions, motor vehicles are required to monitor and detect misfire, as well as to identify which combustion cylinders have not been completed. This detection is a central part of the On Board Diagnostic (OBD). Its implementation consists of the use of sophisticated methods and the processing of the measured signals, discovering in very dense and complex algorithms [16].

1.1 Problem

As previously mentioned, the ignition failure detection algorithms are very dense and complex, they have a lot of data, where if it is necessary or calibrating something it is not very efficient. The overall idea is that the different engines and different markets need to have the algorithm adjusted, for example in Mexico City due to the altitude, the algorithm must be calibrated differently, so this work aims to simplify the ignition failure detection algorithm, already developed by Continental Engineering Services (CES), using the method of crankshaft angular speed through Machine Learning algorithms and to prove, or not, if it is possible to calibrate / change data autonomously and if the model developed in this project has better performance than CES model.

For this, the MATLAB 2020a software will be used to perform the data processing, MATLAB / Simulink 2016b for the Machine Learning model and the dSPACE TargetLink to generate the final C code.

1.2 Continental Engineering Services

Continental Engineering Services (CES) – a subsidiary of the Continental group – was founded in 2006, Figure1.2, to provide engineering services to industries. It started its activity in two German cities, Frankfurt and Nuremberg, and currently has more than 20 locations worldwide. With locations and teams spread across Europe, America, and Asia, it is within reach of all major industrial centers [2].

The work carried out at CES focuses on automotive electronics, drive and chassis technology, as well as electric mobility. In addition, they are also working on adapting automotive technologies to a wide spectrum of industrial applications. The work at CES ranges from consulting concept studies to carrying out prototypes and small series, having control of the entire product development process [2].

CES presents itself as the best partner in engineering solutions within the automotive sector as well as for all other industries. In Portugal, it is headquartered in Porto [2].

Figure 1.2: CES' logo [2]

1.3 Goals

As mentioned, the main objective is to replace the algorithm already developed by a machine learning algorithm. The work was divided into tasks to achieve the objective. These tasks are:

- Study and Understand the need for detection of misfire;
- Understand the need and what is ML;
- Study of different ML algorithms;
- Problem Formulation and Necessary Requirements;
- Data Treatment;
- Model Construction and Evaluation;
- Model Export;
- Comparison with CES algorithm.

1.4 Plan

The graph illustrated in Figure 1.3 summarizes the work plan for this project.

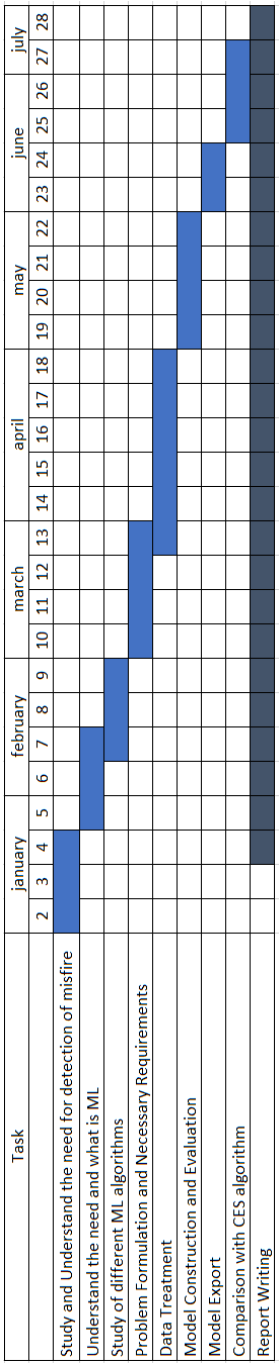


Figure 1.3: Development plan

1.5 Outline

Chapter 1 introduces what the misfire is and contextualizes the problem to be solved. In addition, the reader is made aware of the objectives of this project and its timetable.

In the next Chapter, 2, the state of the art is realized. In this chapter the main methods of detecting the misfire are discussed: in-cylinder pressure, crankshaft speed fluctuation, ion Current observation, and exhaust gas pressure. In addition, the approach to the ML theme is made: concepts and terminologies, learning scenarios, supervised learning algorithms, machine learning workflow and presence in the automotive industry.

In Chapter 3, the problem is formulated, showing how the data is organized and the behavior of some features and targets. In addition, the tools needed to solve the problem are exposed, such as MATLAB and dSpace TargetLink. Finally, through a block diagram, all the steps to be carried out in the implementation phase are explained.

In the next Chapter, 4, the implementation of the solution is carried out, explaining all the necessary steps. The result of the evaluation of the models and the reason for choosing one of them is also exposed.

In Chapter 5, a comparison is made with the existing model.

In the last Chapter, 6, the main conclusions and prospects for future developments are brought together.

Chapter 2

State-of-the-Art

The misfire causes inconvenience and problems for drivers and puts their safety at risk. In this chapter, an approach to the main ignition failure detection techniques is carried out, as well as the topic of machine learning and how it is inserted in the automotive industry.

2.1 Misfire Detection

In the past, many techniques and methods have been developed for the detection of misfire and have been classified according to whether these are based directly on the principles of physics or not, whereas direct methods are based on the principles of physics and indirect methods are not [3].

Direct methods are applied during the development of the engine and have the disadvantages of the high cost related to the use of many sensors and due to defects and malformations that may arise from the vibration or hostile environment of the engine. Examples of direct methods are the measurement of exhaust gas pressure, the measurement of the exhaust gas flow rate, temperature measurements and observation of cylinder pressure and ion current [3].

Indirect methods require model or converter assumptions for observations, as they do not directly use the principles of physics to detect misfire. Comparing them to direct methods, these have the advantage of being more economical because, while direct methods use many sensors, indirect methods use only one sensor and at best Engine Management System (EMS) sensors. The assumptions for creating these methods are based on models that describe the behavior of the system, and the accuracy of the response depends on the quality of the model. The main problems with these models stem from the complexity of some systems and the assumption that they will not change in the future. That is why

the models have to be constantly updated. Systems with non-linear behaviors require very complex algorithms, such as the measurement of the flow rate or pressure of the intake manifold, fluctuation of the crankshaft speed and vibration monitoring systems [3].

Crankshaft speed fluctuation is the most widely used method in the automotive industry for detecting misfire. This has the greatest challenge to maintain the accuracy of the result when the engine is at high speeds and has a large number of cylinders. To combat this, machine learning, adaptive filters and other techniques were started [3].

The Figure 2.1 shows the different methods for detecting the misfire, where all of these are used in the automotive industry.

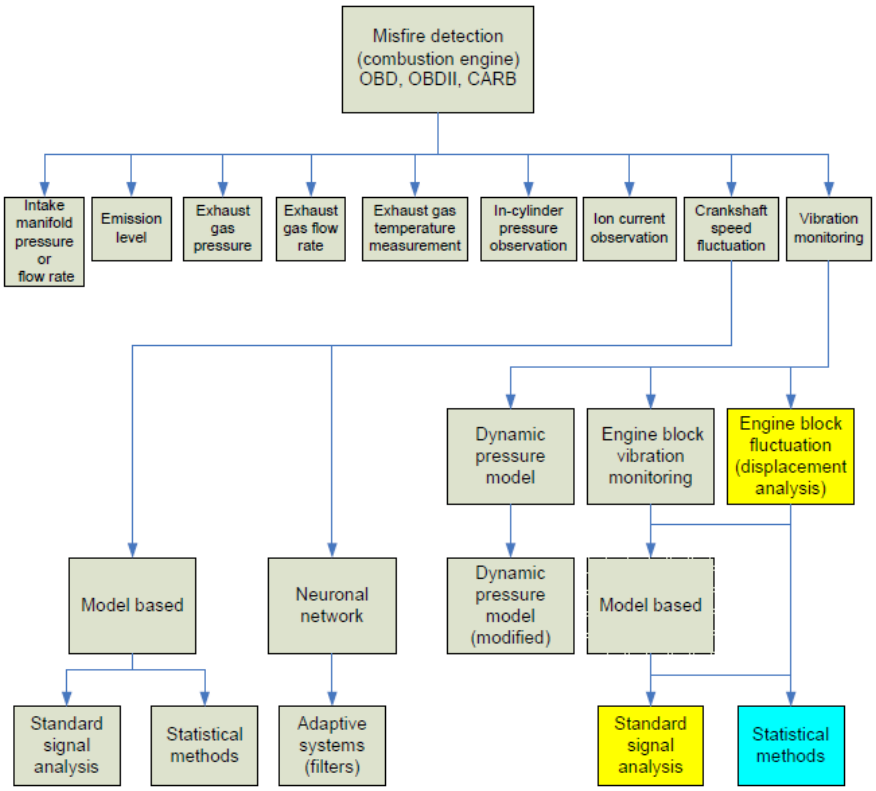


Figure 2.1: Misfire Overview [3]

Then, it exposes itself where the methods and techniques are mainly based such as in analysis of: in-cylinder pressure, crankshaft speed fluctuation, ion current and exhaust gas pressure.

2.1.1 In-cylinder pressure

The analysis and measurement of pressure in the cylinder is the most reliable method for detecting misfires, but it is not the most used method in the automotive industry, as it is not economically viable, as it requires pressure sensors in the combustion chamber of all cylinders, which consequently becomes expensive. Although technological advances have already been made in this field, making it more economically reliable, there are still many problems and limitations that make the work very difficult [3, 18, 19].

This method consists of the analysis of the cylinder pressure where, if there is any failure in the combustion process, it causes the cylinder to operate in "motorized" mode, which will result in a negative average effective pressure, thus signaling a misfire [3, 18, 19].

2.1.2 Crankshaft Speed Fluctuation

As stated earlier, fluctuating crankshaft speed is a more common method in the automotive industry. The non-combustion of the fuel mixture in an engine cycle causes the torque values of the rotation speed to decrease. An engine usually consists of a crank position sensor, a gear wheel, and other important sensors for managing an engine. The sprocket is usually mounted on the crankshaft and has a predefined position referenced to a fixed point on the engine. A sprocket usually has 36-2 teeth, leading to a 10-degree crank angle. If necessary, this resolution can be increased using interpolation methods [3, 19].

The speed sensor, connected to the Electronic Control Unit (ECU), captures the edges of the sprocket and transfers it to the ECU, where synchronization is activated on a specific tooth that will later be the reference for later calculations. At a given position from the crank angle to an ECU, it triggers an ignition pulse. If this ignition pulse and the combustion process are initiated, the crankshaft will be accelerated. In normal engine operation, the crankshaft speed frequency is stable and predictable over a full engine rotation. In this case, the engine has different widths between the cranks caused by the ignition events. In the event of a failure in the ignition system, compared to the operational operation of the engine, it shows a significant change in pulse widths having an instant influence on the crankshaft rotation speed, so the misfire can be detected by analyzing the speed crankshaft angle [3].

This method must be invulnerable to disturbances and distortions caused by the tolerance of the sensor wheel, torsional oscillations and engine cycles after the misfire, dynamic speed and load changes transferred to the engine crankshaft, the effect of road roughness at the angular speed of the engine crankshaft and time changes in the ignition circuit caused by the function of the active combustion detonator controller. For this to happen, special algorithms are used to

combat these problems, using various signal filtering techniques. The better the quality of the crankshaft model, the less likely it is that there will be mistakes, which leads to a robust monitoring system [18, 19].

2.1.3 Ion Current Observation

An analysis of the ionization current provides information on the quality of combustion and, based on this information, it is possible to establish various parameters of the combustion process, such as the pressure evaluated in the combustion chamber, the air-fuel ratio at the start of combustion, among others. A low ionization value means that combustion in a specific cylinder does not occur, and this information is useful for detecting misfire [3, 18, 19].

The ionization signal consists of three phases: ignition phase, flame front, and after flame. To extract the added features of this signal, it undergoes analog and digital processing. The information on combustion is stored in low-frequency offsets, the signal is processed to extract characteristics such as its maximum and the integral in a given definition window. These features of the ionization signal as a function of time, have not very clear information about ignition process and its failures [19].

2.1.4 Exhaust Gas Pressure

The pressure of the exhaust gases is easy to measure, as it undergoes variations as a result of the misfire. At each stage of the engine, the valve opens to change the load in the combustion chamber, causing the pressure in the outlet duct to increase, due to the sudden release of combustion products and piston movements while the valve moves. In the event of a misfire, the pressure of the exhaust gas decreases considerably, making it easy to identify when the failure occurs [18, 19].

This method is easy to work with, but the answer is not precise, and it is not possible to identify which cylinder the misfire occurred in. In contrast, this method can be used in conjunction with others to better understand the causes that led to misfire [18, 19].

2.2 Machine Learning

Learning, as a generic process, is about acquiring new, or modifying existing, behaviors, values, knowledge, skills, or preferences, among others. Learning from experience is a natural form of learning for humans, which is not the case when it comes to machines [20].

Machine Learning, is a category of artificial intelligence (AI) that consists of computational methods that improve performance or make accurate predic-

tions, using experience, making the process more autonomous. In this case, experience refers to those previously collected and made available to carry out their analysis. This data can be of different types and their quality and size are crucial for the success of future forecasts [20, 21]. That is, ML consists of choosing efficient and accurate forecasting algorithms. For the success of the chosen algorithm, as it is dependent on the data used, ML is inherently related to data analysis and statistics. The learning techniques are based on computer science concepts, using ideas of statistics, probability and optimization [21].

Finally, ML is a multidisciplinary field, with a wide range of research domains that reinforce its existence. ML is present in areas such as, statistics, data mining, pattern recognition, mathematical modeling, neuroscience, among others [22].

2.2.1 Concepts and Terminologies

The following is a list of some of the most commonly used terms in machine learning [21]:

- Data: data items used for learning or assessment;
- Characteristics: set of attributes, represented as a vector, associated with an example;
- Labels: Values or categories assigned to examples;
- Hyperparameters: a machine learning hyperparameter is a parameter that can be set before the learning process starts. It can affect the way a model trains. Some examples of hyperparameters in machine learning are : learning rate, number of epochs, momentum, regularization constant, number of branches in a decision tree, number of clusters in a clustering algorithm (like k-means), among others;
- Training sample: data used to train a learning algorithm;
- Validation sample: is a subset of data used to validate the model's performance and current parameters. Based on the results, it is possible to do another iteration on the parameters and reuse the validation sample to evaluate the model / performance of the parameters. This is very important, as the validation standard should not be used as training data, otherwise your performance metrics will be irrelevant. Thus, the division of training / test data (test = validation) is crucial in all data modeling problems, when working with labeled data;

- Test sample: provides the gold standard used to evaluate the model. It is only used once a model is completely trained (using the train and validation sets). The test sample is separate from the training and validation data and is not available in the learning phase;
- Loss function: A function that measures the difference, or loss, between a predicted label and a true label;
- Hypothesis set: A set of functions mapping features (feature vectors) to the set of labels Y.

2.2.2 Learning Scenarios

Learning is a very vast domain, so the field of ML has been divided into several subfields, where each deals with different types of learning. In general, there are four main kind of learning types: supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning [23], as seen in the diagram shown in the Figure 2.2.

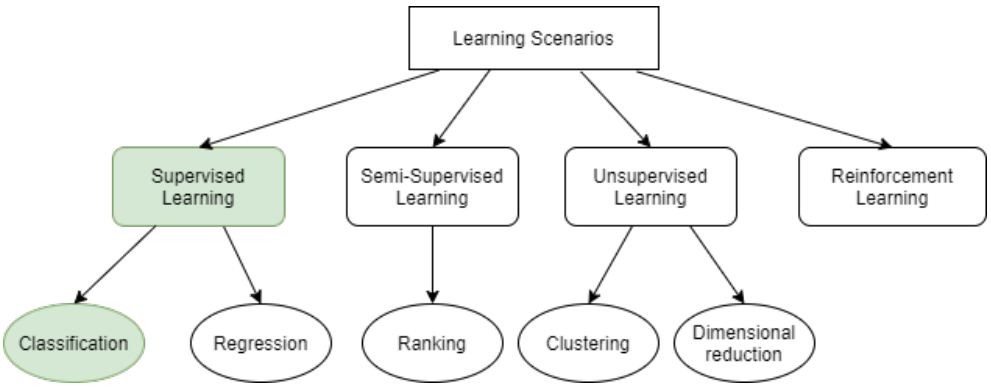


Figure 2.2: Learning Scenarios Overview

Supervised learning

Supervised learning is normally used for predicting events. In general, the objective of supervised learning is to create a function, given a sample of data and desired results, it comes closer to the relationship between observable input and output in the data. The fact of being supervised is related to the analysis made by an external supervisor who understands the data domain, about which output should be exultant. The hypothesis is evaluated, according to the ability to predict the output value of new examples. Basically, supervised learning means that the data being introduced into the model is labeled. Problems such as classification and regression are present in this type of learning [22, 24].

- Classification: is the problem of assigning a category to each item, as for example the documents can be classified as being of politics, economy, sport, among others, while in the classification of an image, that image can be categorized as being a car, a person, a landscape, etc. Depending on the number of output classes, the problem may be binary or multiclass classification problems [22].

  – Binary Classification: is a type of supervised classification problem where the target class label has two classes and the task is to predict one of the classes. An example is the detection of whether the email is spam or not, cancer detection and it also fits into this work, as it wants to detect whether there was a misfire or not [24]. The Figure 2.3 shows a binary classification problem, where the components are separated according to their type and color.

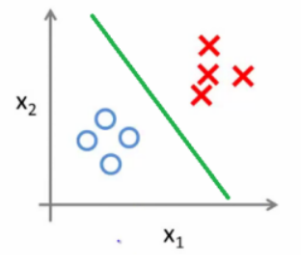


Figure 2.3: Binary Classification Example [4]

  – Multiclass Classification: refers to the classification task that has more than two class labels. Exists two different approaches to solve this types of classification problems: One vs Rest strategy splits a multi-class classification into one binary classification problem per class, and One-vs-One strategy splits a multi-class classification into one binary classification problem per each pair of classes. Examples of multiclass classification are: face recognition, animals recognition, among others. The Figure 2.4 shows a multiclass classification problem, where the components are separated according to their type and color [24, 25].

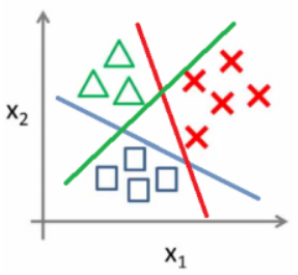


Figure 2.4: Multiclass Classification Example [4]

- Regression: it is the problem of predicting a real value for each item. Regression algorithms are used for problems that deal with questions like "how much" or "how many". Forecasting stock value or economic variations are examples of regression problems. In regression problems, an incorrect prediction is the greater the difference between the true and predicted values [22].

The main algorithms for a supervised learning scenario are linear regression, logistic regression, k-nearest neighbors, decision trees, random forests and neural networks [26].

Unsupervised learning

Unsupervised learning is used to describe events not yet known. Unlike supervised learning, it has no labeled output, being useful in exploratory analysis because it can automatically identify the data structure. The algorithms of unsupervised learning, learn from data and when new ones appear, they use previously learned resources. Common problems in this type of learning are clustering and dimensional reduction or manifold learning [22, 24].

- Clustering: it is the problem of dividing a data set into subsets taking into account their similarity, with the new clusters or classes being labeled. It is used to analyze very large data sets [22].

- Dimensional reduction or manifold learning: it is the problem of reducing a large initial quantity of items to one of a lower dimension, preserving the properties of the initial representation. Digital image pre-processing in computer vision tasks is an example of this type of problem [22].

The main algorithms for an unsupervised learning scenario are k-means, neural networks, principal component analysis, independent component analysis, self-organizing maps, and hidden Markov models [26].

Semi-supervised learning

Semi-supervised learning is a mixture of supervised and unsupervised learning, as it is a mixture of labeled and non-labeled data. In most situations, unlabeled data is in abundance while labeled data is scarce. The use of unlabeled data in the training process can bring benefits to the final model, making it more accurate. The problems that normally use this type of learning are classification, regression or ranking [22, 24].

- Ranking: it is the problem of ordering items according to some criteria. Searching the web is an example of a canonical ranking, as it is necessary to show the most relevant results depending on the search. Many other ranking problems also arise from information extraction or natural language processing systems [22].

Reinforcement Learning

In reinforcement learning the training and testing phases are mixed. For data collection, you have to interact directly with the environment, affecting it in some cases. You receive an immediate reward for each action, aiming to maximize these rewards over a course of actions and iterations with the environment. The problem is that this type of learning does not provide any long-term reward feedback, that is, you must choose between exploring unknown actions or exploring information already collected [22, 24]. Examples of reinforcement learning algorithms are Q-learning and Deep Q-Learning [26].

2.2.3 Machine Learning Algorithms

Taking into account the type of problem in this work, a binary classification problem, since one wants to find out if there was a misfire or not, the study of the algorithms for this type of problem is made. These algorithms are [24, 27, 15]:

- Naive Bayes (NB): Bayesian algorithms are a set of probabilistic classifiers used in ML-based on Bayes' theorem. The Naive Bayes classifier was one of the first algorithms used in the context of ML. Suitable for binary and multiclass classification, it allows making predictions based on previously obtained results. A classic example of using this type of algorithm is spam filtering systems. This algorithm takes little computational time for training and returns the probability, which makes it very simple to apply for a wide variety of tasks. Bayesian algorithms are not applicable when it is necessary to take into account the interactions between resources;
- Decision Trees (DT) and Random Forests (RF): DT are very easy to use, interpret and explain. Some famous algorithms are ID3, c4.5, c5.0 and cART.

DT can handle several types of data: numeric, nominal, textual, among others. They have a good generalization, are robust and provide good performance for a relatively small computational effort. When there is a large number of data it is difficult to work with decision trees, because even if the computational time is small, due to a large number of data the time when building the tree would increase considerably. Decision trees tend to "divide to conquer", which results when it is a small data set, but not when the opposite is true. The errors spread through the tree, making them even more problematic. In addition, as there is a large number of data, the tree grows with the possible consequence of several records on each small leaf, where it subsequently makes the statistical decisions about the represented class very unreliable - data fragmentation problem. This problem can be avoided if a division limit is established. Without proper pruning, decision trees can be over-fit, which is why a RF set model was developed.

RF, as already mentioned, is a classification algorithm that uses a set of unprocessed DT (pruning methods not applied), each of which is built on a specific sample of the data (bootstrapping). Random forests are fast, scalable, robust to external noise, easy to use and do not over-adjust, but when the size of the data increases, the algorithm becomes slow for real-time prediction. In the following Figure 2.5 is represented a diagram showing how a random forest works.

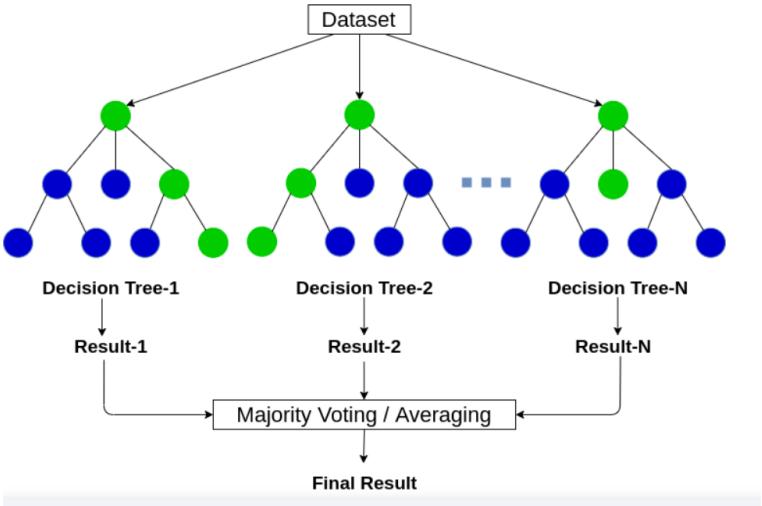


Figure 2.5: Random Forest Example [5]

- K nearest Neighbour (KNN): It is a non-parametric classification algorithm. This consists of an unlabeled sample point, the class of the closest

to a set of previous labeled points. This record is independent of the distribution of the sample points and their classifications. K Nearest neighbor is suitable for multiclass classification problems. Simple method, but it has a very small efficiency, without even being described as a “lazy” method. Its performance depends on the selection of an opportune value for "k", and the only way to assign a good value to this parameter is by using expensive computational techniques. Furthermore, it is not a very robust algorithm as it is affected by noise and is sensitive to irrelevant characteristics.

- Support Vector Machine (SVM): are learning machines for classification. It is a complex algorithm but it provides good accuracy. SVM map a predictor vector to a higher dimensional plane, using linear and non-linear kernel functions. This algorithm is part of the group of supervised learning techniques, where it aims to produce a model (based on training data) that predicts the target values of the test basis. The performance and accuracy of the algorithm are independent of the size of the data set or the number of training cycles. Typically, this type of algorithm is popular in solving text classification problems. Robust for large size of data and has a good generalization, but the training speed is slow and the performance depends on the chosen parameters. In the following Figure 2.6 is represented a resume and how SVM works.

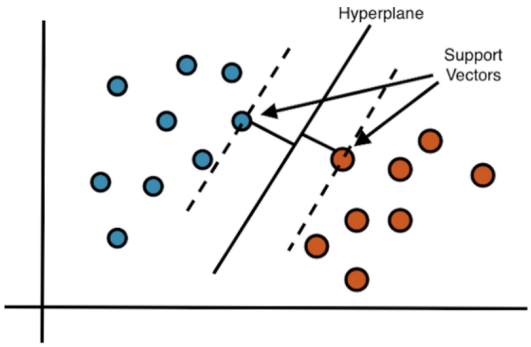


Figure 2.6: SVM Example [6]

- Artificial Neural Networks (ANN): are a good choice for solving classification problems. Basically, neural networks simulate the functioning of human neurons and are used in several fields such as robotics, economics, applied mathematics, among others. In most cases, a neural network will only have one output variable, although, in multiclass classification problems, it can be more than one output variable. This type of algorithm depends on three fundamental aspects: input functions, network architecture, and the weight of each input connection, the first two aspects being

fixed, the behavior of the algorithm will depend on the assigned weights. Initially, the weight values are randomly assigned, and then data sets for training are exposed to the network. Subsequently, the obtained output value is compared with the desired value. After this analysis is made, the network weights are adjusted to get closer to the desired value. These are a good alternative to conventional techniques as they can capture many types of relationships that allow you to model phenomena quickly and relatively easily.

ANN can be of different structures: perceptron or multi-layer perceptron (MLP). Perceptron is one of the simplest ANN architectures and deals with a single neuron where it uses linear patterns to obtain the results. Each input is connected to a weight, the value of each output is linear, and therefore a neural network with this structure is not able to obtain complex patterns. The Figure 2.7 shows an example of a perceptron structure.

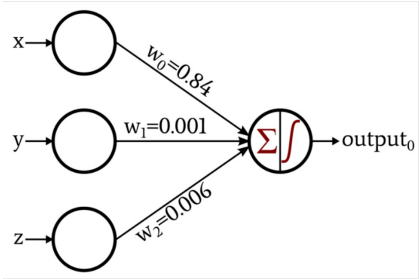


Figure 2.7: Percepton Example [7]

The MLP is the evolution of the perceptron, more developed and with a greater number of neurons. These arise with the aim of dealing with non-linearly separable problems, adding hidden neuron layers to the model. The Figure 2.8 shows the structure of a MLP.

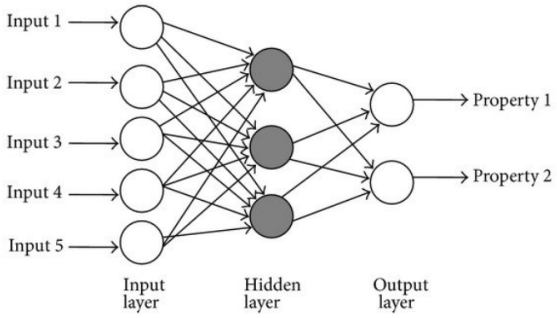


Figure 2.8: MLP Example [8]

The complexity of an MLP network is due to the number of hidden layers added. There are three types of layers: input layer, hidden layer, and output layer. The input layer provides the initial elements of the network, the hidden bed is responsible for processing the data and has no contact with the outside world. The function of neurons in the hidden layer is to interfere between the input and output layers. The output layer transmits the final value.

In the Table 2.1, a summary of the algorithms described throughout this section is made, and compared.

Table 2.1: Comparing Learning Algorithms (**** stars represent the best and * star the worst performance) [15]

Characteristics	DT	ANN	NB	KNN	SVM
Accuracy	**	***	*	**	****
Speed of learning	***	*	****	****	*
Speed of classification	****	****	****	*	****
Tolerance to missing Values	***	*	****	*	**
Tolerance to irrelevant Values	***	*	**	**	****
Tolerance to redundant Attributes	**	**	*	**	***
Tolerance to noise	**	**	***	*	**
Model Parameter Handling	***	*	****	***	*

2.2.4 Machine Learning Workflow

In the previous sections a theoretical review of the main techniques available to develop an ML model was carried out. In this section, the necessary steps will be taken in practice to develop a generic ML model. All steps are illustrated in the following diagram, Figure 2.9, and explained below.

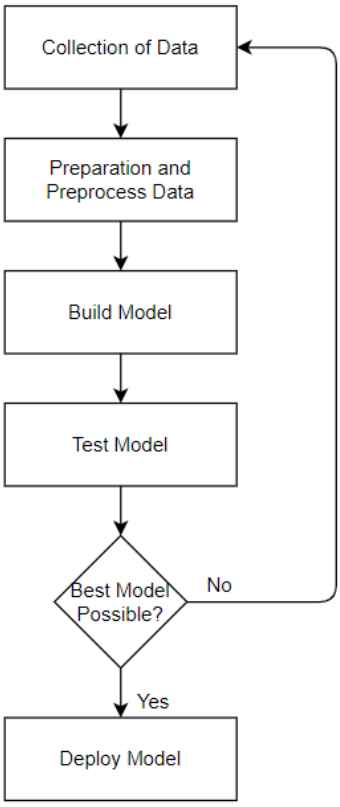


Figure 2.9: Steps to build a ML model

1. Collection of data: Data is the basis of any machine learning project. This stage of the project involves the collection and selection of data. Each of these phases can be divided into several stages. Data collection is the first step towards building a machine learning model. The quantity and quality of the data will determine the accuracy of the model to be developed. It is difficult to estimate which part of the data will provide the most accurate results until model training begins, which is why it is important to collect and store all the data. The data can represent quantitative characteristics (numerical format) or qualitative characteristics (symbols or text), and depending on their type, the operations to be performed are defined. The graphically represented data allows a better understanding and analysis of these. After all the information is collected, the data is chosen to solve the defined problem. The selected data includes attributes that need to be considered when building a predictive model [23, 28].

2. Preparation and Preprocess Data: The purpose of pre-processing is to convert raw data into data in a format adapted for machine learning. To

achieve these objectives, the data undergoes some procedures such as [28, 29]:

- Data cleaning - this procedure allows removing noise, correcting data inconsistencies, filling in missing data using imputation techniques, for example filling with the average value. In addition, it also checks for outliers - observations that deviate greatly from the rest of the data. If they exist, they are removed as incomplete and useless data.
- Sampling: If large data sets exist, the data sampling procedure can be used, where a representative portion is selected. Not all the data collected is necessary or adds anything to the operation of the algorithm.
- Formatting: each algorithm needs inputs to be provided in a specific file format depending on the programming library used (List, .csv, .txt, .mat)
- Dimensioning: in a more final stage of pre-processing, the variables obtained can be of several units or scales, being necessary to normalize them to a fixed scale, for example between 0 and 1.

The preparation and pre-processing of the data is a gradual and time-consuming process.

3. Build Model: With the treatment of the data done, the construction of the model follows - this being the most relevant step, despite representing a small part of the development time. The creation of the model aims at generalization - grouping similar phenomena with the ability to adapt to new data. For this, the model is presented with input variations until a good accuracy is achieved through data analysis (training), learning from the data (validation) and the completion of the model's performance (test). In the following Figure 2.10, it is possible to observe the division of the sample subset [23, 28].

Figure 2.10: Data Split [9]

In this stage, the training and validation data are treated. The training and test set represents the largest part of the data set, usually 80% of the total samples. The training set, as already explained in section 2.2.1, includes the set of input examples in which the model will adjust its parameters, and since the model needs a large volume of data to learn the patterns,

these are 80% of the set of training. In order for the model to learn the classifications it needs to be evaluated periodically in the test stage. The model will loop between training and validation until it reaches an optimal model [23].

4. Test Model: Second-last stage of the construction of the machine learning model where the generalization test of the built model is carried out. In this phase, a calculation of the accuracy of the prediction is made on a different set of data. This is considered a step outside the construction of the model because it is a different set, about 20% of the development sample. The evaluation of the model is carried out through the analysis of several metrics and they vary according to the type of problem. As the problem of this project is a classification problem, the main metrics being accuracy, precision and recall [23]. These metrics are calculated from a confusion matrix, as shown in Table 2.2.

Table 2.2: Confusion Matrix

	Actually Positive	Actually Negative
Predicted Positive	True Positives (TP)	False Positives (FP)
Predicted Negative	False Negatives (FN)	True Negatives(TN)

Each row of the matrix represents the instances in a predicted class and each column represents the instances in a real class. Thus, is possible to calculate the referred metrics from this matrix as follows [23]:

- Accuracy: number of correct forecasts divided by the total number of forecasts:
$$\frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$
- Precision: number of correct forecasts in a class by the total number of forecasts in the class:
$$\frac{TP}{TP + FP} \tag{2.2}$$
- Recall: number of correct predictions in a class by the actual total number. Positive class called True Positive Rate (TPR) for positive class and False Negative Rate (FNR) for negative class:
$$\frac{TP}{TP + FN} \tag{2.3}$$
- F1 score: statistical metric that combines recall with precision in a single number:
$$\frac{2TP}{2TP + FP + FN} \tag{2.4}$$

Finally, after the calculated metrics, the test and training results are compared. If the results found are similar, it means that a good model was obtained, if they are divergent, the main problem to take into account is overfitting - when the model is unable to make predictions to new data. Therefore, it is necessary to go back to previous steps to make changes to the data. Note that machine learning is an iterative process [23, 28].

5. Deploy Model: After obtaining the optimal model, it is implemented. There are two types of implementation: Active models - predictions generated by the model are treated automatically without human involvement, and Passive models - predictions generated by the models have human involvement, and these models have more problems because they are exposed to human error [23].

2.3 Machine Learning in Automotive Industry

Machine Learning can be used in the most diverse areas such as e-commerce, health, finance, cybersecurity, the automotive industry, among others. As this project aims at a problem related to the automotive industry, it will be exposed below where machine learning is used in this industry. The Figure 2.11 shows the principal areas where ML acts in automotive industry.

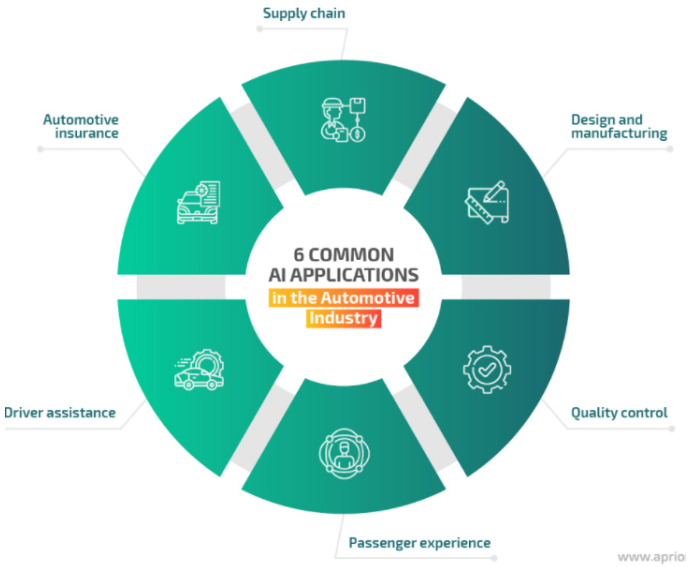


Figure 2.11: ML in automotive industry [10]

- Design and manufacturing: Solutions with AI technology and ML algorithms help vehicle manufacturers to improve production processes, as-

sess cases of risk and damage to vehicles, among many other things. The use of technology such as computer vision, natural language processing, and conversion interfaces are implemented in robotic solutions and widely applied in vehicle manufacturing. Nvidia has developed a video card, the Quadro RTX, Figure 2.12a, which significantly speeds up the design workflow. Another example is the use of collaborative robots to perform more boring tasks, such as handling heavy materials and inspecting the produced parts, Figure 2.12b [10].

(a) Nvidia Quadro RTX [30]

(b) Collaborative Robot in Automotive Industry [31]

Figure 2.12: Design and manufacturing examples

- Supply Chain Optimization: Along the supply chain, analytical models are used to identify demand levels for different marketing strategies, selling prices, locations, and many other data points. This predictive analysis determines the required inventory levels at different facilities. These models are constantly tested for different scenarios to ensure the best possible stock level, reducing unnecessary retention costs [10, 32].

  After comparing current and projected stock levels, optimization models are created that help guide the exact flow of stock from the manufacturer to distribution centers and ultimately to customer-facing windows. ML helps make this process more efficient and profitable while improving customer service and brand reputation [10, 32].

  For example, Blue Yonder uses AI technologies to increase visibility into inventory movement and enable manufacturers to predict potential delivery disruptions. In the Figure 2.13 is represented the interface of Blue Yonder [10, 32].

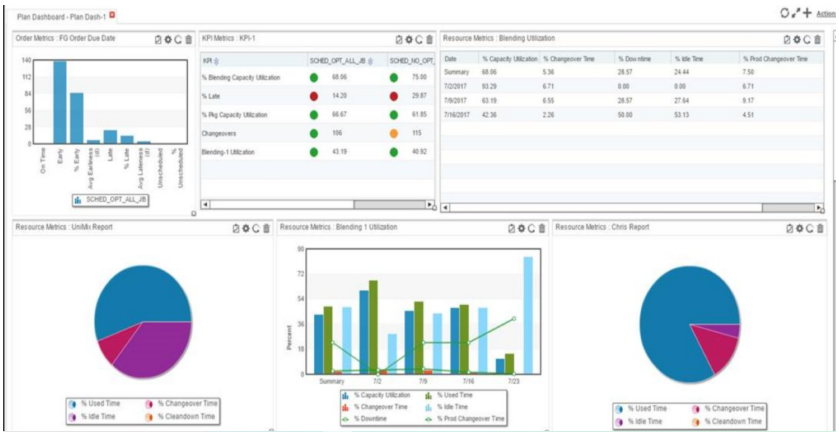


Figure 2.13: Blue Yonder interface [11]

- Quality Control: Using artificial vision algorithms, it is easy to detect anomalies and eliminate them during the vehicle construction process. These algorithms are highly accurate and allow you to detect a defective part, after which it is decided whether that part is reworked later or if it is eliminated. One of these options is much less expensive than repairing the part later. In addition to this more economical aspect, it also helps to ensure safety, satisfaction and customer retention. For the construction of the model, a large number of images is necessary, with anomalies and others without [10, 32].

  BMW uses AI-based solutions for predictive maintenance of weld tongs and paint quality analysis, among other tasks. Another example is the platform developed by Predii where it prescribes vehicle repairs based on the analysis of sensor data [10].

- Predictive Maintenance: ML algorithms can help with vehicle maintenance, helping drivers with accurate and important information, thus protecting your vehicle as well as maintaining your own safety. Instead of a static maintenance schedule updated a few times a year, you can have a predictive analytic model where you are continually learning from the thousands of performance points collected at the factory, suppliers and real vehicles on the road. The sector is moving towards these types of predictive schedules that evolve over time. This work is also an example of predictive maintenance, as mentioned above [10, 32].

- Enhancing overall in-vehicle user experience: The use of machine learning facilitates the personalization and intelligent personal assistance in vehicles. This learns the user's traits, creating profiles that can be used to provide personalized services and assistance. Facial recognition algorithms and methods of identifying the mood of passengers and driver are used.

For example, brands like Dentsu and Hyundai have invested a few million in these systems, where passengers will be able to choose music, listen to personalized playlists and so on. Amazon is also working on the integration of Alexa - (voice assistant) in various vehicles such as BMW, Toyota, Ford and Audi, Figure 2.14 [10].

Figure 2.14: Alexa in Vehicles [12]

- Driver Assistance: There are several models of machine learning that assist the driver, informing them about traffic, weather, shortcuts on routes or even paying for goods and services when driving. CarVi, Figure 2.15a is an advanced driver assistance system (ADAS) that analyzes traffic data and notifies the driver, in real time, of possible dangers. In a more advanced role there are other systems that aim to assume the role of the driver, either temporarily as in some Tesla cars or in driverless cars by Waymo, Figure 2.15b [10].

(a) CarVi [33]

(b) Waymo [34]

Figure 2.15: Driver Assistance examples

- Automotive Insurance: ML and AI based solutions can also be useful for handling insurance claims. On the driver side, these solutions can store incident data and fill out claims. On the insurance side, through image processing it can help improve the accuracy of vehicle damage analysis.

An example of this type of application on the driver side is Nauto's intelligent fleet management (Figure 2.16), based on artificial intelligence technologies, it has a collision detection feature that allows faster and more accurate processing of insurance claims. On the insurance side, the Ping An Auto Owner app evaluates the photos uploaded by users who make insurance claims [10].

Figure 2.16: Nauto [13]

2.3.1 Challenges of Machine Learning in Automotive Industry

Although ML and AI are increasingly present in the automotive industry, there are still several challenges in building an AI solution: biased AI, low data quality, limitations of devices and sensors and non-compliance with local regulations [10]:

1. Biased AI: AI-based solutions can be biased. One can try to solve the problem using robust algorithms instead of an AI solution. The chosen algorithms responsible for critical security functions must be carefully analyzed and verified. Algorithm accuracy is evaluated manually. You must first verify that the algorithm works correctly before introducing new data.

2. Low data quality: Data is the focal point for any AI system, so it should be as good as possible. However, collecting good quality data is challenging. The AI models used for the vehicles must be predictable, accurate and fast enough to have the right and safe response in real time. Some of the necessary data can be collected through intelligent systems, robots used in the factory or sensors in the car. When this is not possible, the data must be created artificially.

3. Limitations of devices and sensors: The quality of data is highly dependent on the technical capabilities of the sensors and devices used to collect

it. For example, when deploying a machine learning model to process audio data received from microphones, it may be more effective to record audio using ultrasonic devices rather than regular devices as these can filter out background noise.

4. Non-compliance with local regulations: There are legal gaps regarding the development of AI powered solutions for the manufacture of vehicle transport. However, there are requirements that must be taken into account when developing the software. This requirement and recommendations can come from The Society of Automotive Engineers International, The European Automobile Manufacturers' Association, among other entities.

2.4 Conclusion of the Chapter

In this chapter, it was presented the misfire is and its main detection techniques. Subsequently, ML was approached. The concepts and terminologies were defined, its learning paradigms were exposed, as well as its problems and algorithms. Then, all the necessary steps to build a ML model were shown. Finally, it was shown how ML is being applied in the automotive industry.

In short, there are several approaches that can be adopted using ML, and it is necessary to evaluate the problem and what is intended to be solved. The automotive industry is very competitive and a ML approach can allow industries to evolve, discovering and capitalizing on the hidden value in their operations, creating new growth opportunities.

Chapter 3

Problem Formulation

Misfire, as explained in Chapter 1, means that the engine's explosion cycle did not work correctly: explosion too early, or too late, the explosion did not occur, among other consequential factors from different causes. These misfire events affect performance and can cause damage to system components, which is why detection of these events is important as it can prevent irrecoverable damage, thus serving as a diagnostic tool during maintenance.

Currently, CES detects misfire events through complex functions related to crankshaft speed, explained in section 2.1.2. In this algorithm, several detection methods are applied. Each method results in an output that represents a deviation from velocity. Then, a "merger" is made to these outputs, applying weights and limits to each method, and these values vary according to the vehicle, type of engine, speed, load, among other factors, resulting in many values and tables to calibrate. As these algorithms are highly dense and complex, these calibrations are complicated to do, wasting a lot of time when it is necessary to calibrate, making them inefficient. ML's approach to replacing these complex algorithm functions aims to make calibration automatic, without the need to waste time with manual calibration. Also, make this algorithm less "memory" intensive.

In terms of the type of learning, this uses the supervised learning scenario, explained in section 2.2.2, as the machine learning model will be trained with the labeled data and with the desired results, in order to predict an outcome, in this case if there was misfire or not. It is a binary classification problem, explained in section 2.2.2, as our target, misfire event, only has two classes - either misfire occurred (0) or did not occur (1).

To implement the solution to this problem, three components will be needed: data, MATLAB software and dSpace TargetLink. It will use two different ver-

sions of MATLAB, version R2020a, for all data processing and model development, and version R2016b, which is the version where the Simulink model already developed by CES is found and is also the version supported by dSpace TargetLink to generate the C code.

3.1 Data

Data is at the heart of any ML approach. In this case, the data is generated artificially, through a CES hardware, where it is possible to simulate the operation of the engine, which it is possible to simulate the event of misfire. Figure 3.1 illustrates this data and how it is organized.

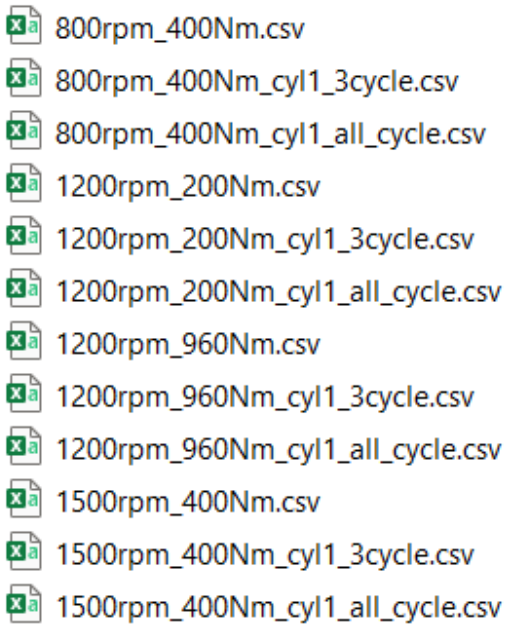
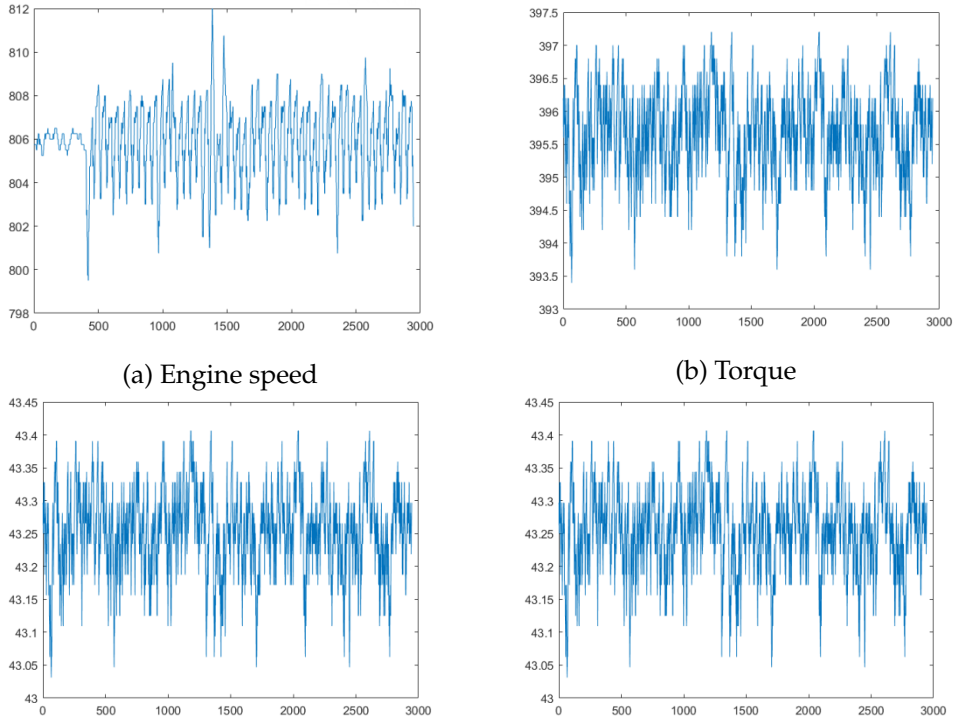


Figure 3.1: Data Organization

As can be seen from the figure, the data correspond to cylinder 1. These are in excel format and there are 3 data files for the same speed and torque. For example for a speed of 800 rpm and a torque of 400 Nm, the following files exist: *800rpm_400Nm.csv, 800rpm_400Nm_cyl1_3cycle.csv and 800rpm_400Nm_cyl1 800rpm_400Nm_cyl1_all_cycle.csv*. In the first file, data was collected without any misfire. In the second, data was collected where the misfire occurred in the 3rd engine explosion cycle, and in the last file this event always occurred. Regarding the collection of data from the remaining cylinders, it was carried out and organized in the same way.

Each dataset is 98 columns, with each column representing a feature or a target. For the misfire detection method, only 6 columns will be required, 5 of which are features and one is the target. The features are: the engine speed (*is1_eng_speed*), the torque (*etc_trq_act_eff*), the fuel mass of bank1 (*fis_act_fm _combustion_bank1*) and bank2 (*fis_act_fm_combustion_bank2*) and the flag that indicates if the car is in motion or stopped (*mfd_cyc_based_non_stationary_cond_b*). The target is the feature *mfd_amp_no_avg_dist_mean_cyl1*. This feature is the ignition amplitude. This signal is related to 'ignition amplitude', signal coming from outside the ECU (comes from the ignition ECU) that indicates the voltage at which ignition occurred (disruption voltage of the gas in the cylinder varies with pressure, temperature, etc.), therefore serving as a strong indicator for the occurrence of misfire.

Figure 3.2 shows four of the five features used by the amplitude detection method. The *mfd_cyc_based_non_stationary_cond_b* is not illustred in the figure, because this feature is always 0. These features samples are stored in the dataset *800rpm_400Nm_cyl1_3cycle.csv* indicated above. Figure 3.2(a) is the engine speed, Figure 3.2(b) the torque and Figure 3.2(c) and Figure 3.2(d) the mass of fuel in Bank1 and Bank2, respectively.



(a) Engine speed

(b) Torque

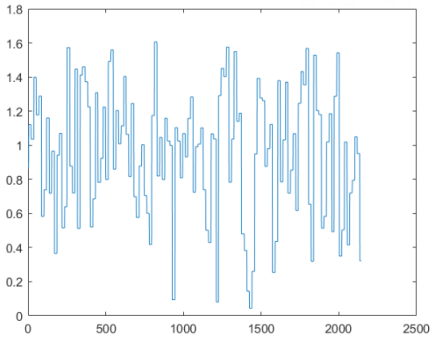
(c) Fuel mass bank1

(d) Fuel mass bank2

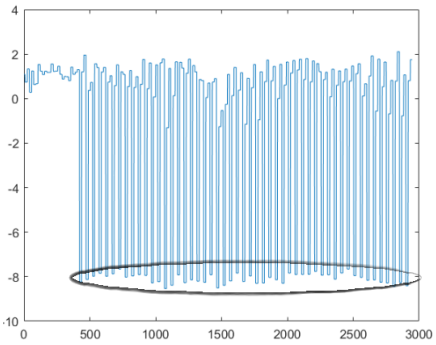
Figure 3.2: *800rpm_400Nm_cyl1_3cycle.csv* features

From the plots in Figure 3.2, it is possible to see some important aspects. The *fis_act_fm_combustion_bank1* and *fis_act_fm_combustion_bank2* features have the same behavior and values. Furthermore, when comparing the torque with these two feaures, it can be seen that they have a very similar behavior, only with the difference in the data values.

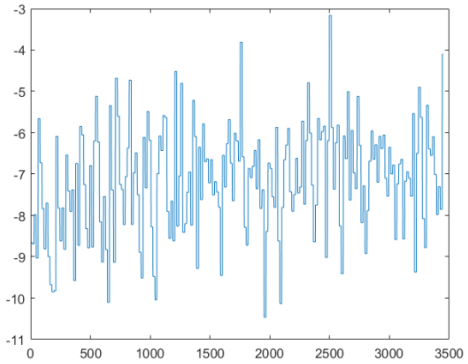
In the following figure, the target for the different datasets mentioned above is represented. In Figure 3.3(a), without a misfire event, in Figure 3.3(b) with the misfire event to occur in the third cycle, and in Figure 3.3(c) is represented when the misfire always occurred. As already explained, it is possible to observe that there is a misfire event when the value is less than -4.



(a) *800rpm_400Nm.csv* target



(b) *800rpm_400Nm_cyl1_3cycle.csv* target



(c) *800rpm_400Nm_cyl1_all_cycle.csv* target

Figure 3.3: Example of target in different datasets

All data analysis and exploration is done later in section 4.1.

3.2 MATLAB

MATLAB is a software optimized for solving scientific problems. In this software, calculations, visualization and programming are integrated in a user

friendly environment and the solutions expressed in familiar mathematical notation. Figure 3.4 illustrates the MATLAB interface [35].

The name MATLAB is an acronym for MATrix LABoratory. It was created with the purpose of being a matrix calculation software. Currently, it is much more than that due to evolution over the years with user input. The language in MATLAB is matrix based which represents the most natural form of computational mathematics. To install this software some minimum requirements are necessary such as: an operating system, any Intel or AMD x86-64 processor, AVX2 instruction set support, with Polyspace 4-core is recommended, 2 GB for MATLAB only, 4–6 GB for a typical installation [35].

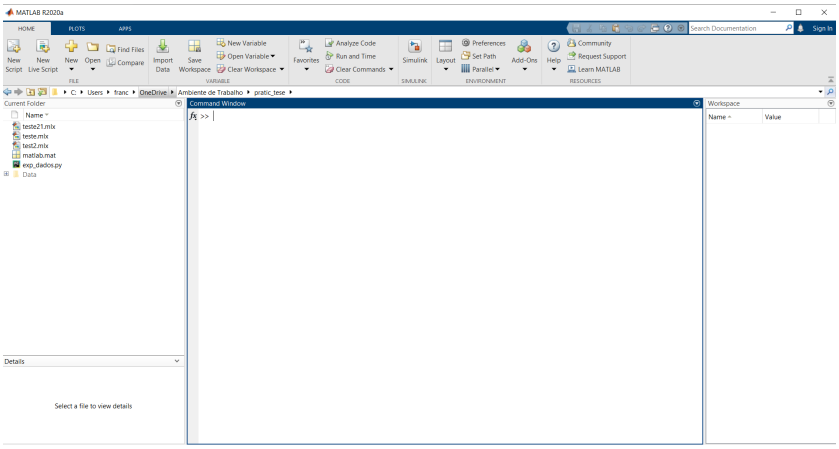


Figure 3.4: MATLAB interface

Solving ML problems becomes easy using the tools provided by MATLAB, because it provides a strong environment for interactive exploration, with numerous algorithms and applications that help in this task. MATLAB has two specific toolboxes for processing machine learning problems. These are the Machine Learning Toolbox for the classification algorithms and the Deep Learning Toolbox for the Neural Networks, Figure3.5 [35].

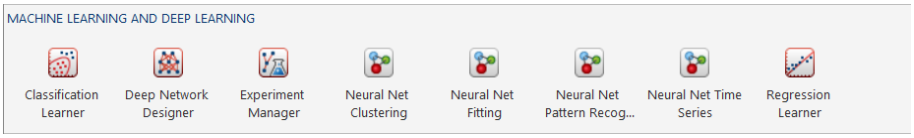


Figure 3.5: Machine Learning and Deep Learning Toolboxes

- Machine Learning Toolbox: contains all the tools needed to extract knowledge from large datasets. It provides functions and applications for ana-

lyzing, describing and modeling data. Exploratory data analysis and preprocessing are easy to do with everything this toolbox offers. Descriptive statistics, visualizations, and clustering can be used for exploratory data analysis; probability distributions to the data; generate random numbers for Monte Carlo simulations and run hypothesis tests. Regression and classification algorithms allow you to create predictive models interactively through Classification and Linear Regression applications, or programmatically through AutoML. For multidimensional data analysis and resource extraction, this toolbox offers Principal Component Analysis (PCA), regularization, dimensionality reduction and more. For classification problems this toolbox offers apps and functions that can cover a variety of parametric and non-parametric classification algorithms: Logistic regression Boosted and bagged decision trees, including AdaBoost, LogitBoost, GentleBoost, and RobustBoost, Naive Bayes classification, KNN classification, Discriminant analysis (linear and quadratic), and SVM (binary and multiclass classification. The Classification Learner application, Figure 3.6, is a very useful tool that does more than just to train data, it interactively explore data, select resources, specify data schemes, cross-validation and evaluate results. Furthermore, this toolbox allows user to select and train several models at the same time [35, 36].

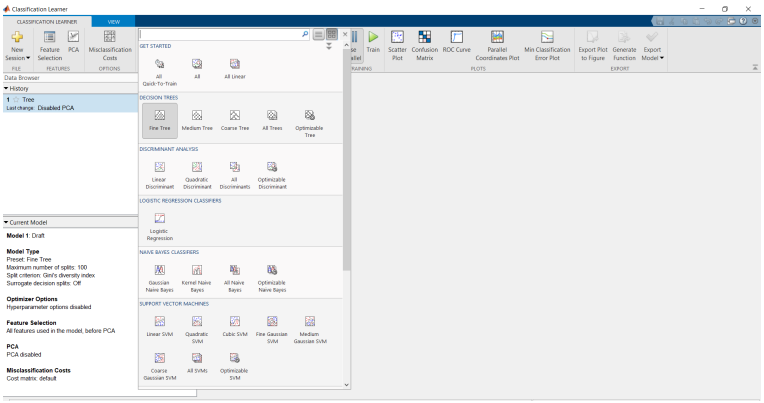


Figure 3.6: Classification Learner App

- Deep Learning Toolbox: it provides a framework for designing and implementing neural networks with algorithm, pretrained models and applications. Convolutional Neural Networks (ConvNets, CNN) as well as Long-Short-Term Memory (LSTM) can be used to perform classification or regression of images, time series and text data. Through the application Deep Network Designer, Figure 3.7, it can design, analyze and train networks graphically. The Experiment Manager application allows to manage

several deep learning experiments, control training parameters, analyze the results and compare codes from these different experiments. It can also view layer activations and graphically monitor training progress [37].

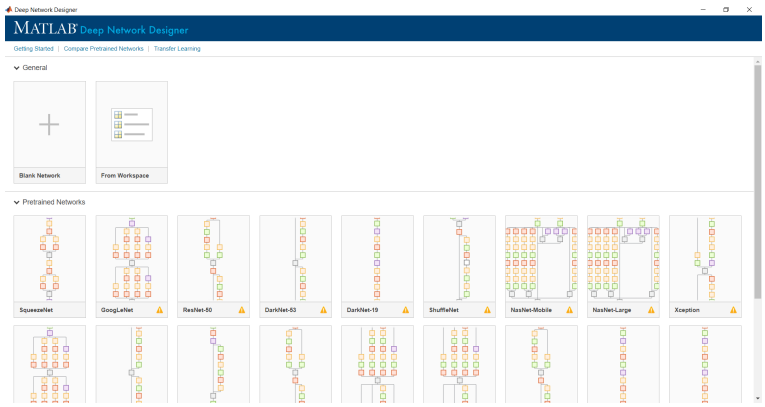


Figure 3.7: Deep Network Designer App

3.3 dSpace TargetLink

Model-based design has become the established development method in many industries, and production code generation is the logical step to transforming models into efficient, production-ready code. TargetLink generates production code (C code) directly from the Simulink graphical development environment, Figure 3.8.

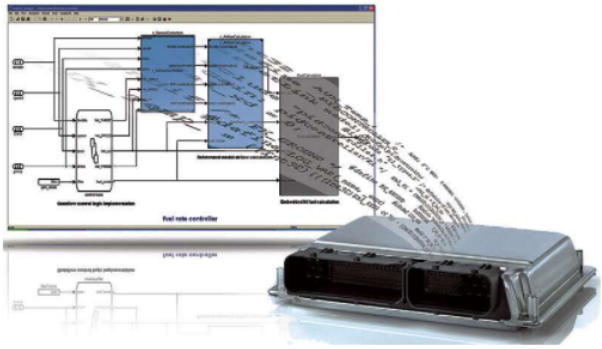


Figure 3.8: dSpace TargetLink [14]

Code generation options, from ANSI C to code for AUTOSAR platforms. Versatile code configuration options ensure that production code can handle processor constraints. The dSpace TargetLink has several benefits such as the

ability to be highly efficient and configurable, have powerful software design features, be suitable for AUTOSAR projects, and more [14].

The diagram illustrated in the Figure 3.9, summarize the workflow from model design to code implementation. It also shows that code verification based on simulation is an iterative process.

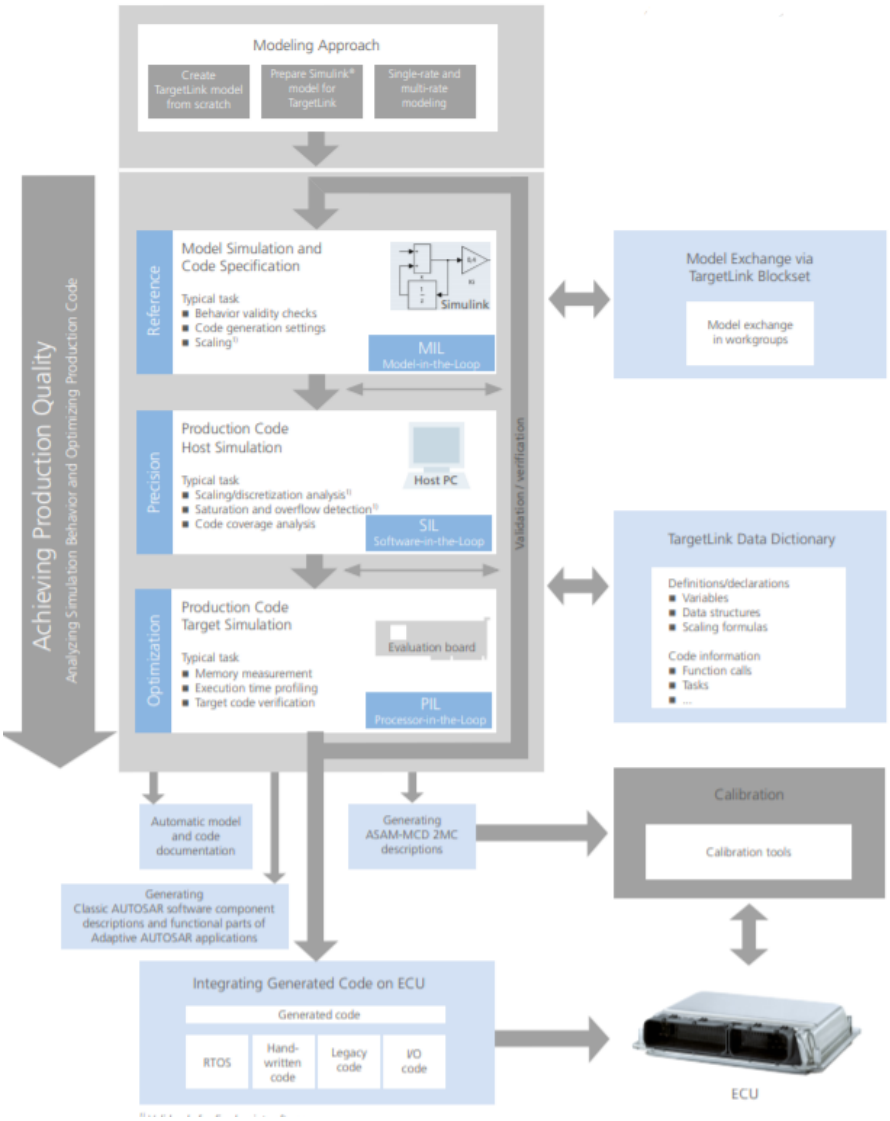


Figure 3.9: Dspace Targetlink Workflow [14]

3.4 Conclusion of the Chapter

In this chapter, the problem to be solved was explained in more detail, as well as the requirements needed to solve it. It was explained how the data is col-

lected, the advantages and why of choosing MATLAB and the chosen versions, and explained the usefulness as well as some advantages of dSpace TargetLink.

Chapter 4

Solution Implementation

In this chapter, all stages of implementation will be described, such as: data exploration, data preparation and pre-processing, building model and deploying the model. The complete code can be found in Appendix A and all the confusion matrices of the models in Appendix B.

4.1 Data Exploration

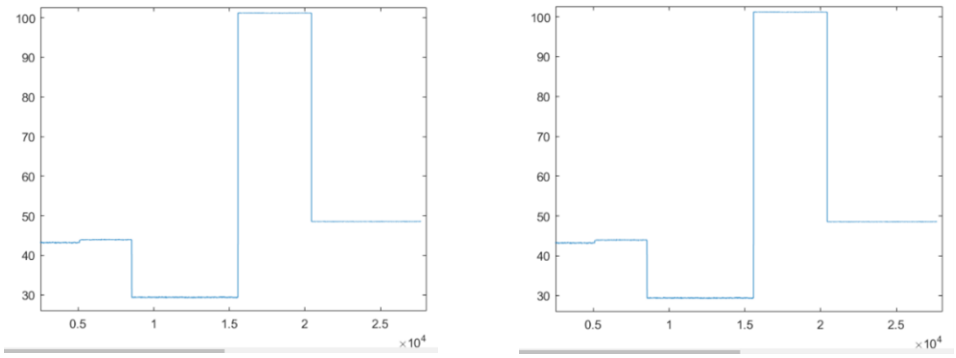
The data, as explained in section 3.1, are organized in different excel files, being all constituted by 98 columns, where each column represents a feature. For the method of detection of the misfire event through amplitude, only six of these features will be used for the implementation of the ML model, because for this method these are the ones used in the model implemented by CES. Five of these inputs are features and one is the desired target. The five inputs are: fuel mass on bank1, fuel mass on bank2, engine speed, engine torque and the non-stationary-b flag. The target is the *misf-amp-no-avg-dist* of cylinder 1. When its value is less than -4 it means that the misfire event occurred, as already explained.

As the data are divided into several excel files, they were vertically concatenated, saving them in a variable *dataset*. Afterwards, the analysis of the different features and the target is made, checking its maximum, minimum, average and behavior throughout the engine cycle. This analysis is explained and shown in the following subsections.

4.1.1 Features

In the Figure 4.1 is possible to see the behavior of the features *fis_act_fm_combustion_bank1*, Figure 4.1(a) and *fis_act_fm_combustion_bank2*, Figure 4.1(b). As

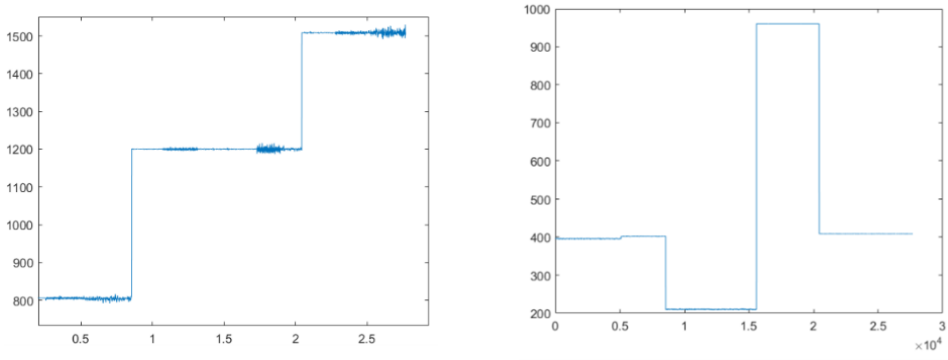
already mentioned, these features represent the fuel mass of bank1 and bank2 and, as shown in the Figure 4.1, they have the same behavior. Both have the same average, minimum and maximum, 43,938 kg, 29,172 kg and 101.23 kg, respectively, and their behavior throughout the engine cycle is the same. In the graph, it is not possible to see the small variations that these features have, as shown in Figure 3.2(c) and 3.2(d), as this graph represents the data all concatenated, and this small variation exists in it.



(a) *fis_act_fm_combustion_bank1* (b) *fis_act_fm_combustion_bank2*

Figure 4.1: Fuel Mass in Bank 1 and Bank 2

The Figure 4.2 represents the behavior of the motor speed and its torque. The speed averages 1200 rpm, minimum 791.75 rpm and maximum 1529.8 rpm. It is possible to observe all speeds present in the data, 800 rpm, 1200 rpm and 1500 rpm and their small variations, Figure 4.2(a). The Figure 4.2(b) shows the behavior of the motor torque. Torque averages 402.6 Nm, minimum 208 Nm and maximum 960.8 Nm. It is also possible to observe, as with speed, the different values of the motor torque: 200 Nm, 400 Nm and 960 Nm.



(a) is1_eng_speed (b) etc_trq_act_eff

Figure 4.2: Speed and torque

Finally, in Figure 4.3 the *mfd_cyc_based_non_stationary_cond_b* is represented, where this already reveals a strange behavior, as it should always be 0, according to the information required by CES. As can be seen in the Figure 4.3, this does not happen, and these points are a strong possibility of outliers.

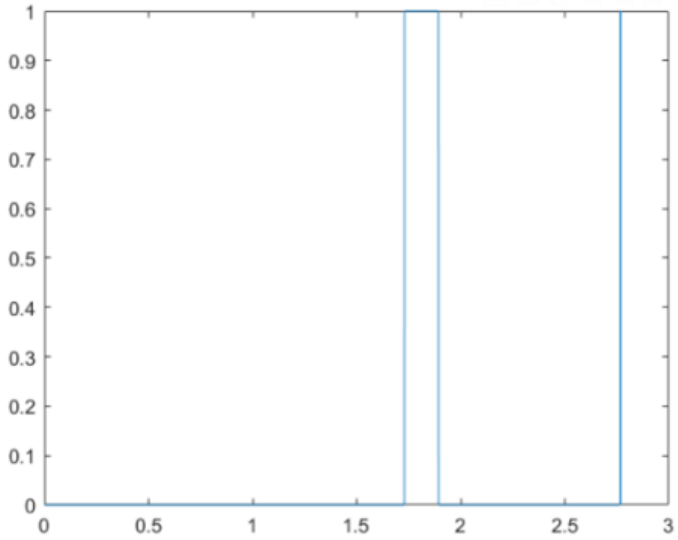


Figure 4.3: Non-stationary flag

This type of exploratory analysis of the features, that will be used to build the model, is important because it allows for a better understanding of the behavior of the data, and if any cleaning or treatment is needed in the next phase. What can be seen from the analysis done is that, in principle, none of the features has a lack of data and, in terms of outliers, it is possible to observe some unusual data in *mfd_cyc_based_non_stationary_cond_b*. This will be confirmed and corrected if necessary in the next phase - Preparation and Preprocess Data.

4.1.2 Target

In the Figure 4.4, the behavior of the target can be observed, after concatenation of all data. This has an average of 0.035, a minimum of -10.462, and a maximum of 8.299. As explained above, when the value is less than -4 it means that the misfire event occurred, that is, this being a binary classification problem. One of the objectives is to transform this target, so that it is only constituted by two classes: 1 if it did not occur and 0 if it did occur.

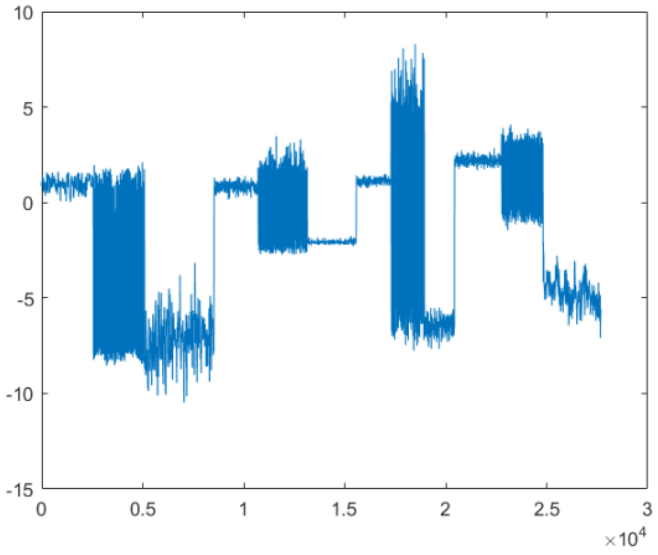


Figure 4.4: Target

After the analysis of the features and the target is done, it moves on to the next phase - Preparation and Preprocess Data.

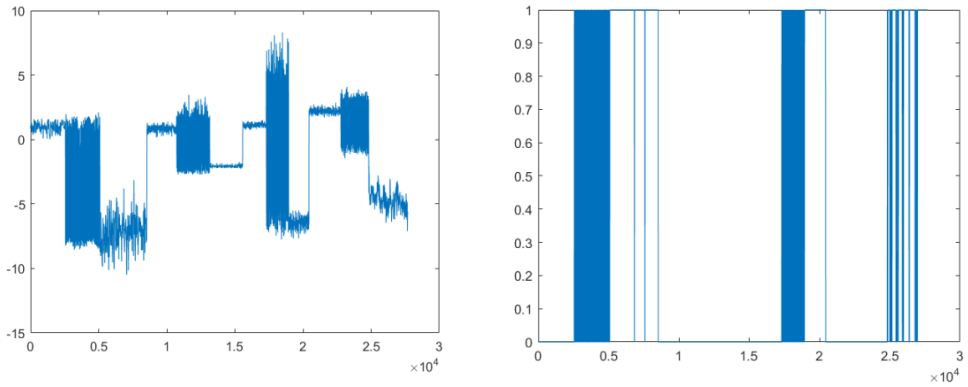
4.2 Preparation and Preprocess Data

This section explains all the steps taken and its results until the data are ready to build the model.

4.2.1 Target Preparation

As said before, the first task to be performed is to transform the target, with all those variations, into a target consisting only of two classes: 0 and 1. For this it was performed a *for* loop that will check, through the if conditions, all lines of these features. If it is less than -4 it means that there was a misfire, then the value 0 is saved in the *misf* variable, if it is greater, the saved value is 1. After the cycle ends, the variable created is in the form of a line, so it was necessary to convert it to a column. This conversion was stored in the variable *misfire_amp*.

In Figure 4.5 it can be seen how the target was before and after this transformation.



(a) Before (b) After

Figure 4.5: Target before and after being prepared

4.2.2 Data Cleaning

For data cleaning two functions were used: the *ismissing()* function and the *isoutlier()* function.

The *ismissing(A)* function, with A being one of the five features, returns a logical array that indicates which elements of an array or table contain missing values. If there are missing values, the matrix returns 1 in that position, otherwise it returns 0. The size of the response matrix is equal to the feature size. In addition to this function, the *sum(B)* function, with B being the answer logical array, was used. This function will add all the positions where a missing value was identified, making it known how many values are missing. After applying this process to the features, it was found that in fact there is no lack of values in the data, as provided for in section 4.1.

The *isoutlier(A, 'method')* function, with A being one of the five features. Like the *ismissing()* function, also returns a logical matrix. The method used was the averaging method, this is the "method" parameter of the function.

This function was used to check how many outliers there are in the features. This method returns true for elements with more than three standard deviations from the mean. This process was processed to all features, and *mfd_cyc_based _non_stationary_cond_b* was the only feature where outliers were detected, 1673 in total. Once detected these outliers must be corrected.

In values that are considered outliers, it returns the value 1 at that position. In this case was also used the *sum(B)* function, with B being the answered logical matrix. For this, the *filloutliers(A,'fillmethod')* function is involved, parameter A being the feature *mfd_cyc_based_non_stationary_cond_b* and the "fillmethod" the spline method. This method consists of filling using cubic spline interpolation

by parts. The Figure 4.6, represents the signal before and after being treated, as well as the outliers found and the threshold used for their detection.

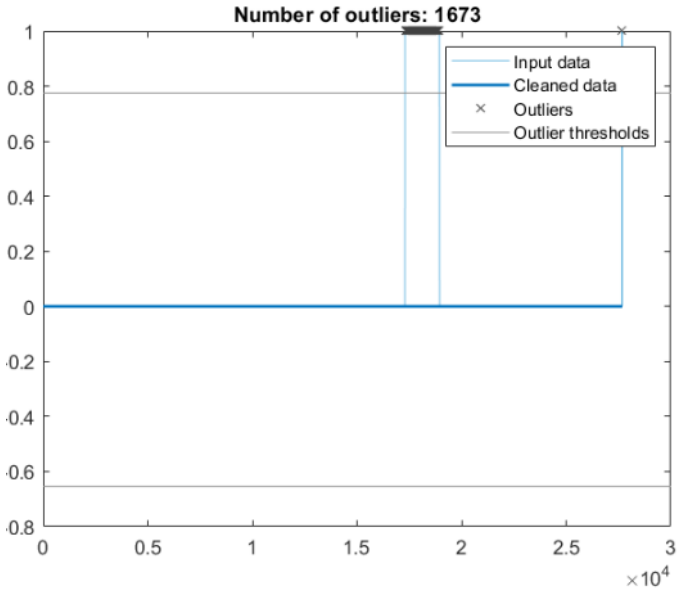


Figure 4.6: Detection and correction of outliers

4.2.3 Data Correlation

After clearing the data, the correlation between the data was performed. Correlation is useful to identify if different features have similar behavior, in which case for this model, these features contain the same information. In short, correlation makes it possible to reduce the number of inputs in the model, making it simpler. To verify the correlation between the data, the *corrplot(X)* function was used, where the X parameter is a variable that contains the data to perform the correlation. In Figure 4.7 the result is represented.

As can be seen from the Figure 4.7, of the 5 features, only 4 are represented in the graph, as the feature *mfd_cyc_based_non_stationary_cond_b* is always 0, it was not possible to calculate the correlation with the other data. We can also infer that as this is always 0, it would not be necessary for the model either. From the 4 represented, it is verified that 3 features have a high correlation, of 1. These 3 features are: the fuel mass in bank1 (the first *fisa*), the fuel mass in bank2 (the second *fisa*) and the engine torque (etc1). As said before, as these have a very high correlation, of these 3 features, it will only be necessary to use one. That is, through the correlation it was possible to reduce the size from 5 features to 2. These two are the engine speed and its torque.

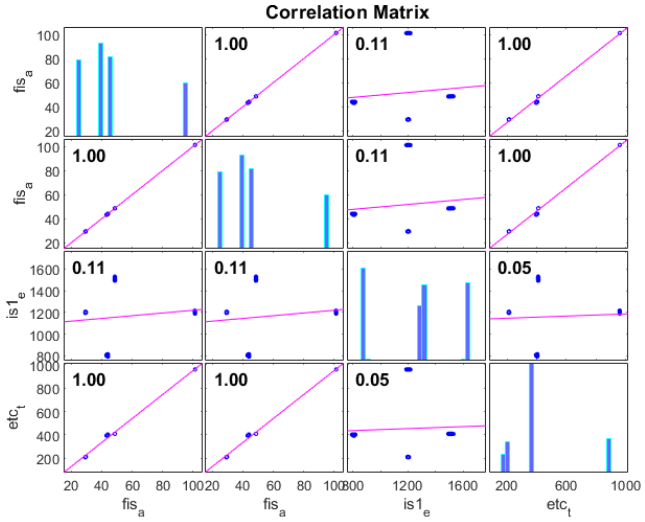
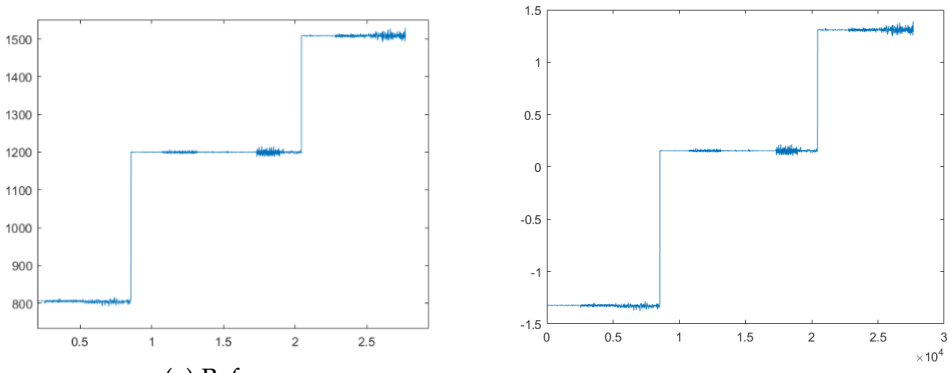


Figure 4.7: Correlation Matrix

Then, the last step of the pre-processing was carried out, the normalization of the data. Normalization, as already mentioned in section 2.2.4, serves to reduce data redundancy, increase integrity and increase its performance. For data normalization, the *normalize(A,method)*, where parameter A is the variable that we want to normalize, and method is the method that we will normalize the data, in this case was the *z-score* method. This method consists of centering and scaling the data to reach a mean of 0 and a standard deviation of 1. The equation (4.1) represents this method, where u is the mean and s is the standard deviation.

$$z = \frac{x - u}{s} \tag{4.1}$$

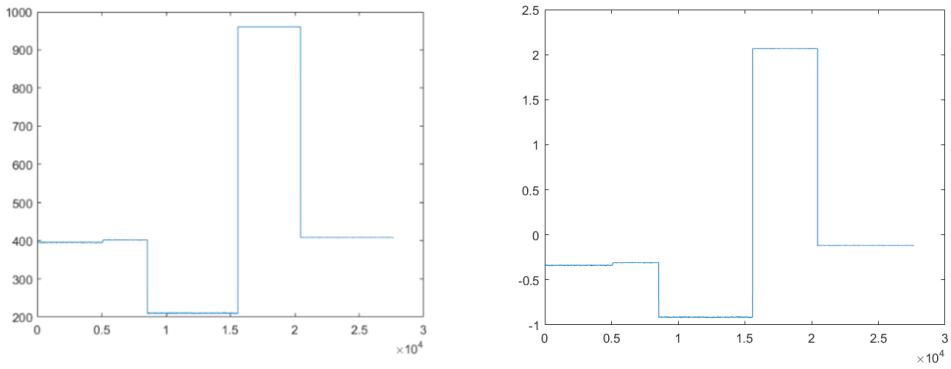
In the Figure 4.8 and 4.9 it is possible to observe the features before and after they are normalized.



(a) Before

(b) After

Figure 4.8: Speed Normalization



(a) Before (b) After

Figure 4.9: Torque Normalization

As can be seen from the previous figures, the behavior of the features has not changed, its values were simply readjusted, so that the model can obtain a better performance. Note that the range for normalization was chosen based on the data. The best solution would be to choose the range for a specific car. For example, a torque between 0 and 1000 Nm or a speed between 500 rpm and 7000 rpm.

In the variable *dataset_v3*, the features with all the treatment described throughout this chapter and the target were stored, being then constituted by 3 columns and 27682 rows.

4.2.4 Data Split

Finally, after all the preprocessing was done, the data was divided to train and test. To perform this division the *cvpartition(n,'Holdout',p)* fuction was used. This function creates a random nonstratified partition for holdout validation on n observations. This partition divides the observations into a training set and a test. So the parameter 'n' is the *dataset_v3* and the parameter 'p' is set to 0.2, to split the data in 80% for training and 20% for test. The holdout method was used because it is recommended for large datasets. After division, data were organized and stored by 4 variables:

- Xtrain – contains data for training features;
- Ytrain – contains target data for training;
- Xtest – contains feature data to test the model;
- Ytest – contains target data to test the model.

After dividing the data, the models were built, trained and tested.

4.3 Build Model

As mentioned in section 3.2, two toolboxes were used, ML and DL. This section is divided according to the toolbox used, as the model building process differs in some aspects.

4.3.1 Model Through Machine Learning

Through this toolbox, several classification models were built and trained, such as NB, DT, KNN, and SVM. The training and testing process of the various models is very similar, with the only difference being the function to be used according to each model:

- NB - *fitcnb()*;
- DT - *fitctree()*;
- KNN - *fitcknn()*;
- SVM - *fitcsvm()*.

In this subsection, the training and testing process of the DT model will be explained, and as already mentioned, the process for the other models is similar.

After the data is prepared and separated, the model is trained with the appropriate data for this task. To train a DT, as mentioned above, the function was used *fitctree(Tbl,ResponseVarName)*. This function returns a fitted binary classification decision tree based on the input variables (also known as predictors, features, or attributes). In this case the *Tbl* parameter is the Xtrain and the ResponseVarName corresponds to the Ytrain. The trained model is stored in the *dtc* variable.

With the trained model it is already possible to predict some results. The prediction is made through the *predict(model, data)* function. This predicts the output of an identified model for new data. So, the parameter "model" is the DT trained model - *dtc*, and the new data is the data stored in *Xtest*. The result is stored in the variable *y_predict_dtc*.

Finally, the result obtained from the trained model is compared with the actual results of the *Ytest* variable. For this, we use the *confusionmat()* function and insert as parameters, as mentioned above, *Ytest* and *y_predict_dtc*. In Figure 4.10, the resulting confusion matrix can be seen.

Figure 4.10: DT confuison matrix

As already mentioned in section 2.2.4, through the confusion matrix it is possible to calculate several metrics to be able to evaluate the model. All the results of the metrics of all models, either through this toolbox or the deep learning one, are summarized in Table 4.1, where the analysis is then made and the reason for choosing the model to use is explained.

4.3.2 Model Through Deep Learning Toolbox

Through this toolbox, two types of ANN were developed. One already developed in the toolbox called Neural Pattern Recognition and another built and developed from scratch.

Pattern Recognition Neural Network

Basically, a Pattern Recognition Neural Network (PRNN), is a feedforward network that can be trained to classify inputs according to target classes.To create and train a Pattern Recognition Network, the *patternnet(hiddenSizes,trainFcn, performFcn)* function is used, which arguments are: hidden layer sizes, which by default is 10, the function to be trained, by default the function used is Scaled Conjugate Gradient (*traincsg*) and finally the performance function by default is crossentropy.

In MATLAB, for the training of this network, the data cannot be in the form of a table, nor the features and targets divided by columns. In order to solve the problem, the features and the target were separated, stored in variables x and y respectively. Then the data was transformed from table to array. Finally, in the

variables *x_t* and *y_t*, the result of the transposition of *x* and *y* was stored. After this process, the data are already usable for network training.

For training, we started by using the default settings, but after some testing it came to the conclusion that some default settings had to be changed. The network was created and the following settings were made:

- Changed default training function. Better results were obtained with the Bayesian Regularization (*trainbr*) training function;
- The data preprocessing is done inside the netowrk through the parameter "net.input.processFcns", where the function "mapminmax" was executed, which basically normalizes the signals in an interval of [-1,1]. Other normalizations were also tested, this being the one with the best result.
- Data division: the data was divided into 3 sets: 75% of the data for testing, 10% for validation and the remaining 15% for testing.
- Transfer functions: this network consists of 2 layers, the first transfer function being logosig and for the second softmax.
- Plots: the plots to show related to the network training were defined as the performance plot, the loss confusion matrix and the training status.

With all the settings performed, the network was trained using the t*rain(net, X, T)* function, with the net parameter being the trained network, the *X* variable the network inputs, in this case it corresponds to the variable *X_T* and the parameter *T* that corresponds to the target of the problem, in this case *Y_t*.

When performing the training of the net, a window appears, Figure 4.11, where it is possible to see the constitution of the net, the net performance during the training, the confusion matrix, the training status and error histogram. In addition, it is also possible to see the settings made when building the network.

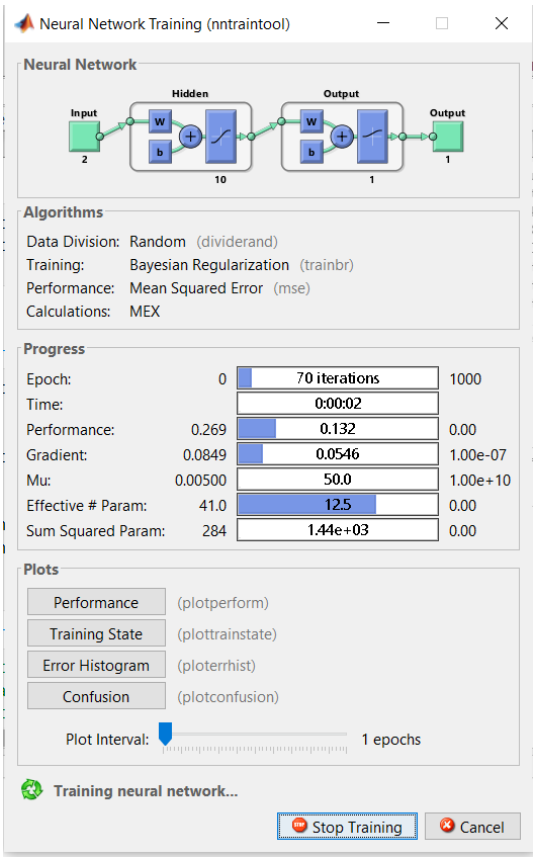


Figure 4.11: Pattern Recognition Network

Then the prediction of test data was made. But before making this prediction, a threshold had to be defined, as the result of the network is a probabilistic result and not a 0 or 1. This threshold was defined so that values greater than 0.55 are 1, and the lower 0. Other values for threshold were tested, but it was this value that got the best result. The network starts with random initial weights, obtaining, whenever trained, different results due to this randomness. To prevent this from happening, the *setdemorandstream(491218382)* function has been added before the construction and training of the network.

Similar to what was done in ML models, all metrics were calculated using the confusion matrix, in order to evaluate the model. The result for this network is also found in Table 4.1, where it is then compared and analyzed in relation to the other models.

Deep Neural Network

In this case, to train a Deep Neural Network (DNN), the target must be categorical, so the first step was to transform the target into categorical. This was done via the *convertvars()* function. This function receives as parameters the ta-

ble where the data is stored, the name of the target and what the target is to be transformed into, in this case categorical. It should be noted that for this model, standardized data were not used, as they will be standardized when the network is built.

Then, the data was divided into three different tables, keeping the same percentage of division made in the Pattern Recognition Network:

- *tblTrain* – used for training, 70% of data;
- *tblValidation* = used for validation, 15% of data;
- *tblTest* – used for the model test, 15% of the data.

The next step is to build the network. This network consists of 6 layers, which are:

- *featureInputLayer* – A feature input layer inputs feature data to a network and applies data normalization. This case has 2 features, speed and torque, and the same normalization performed for the other models, the z-score, was applied.
- *fullyConnectedLayer* – A fully connected layer multiplies the input by a weight matrix and then adds a bias vector. In this case, the outputsize was set to 20.
- *reluLayer* – activation layer, all values less than 0 this layer sets to 0.
- *fullyConnectedLayer* – another fully connected layer, but this time with an output size of 2, which is the number of target classes.
- *softmaxLayer* - A softmax layer applies a softmax function to the input.
- *classificationLayer* - A classification layer computes the cross-entropy loss for classification and weighted classification tasks with mutually exclusive classes.

After defining the constitution of the network, some training options were defined, such as:

- Solving algorithm - *adam*;
- Resolution environment - CPU;
- Maximum Epochs- 20;
- Initial Learn Rate - 0.01;

- Validation Data - *tblvalidation*;
- Validation Frequency - 100.

The network training was performed through the *trainNetwork(features,layers, options)* function. This function trains a neural network for feature classification using the feature data and responses specified by features. The features parameters of this function are our training data, *tbltrain*, the layers, the constitution of the network and the options, the previously defined training options. The trained net is stored in the *net_5* variable.

In Figure 4.12, the network training-process is illustrated. On the right side, the defined training options can be observed, as well as the result of accuracy in relation to the validation data. As for the plots, the top one represents the status of the network when it is being trained and the bottom plot represents the network loss.

Figure 4.12: Deep Neural Network Training

As with all other models, after trainning the model, the model was tested. The result was stored in the variable *YPred* and then compared with the real values, stored in the variable *tblTest*.

4.3.3 Results

As already explained in section 2. there are several metrics to evaluate the model. In the following Table 4.1, the metrics of all models explained above are calculated.

Table 4.1: Metric Results

Models	Accuracy	Precision	Recall	F1 score
NB	67.58%	-	-	-
KNN	83.15%	54.8%	88.90%	67.80%
DT	85.57%	70.0%	82.70%	75.80%
SVM	67.58%	-	-	-
PRNN	86.39%	83.75%	72.30%	78.00%
DNN	80.30%	97.30%	38.90%	55.60%

Looking at one metric at a time, accuracy is the most intuitive performance metric and is simply a ratio of correctly predicted observation to total observations. The greater the accuracy, the better, but only when the data is balanced, which is not the case with this project, hence the need to use other evaluation metrics and not just this one. For this metric, the PRNN model is more accurate, followed by DT, KNN, then DNN, and finally NB and SVM, both with the same accuracy.

Regarding precision, this metric is the proportion of positive observations correctly predicted to the total of positive observations predicted. The high precision is related to the low rate of false positives, in this case of all classified events that ignition failure occurred, which of these actually occurred. In this metric, what achieved the best result were the neural networks, with the DNN being the most accurate, 97.3%, and the PRNN being the second with the most precision, 83.75%.

Then, recall consists of the proportion of positive observations correctly predicted to all observations of the real class - misfire (0) event occurred. The question to be asked to interpret this metric is: of all the misfire events that occurred, how many were labeled? The models that achieved a better result in this metric were the DT and the KNN, respectively. DNN had the worst performance in relation to this metric with only 38.9% recall.

F1 means that it is a combination of two metrics, precision and recall, so this metric takes into account both false positives and false negatives. This metric can be a better metric for evaluating models where data is not balanced, as is the case in this project. As can be seen in Table 4.1, the PRNN model is the one with the best result according to this metric, followed by the DT model.

Finally, after analyzing all the metricsit can be concluded that, in general, the model with the best performance is the PRNN model, as this has the best result in the F1 score metric, and this metric is the most important, as referred previously, due to the data not being balanced, as mentioned above. Apart from that, this one also has very good results in the other metrics. Another important

factor in choosing this model was that it is much easier to implement when compared to the second best model like the DT. It can be seen from Table 4.1 that the NB and SVM models do not have all the calculated metrics, as they were not able to identify any misfire event.

The implementation of the chosen model, PRNN, is described in the next section.

4.4 Deploy Model

As mentioned in the previous section, the model selected was the Pattern Recognition Network. Exporting to Simulink just use the *gensim(A)* function, where A is the trained model, in this case the model trained on the variable *net*. After obtaining it, generate the model in simulink as shown in Figure 4.13.

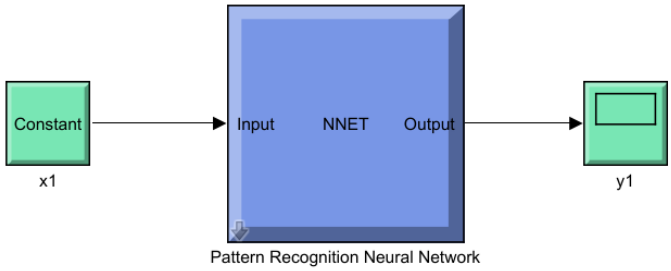


Figure 4.13: Pattern Recognition Neural Network Model

When opening the PRNN simulink block, it is possible to observe the network design in Simulink environment, Figure 4.14.

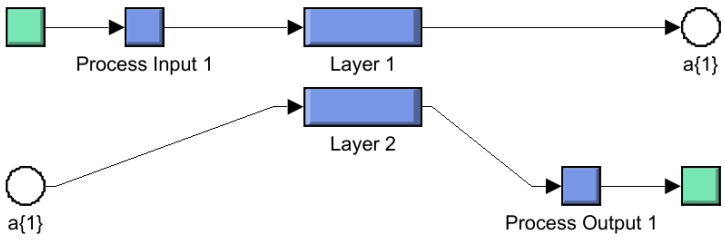


Figure 4.14: PRNN Design Simulink

As explained previously, pre-processing is done within the network, so it is not necessary to add any more pre-processing blocks to the model. You can

observe the pre-processing that is done to the model inputs when clicking on the Process Input 1 block (Figure 4.14) and later in the blue block, which corresponds to the "mapminmax" normalization explained above, Figure 4.15.

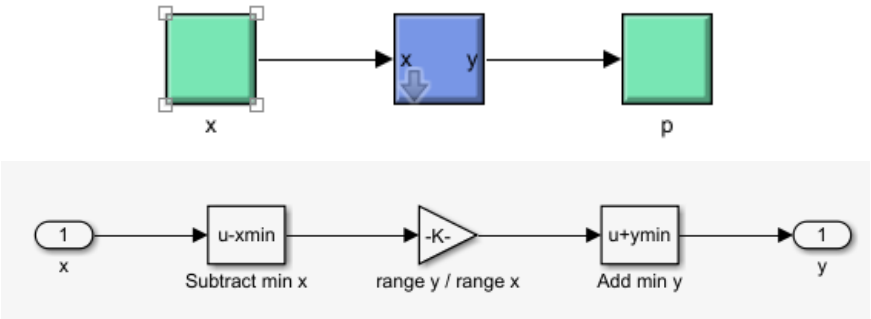


Figure 4.15: Input normalization in Simulink

The same does not happen with the threshold defined above, as this is performed outside the network, so it was necessary to add a post-processing block to the model in Simulink. The block used was the "Compare to Constant", and it was added inside the Process Output 1 block (Figure 4.14) of the network. In the following Figure 4.16 it is possible to observe that.

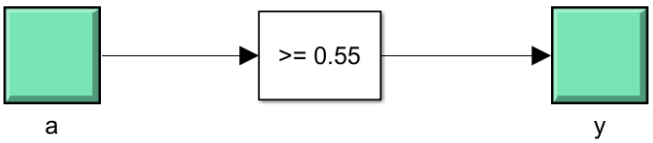


Figure 4.16: Threshold in Simulink

After finalizing the model in Simulink, the last step was performed - generate the C code with dSpace TargetLink. For this, CES defined all the necessary template configurations for the C code to be generated. An excerpt of C code is found in Appendix C.

4.5 Conclusion of the Chapter

In this chapter, all the steps performed were shown: data exploration, preparation and preprocess data, model building and lastly the model export. It is concluded that the best model, of the developed ones, was the PRNN model, as it obtained the best result from the calculated metrics. It was also proven that

data processing is the most crucial step in the development of a ML model, as the performance of the model varies depending on the data treatment.

Chapter 5

Comparison Between Models

This chapter compares all models: The PRNN model in Simulink environment, the same in TargetLink and the CES model. Two points to be highlighted in relation to this comparison: unlike what was done before, the outputs were toggled in order to make it easier to interpret, that is, 1 when there is misfire and 0 not. In addition, the comparison between the models was divided into three tests: one where no misfire occurred, another where it always occurred and another where it occurred every 3 engine cycles. These tests were carried out similarly to the organization of data described in section 3.1.

The results are constituted by five plots, where the first two are the 2 features used for the model, speed and torque respectively, and the other three the PRNN model in Simulink, PRNN model in TargetLink and the CES model.

5.1 Test 1

Figure 5.1 represents the result for a dataset where no misfire occurs. As can be seen from the figure, the behavior between the Simulink model and the TargetLink model is the same as it was supposed. Compared to the CES model, this one behaves quite well, however the PRNN model detects 3 misfires where none exists. This is due to the large speed peaks that occur at these instants.

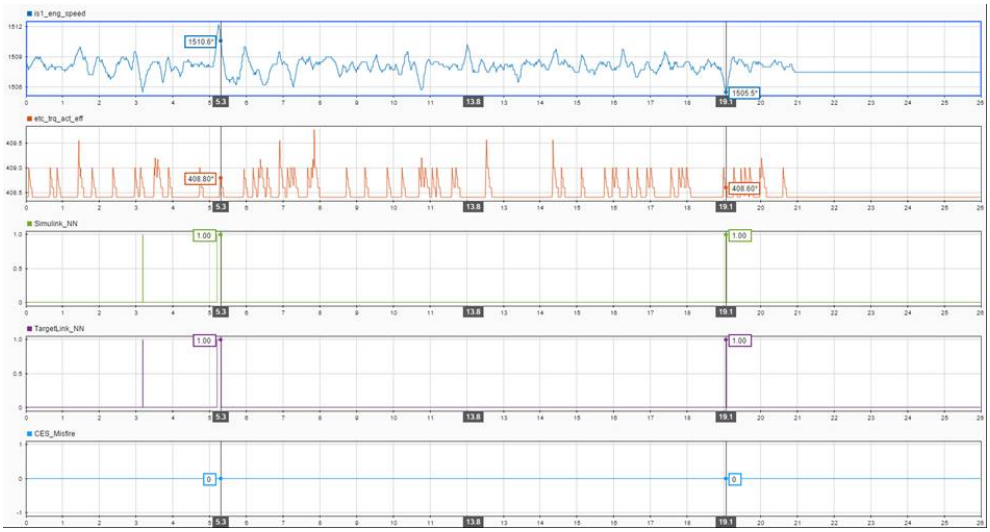


Figure 5.1: Test 1 - no misfire

5.2 Test 2

In Figure 5.2 it is shown the plots for the second test case - where the misfire event always occurs. As can be seen from the figure, both Simulink and TargetLink models have the same behavior. Furthermore, compared to the CES model, the PRNN model performs excellently, identifying all misfires correctly, as it was supposed.

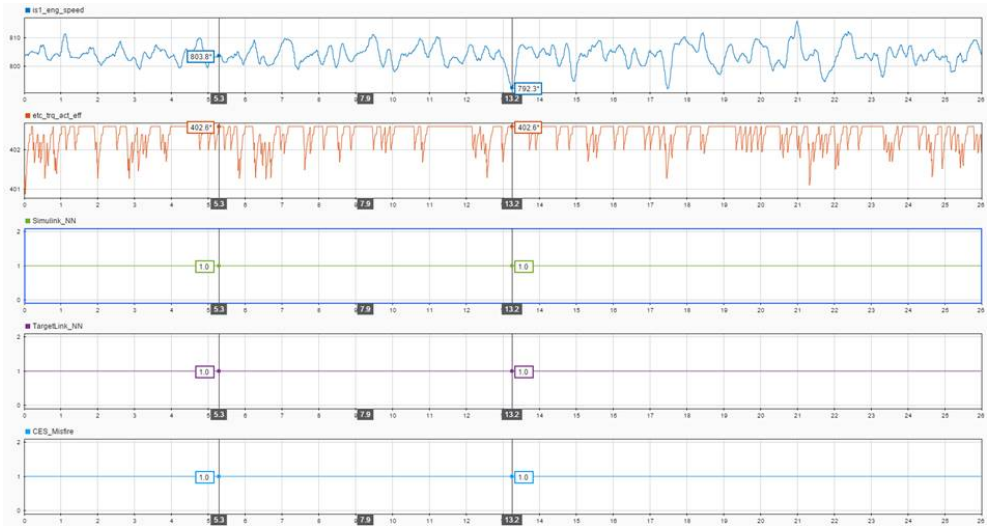


Figure 5.2: Test 2 - always misfire

5.3 Test 3

Finally, in Figure 5.3, the plots for the third test case are represented - misfire event occurs every 3 engine cycles. Once again, the behavior of the PRNN model in Simulink and TargetLink is the same, as can be seen in the figure. Compared to the CES model, this one performs a bit erratic. Although the model clearly identifies the misfire event, this identifies more events than supposed, instead of identifying one event every 3 cycles, sometimes the model identifies more than one. These lower performance compared to other test cases, may be related to how the network was designed, while the CES model detects misfire cyclically with a constant period, the PRNN model detects misfire without apparent periodicity.

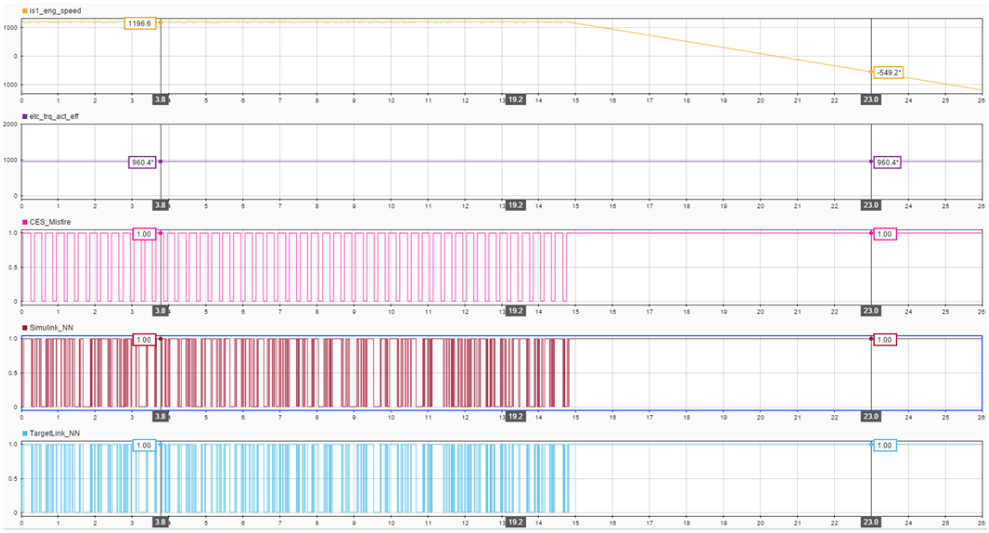


Figure 5.3: Test 3 - misfire every 3 engine cycles

5.4 Conclusion of the Chapter

From the results obtained, it can be seen that the developed model has a good performance. Of all the analyzed cases, the one that had the best performance is in test case 2, where it always occurs misfire. In this case, the model always detected the misfire event, then having an excellent performance.

In the case of test 1, the model also has a very good performance, despite having identified 3 false misfire events, justified due to speed peaks, inducing the model to error. The model having identified these false positives is not problematic, as it is better to detect when it does not happen than not to detect it when it really exists, as the damage is much more serious when this happens.

Finally, in the last test case, as mentioned, the model identifies well when misfires occur, but, due to not considering the time variable, it can identify more misfires than supposed, that is, it identifies more than one misfire each 3 cycles, and the assumption was to identify only 1 every 3 cycles. But again, this false detection of the ignition event, similar to test case 1, is not problematic because the real problem was if the misfire actually happened, and the model did not detect it.

Chapter 6

Conclusion

With this work, it was shown the versatility that solutions based on ML can adopt, in this case in the automotive industry for the detection of misfire events.

MATLAB has turned out to be a very useful software for problems of this kind. With its strong evolution over time, MATLAB, through its Machine Learning and Deep Learning toolboxes, becomes a very advantageous option, as in addition to having these toolboxes, it is very intuitive to use. This software not only has several models and algorithms, it is also possible to solve different types of ML problems, not only supervised learning probabilities as is the case in this project. Furthermore, this software is also very good for data processing.

After building the models, the one that obtained the best result according to the calculated metrics was the PRNN model. This model achieved an accuracy of 86.39%, a precision of 83.75%, a recall of 82.70% and an F1 score of 75.80%. Compared to the CES model, the model performs very well. It performs excellently in test case 1 and 2. In test case 3, the model also performs well as it clearly detects when the misfire event occurs, but it was supposed to detect a misfire every 3 cycles and this detects more than one. This is due to the way the network was conceived, that is, while in the CES model the variable "time" is considered, in the developed model it is not. This has the consequence that while the CES model cyclically detects misfire with a constant period, the network detects misfire without apparent periodicity. However, as mentioned, this network presents an accuracy of 86,39%, and, taking into account the variable "time", this result can be improved.

In terms of future developments, as ML is an iterative area, we will try to continue to improve the developed model. If it is not possible to improve, another approach can be taking into account the variable "time". For this, we can use an LSTM network, or a CNN network.

Finally, it can be concluded that the work was carried out successfully, as all objectives were met. The model developed achieved very satisfactory results, with a possible margin for improvement, as mentioned above. In addition, this model brings some advantages over the CES model, such as making the model less dense and easier to calibrate. In the model developed by CES there were maps and calibration tables for the input values, while with the model developed, it is no longer necessary. It is enough to insert the "raw" values and the network takes care of doing everything. In other words, in terms of memory, the PRNN model is less demanding than the CES model because these calibration maps and tables are no longer needed.

References

[1] CarMD, "Vehicle health index-2019." https://www.carmd.com/wp/vehicle-health-index-introduction/2019-carmd-vehicle-health-index/. Accessed: 2021-07-03. [Quoted on p. iii, 1, 2]

[2] C. E. Services, "Company." https://conti-engineering.com/company/. Accessed: 2021-07-03. [Quoted on p. iii, 2, 3]

[3] A. Mair, "Misfire detection for internal combustion aircraft engines." https://diglib.tugraz.at/download.php?id=58132a9e514de&location=browse. Accessed: 2021-07-03. [Quoted on p. iii, 2, 7, 8, 9, 10]

[4] M. Terry-Jack, "Tips and tricks for multi-class classification." https://medium.com/@b.terryjack/tips-and-tricks-for-multi-class-classification-c184ae1c8ffc. Accessed: 2021-07-03. [Quoted on p. iii, 13, 14]

[5] A. Sharma, "Decision tree vs. random forest – which algorithm should you use?." https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/. Accessed: 2021-07-03. [Quoted on p. iii, 16]

[6] ICHI.PRO, "Máquina de vetores de suporte (svm) explicada." https://ichi.pro/pt/maquina-de-vetores-de-suporte-svm-explicada-97743104690915. Accessed: 2021-07-03. [Quoted on p. iii, 17]

[7] R. Keim, "How to train a basic perceptron neural network." https://www.allaboutcircuits.com/technical-articles/how-to-train-a-basic-perceptron-neural-network/. Accessed: 2021-07-03. [Quoted on p. iii, 18]

[8] J. Djuris, D. Medarević, M. Krstić, I. Vasiljević, I. Aleksić, and S. Ibrić, "Design space approach in optimization of fluid bed granulation and tablets compression process." https://www.researchgate.net/publication/230735806_Design_Space_Approach_in_Optimization_of_Fluid_Bed_Granulation_and_Tablets_Compression_Process/figures?lo=1, Accessed: 2021-07-03. [Quoted on p. iii, 18]

[9] D. Martín, "Why we split." http://mateos.io/blog/train-test-split/. Accessed: 2021-07-03. [Quoted on p. iii, 21]

[10] M. Yatsenko and V. Zh., "Using artificial intelligence in the automotive industry: 6 key applications for a competitive advantage." https://www.apriorit.com/dev-blog/728-ai-applications-automotive-industry. Accessed: 2021-07-03. [Quoted on p. iii, 23, 24, 25, 26, 27]

[11] Microsoft, "Blue yonder factory planning and scheduling jda software - global." https://appsource.microsoft.com/fr-fr/product/web-apps/jdasoftware-global.by_factory_planning_and_scheduling?tab=Overview. Accessed: 2021-07-03. [Quoted on p. iii, 25]

[12] NewVoice, "Veículos da bmw ganham integração com a alexa." https://newvoice.ai/2020/12/03/veiculos-da-bmw-ganham-integracao-com-a-alexa/. Accessed: 2021-07-03. [Quoted on p. iii, 26]

[13] NAUTO, "Nauto mark button." https://www.lilin-design.com/spellbound-vrdatingexperiencedesign. Accessed: 2021-07-03. [Quoted on p. iii, 27]

[14] dSpace, "Targetlink." https://www.dspace.com/en/inc/home/products/sw/pcgs/targetlink.cfm#176_25807. Accessed: 2021-07-03. [Quoted on p. iii, 35, 36]

[15] O. F.Y., A. J.E.T, A. O., H. J. O., O. O., and A. J., "Supervised machine learning algorithms: Classification and comparison." https://www.researchgate.net/profile/J-E-T-Akinsola/publication/318338750_Supervised_Machine_Learning_Algorithms_Classification_and_Comparison/links/596474ae0f7e9b819497e053/Supervised-Machine-Learning-Algorithms-Classification-and-Comparison.pdf. Accessed: 2021-07-03. [Quoted on p. v, 15, 19]

[16] J. Siegel, S. Kumar, I. Ehrenberg, and S. Sarma, "Engine misfire detection with pervasive mobile audio." https://www.researchgate.net/publication/320616264_Engine_misfire_detection_with_pervasive_mobile_audio. Accessed: 2021-07-03. [Quoted on p. 1, 2]

[17] A. Sharmaa, V. Sugumaranb, and S. Devasenapatic, "Misfire detection in an ic engine using vibration signal and decision tree algorithms." https://www.sciencedirect.com/science/article/pii/S0263224114000244. Accessed: 2021-07-03. [Quoted on p. 1]

[18] F. Lo Bue, A. Di Stefano, C. Giaconia, and E. Pipitone, "Misfire detection system based on the measure of crankshaft angular velocity." https://link.springer.com/content/pdf/10.1007%2F978-3-540-71325-8_12.pdf. Accessed: 2021-07-03. [Quoted on p. 9, 10]

[19] J. Merkisz, P. Bogus, and R. Grzeszczyk, "Overview of engine misfire detection methods used in on board diagnostics." https://ilot.lukasiewicz.gov.pl/KONES/2001/JOK2001%20NO%201-2/R39.pdf. Accessed: 2021-07-03. [Quoted on p. 9, 10]

[20] J. Alzubi, A. Nayyar, and A. Kumar, "Machine learning from theory to algorithms: An overview." https://iopscience.iop.org/article/10.1088/1742-6596/1142/1/012012/pdf. Accessed: 2021-07-03. [Quoted on p. 10, 11]

[21] M. Mohri, A. Rostamizadeh, and A. Talwalkar, "Foudations of machine learning." https://d1rkab7tlqy5f1.cloudfront.net/EWI/Over%20de%20faculteit/Afdelingen/Intelligent%20Systems/Pattern%20Recognition%20Laboratory/PR/Reading%20Group/Foundations_of_Machine_Learning.pdf. Accessed: 2021-07-03. [Quoted on p. 11]

[22] S. Mankad, "Machine learning basics for noobs." https://www.opensourceforu.com/2018/10/machine-learning-basics-for-noobs/. Accessed: 2021-07-03. [Quoted on p. 11, 12, 13, 14, 15]

[23] F. M. Quintão Mateus and M. d. Carvalho Mendonça, "Machine learning na melhoria de processos internos: Estudos de caso na indústria de varejo brasileira." http://repositorio.poli.ufrj.br/monografias/monopoli10031889.pdf. Accessed: 2021-07-03. [Quoted on p. 12, 20, 21, 22, 23]

[24] A. J. da Cunha Carneiro Lima, "Aplicação de aprendizagem de máquina no diagnóstico de declínio cognitivo e demência de alzheimer baseado em teste cognitivos e marcadores genéticos." http://repositorio.poli.ufrj.br/monografias/monopoli10031889.pdf. Accessed: 2021-07-03. [Quoted on p. 12, 13, 14, 15]

[25] J. Brownlee, "One-vs-rest and one-vs-one for multi-class classification." https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/. Accessed: 2021-07-03. [Quoted on p. 13]

[26] F. A. P. de Figueiredo, "Inteligência artificial e machine learning: Introdução." https://www.researchgate.net/publication/339777203_TP555_-_Introducao_a_Inteligencia_Artificial_e_Machine_Learning. Accessed: 2021-07-03. [Quoted on p. 14, 15]

[27] A. Singh, N. Thakur, and A. Sharma, "A review of supervised machine learning algorithms." https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7724478. Accessed: 2021-07-03. [Quoted on p. 15]

[28] AltexSoft, "Machine learning project structure: Stages, roles, and tools." https://www.altexsoft.com/blog/datascience/machine-learning-project-structure-stages-roles-and-tools/. Accessed: 2021-07-03. [Quoted on p. 20, 21, 23]

[29] P. D. B. Azevedo, "Aplicações de big data e algoritmos de machine learning à gestão inteligente da rega." https://www.repository.utl.pt/bitstream/10400.5/17896/1/Tese_PA_MEA.pdf. Accessed: 2021-07-03. [Quoted on p. 21]

[30] Nvidia, "Nvidia quadro rtx 8000." https://www.nvidia.com/en-us/design-visualization/quadro/rtx-8000/. Accessed: 2021-07-03. [Quoted on p. 24]

[31] M. Tomorrow, "Collaborative robots working in manufacturing." https://www.manufacturingtomorrow.com/article/2016/02/collaborative-robots-working-in-manufacturing/7672/. Accessed: 2021-07-03. [Quoted on p. 24]

[32] H. Mian, "4 machine learning use cases in the automotive sector." https://www.anaconda.com/blog/4-machine-learning-use-cases-automotive. Accessed: 2021-07-03. [Quoted on p. 24, 25]

[33] G. Rice, "Carvi provides real-time driving assistance." https://www.newegg.com/insider/carvi-real-time-smart-driver-safety-aide/. Accessed: 2021-07-03. [Quoted on p. 26]

[34] D. Graham, "Waymo - evolution of the self-driving car." https://primerhub.com/self-driving-cars/waymo. Accessed: 2021-07-03. [Quoted on p. 26]

[35] G. Ciaburro, "Matlab for machine learning." . Accessed: 2021-07-03. [Quoted on p. 33, 34]

[36] MathWorks, "Statistics and machine learning toolbox." https://www.mathworks.com/products/statistics.html. Accessed: 2021-07-03. [Quoted on p. 34]

[37] MathWorks, "Deep learning toolbox." `https://www.mathworks.com/products/deep-learning.html`. Accessed: 2021-07-03. [Quoted on p. 35]

Appendix A

MATLAB Code

```
/*DATA PREPARATION
Vertical Concatenation of Files*/

data =
    vertcat(x800rpm400Nm,x800rpm400Nmcyl13cycle,x800rpm400Nmcyl1allcycle,
x1200rpm200Nm,x1200rpm200Nmcyl13cycle,x1200rpm200Nmcyl1allcycle,
x1200rpm960Nm, x1200rpm960Nmcyl13cycle, x1200rpm960Nmcyl1allcycle,
    x1500rpm400Nm, x1500rpm400Nmcyl13cycle, x1500rpm400Nmcyl1allcycle);

dataset = data(:,
    ["fis_act_fm_combustion_bank1","fis_act_fm_combustion_bank2,
"is1_eng_speed","etc_trq_act_eff","mfd_cyc_based_non_stationary_cond_b",
"mfd_amp_no_avg_dist_mean_cyl1"]);

/*Summary of variables (Min. Max. and Average */

summary(dataset)

/* Target creation in relation to thereshold: when <-4, misfire (0) if
    not (1) */

for i=1:length(dataset.mfd_amp_no_avg_dist_mean\_cyl1)
    if dataset.mfd_amp_no_avg_dist_mean_cyl1(i) < -4
        misf(i)=0;
    else
         misf(i)=1;
    end
end
```

```
misfire_amp = misf(:);

/* Cleaning and Pre-processing
Check for missing values */

bank1_miss= sum(ismissing(dataset.fis_act_fm_combustion_bank1));
bank2_miss= sum(ismissing(dataset.fis_act_fm_combustion_bank1));
is1_speed_miss= sum(ismissing(dataset.is1_eng_speed));
trq_act_miss= sum(ismissing(dataset.etc_trq_act_eff));
stationay_miss=
    sum(ismissing(dataset.mfd_cyc_based_non_stationary_cond_b));

/* Outliers Verification */

bank1_out= sum(isoutlier(dataset.fis_act_fm_combustion_bank1,"mean"));
bank2_out= sum(isoutlier(dataset.fis_act_fm_combustion_bank2,"mean"));
is1_speed_out= sum(isoutlier(dataset.is1_eng_speed,"mean"));
etc_trq_out= sum(isoutlier(dataset.etc_trq_act_eff,"mean"));
stationay_out=
    sum(isoutlier(dataset.mfd_cyc_based_non_stationary_cond_b,"mean"));

/* Fix the outliers */

[mfd_cyc_based_non_stationary_cond_b,outlierIndices2,thresholdLow,
thresholdHigh] = ...
    filloutliers(dataset.mfd_cyc_based_non_stationary_cond_b,'spline',
    'mean');


/* Display results */

clf
plot(dataset.mfd_cyc_based_non_stationary_cond_b,'Color',[109 185
    226]/255,...
    'DisplayName','Input data')
hold on
plot(mfd_cyc_based_non_stationary_cond_b,'Color',[0 114
    189]/255,'LineWidth',1.5,...
    'DisplayName','Cleaned data')
% Plot outliers
plot(find(outlierIndices2),data.mfd_cyc_based_non_stationary_cond_b
(outlierIndices2),'x',...
    'Color',[64 64 64]/255,'DisplayName','Outliers')
title(['Number of outliers: ' num2str(nnz(outlierIndices2))])
% Plot outlier thresholds
plot([xlim missing xlim],[thresholdLow*[1 1] NaN thresholdHigh*[1 1]],...
    'Color',[145 145 145]/255,'DisplayName','Outlier thresholds')
```

```
hold off
legend

dataset.mfd_cyc_based_non_stationary_cond_b =
    mfd_cyc_based_non_stationary_cond_b;
stationay_out2=
    sum(isoutlier(dataset.mfd_cyc_based_non_stationary_cond_b,"mean"));

/* Correlation between data */

plot(dataset.etc_trq_act_eff)
plot(dataset.fis_act_fm_combustion_bank1)
VariableNames
    ={'fis_act_fm_combustion_bank1','fis_act_fm_combustion_bank2',
'is1_eng_speed','etc_trq_act_eff'};

corrplot([dataset.fis_act_fm_combustion_bank1,dataset.fis_act_fm
_combustion_bank2, dataset.is1_eng_speed,dataset.etc_trq_act_eff],
'varnames',VariableNames)

/* High Correlation between (banki_1, bank_2 e trq)
Data normalization */

dataset_v1 = table(dataset.etc_trq_act_eff, dataset.is1_eng_speed);
dataset_v1.Properties.VariableNames= ["etc_trq_act_eff","is1_eng_speed"];
dataset_v2 = normalize(dataset_v1);
misf_ampT = table(misfire_amp);
dataset_v3 = [dataset_v2,misf_ampT(:,:)];
```

```
/*DATA DIVISION
HoldOut */

cv = cvpartition(size(dataset_v3,1),'HoldOut',0.2);
datatrain = dataset_v3(cv.training,:);
datatest = dataset_v3(cv.test,:);

Xtrain = datatrain(:,1:2);
Ytrain = datatrain(:,3);
Xtest= datatest(:,1:2);
Ytest = datatest(:,3);
```

```
/* DECISION TREES */

dtc = fitctree(Xtrain,Ytrain)
y_predict_dtc = predict(dtc, Xtest)
```

```
confusionchart(Ytest.misfire_amp, y_predict_dtc)
```

```
/* K NEAREST NEIGHBOUR */

knnc = fitcknn(Xtrain,Ytrain)
y_predict_knn = predict(knnc, Xtest)
confusionchart(Ytest.misfire_amp, y_predict_knn)
```

```
/* NAIVE BAYES */

nbc = fitcnb(Xtrain,Ytrain)
y_predict_nb = predict(nbc, Xtest)
confusionchart(Ytest.misfire_amp, y_predict_nb)
```

```
/* SUPPORT VECTOR MACHINE */

svmc = fitcsvm(Xtrain,Ytrain)
y_predict_svmc = predict(svmc, Xtest)
confusionchart(Ytest.misfire_amp, y_predict_svmc)
```

```
/* PATTERN RECOGNITION NETWORK

Data */

dat = [dataset_v1, misf_ampT]
x1 = dat(:,1:2);
x_v1 = table2array(x)
y = dat(:,3)
y_v1 = table2array(y)
xh = x_v1';
y_t = y_v1';

/* Net */

setdemorandstream(491218382);

hiddenLayerSize = 10;
net = patternnet(hiddenLayerSize);

net.trainFcn = 'trainbr';
net.input.processFcns = {'mapminmax'};
net.layers{2}.transferFcn = 'softmax';
net.divideFcn = 'dividerand';
net.dividemode = 'sample';
```

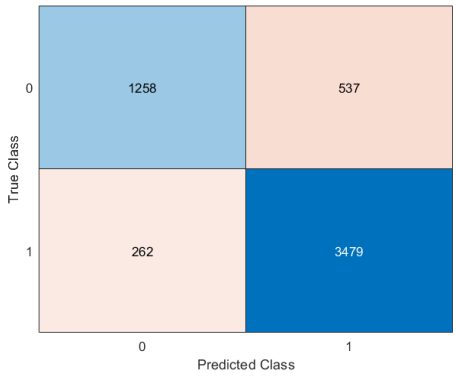
```
net.divideParam.trainRatio = 75/100;
net.divideParam.valRatio = 10/100;
net.divideParam.testRatio = 15/100;
net.sampleTime = 0.01;
net.plotFcns = {'plotperform','plottrainstate','ploterrhist',
    'plotconfusion'}

/* Train Model */

trtest = xh(:,tr.testInd);
[net,tr]=train(net,xh,y_t)

/* Confusion Matrix */

OutputData = net(trtest)
llogical_output = zeros(1,4152);
logical_output(OutputData >= 0.55) = 1

tsTarg = y_t (:,tr.testInd);
confusionchart(logical_output, tsTarg)
```

```
/* DEEP NEURAL NETWORK

Data */

misf_ampT = table(misfire_amp);
tbl1 = [dataset_v1, misf_ampT(:,:)]
labelname = "misfire_amp";
tbl1 = convertvars(tbl1,labelname,'categorical');
tbl1 = splitvars(tbl1);
head(tbl1)
classNames = categories(tbl1{:,labelname})

/* Data Split */

numObservations = size(tbl1,1);
numObservationsTrain = floor(0.7*numObservations);
numObservationsValidation = floor(0.15*numObservations);
numObservationsTest = numObservations - numObservationsTrain -
    numObservationsValidation;

idx = randperm(numObservations);
idxTrain = idx(1:numObservationsTrain);
idxValidation =
    idx(numObservationsTrain+1:numObservationsTrain+numObservationsValidation);
idxTest = idx(numObservationsTrain+numObservationsValidation+1:end);
```

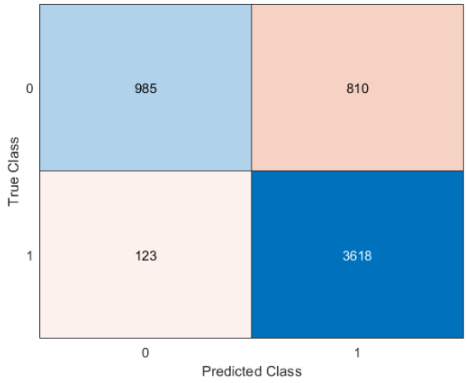
```
tblTrain = tbl1(idxTrain,:);
tblValidation = tbl1(idxValidation,:);
tblTest = tbl1(idxTest,:);

/* Network Design */

numFeatures = size(tbl1,2) - 1;
numClasses = 2;

layers = [
    featureInputLayer(numFeatures,'Normalization', 'zscore')
    fullyConnectedLayer(20)
    reluLayer
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];

minibatchsize =156;
maxepochs = 20;
vf = 100;

options = trainingOptions('adam', ...
    'ExecutionEnvironment','cpu', ...
    "MaxEpochs", maxepochs, ...
    "MiniBatchSize", minibatchsize, ...
    'Shuffle',"every-epoch", ...
    "InitialLearnRate", 0.01, ...
    "ValidationFrequency", vf, ...
    'ValidationData',tblValidation, ...
    'Plots','training-progress', ...
    'Verbose',false);

/*Train */

net_5 = trainNetwork(tblTrain,layers,options);

/* Confusion Matrix */

YPred = classify(net_5,tblTest);
YTests = tblTest{:,labelname};
confusionchart(YPred, YTests)
```

Appendix B

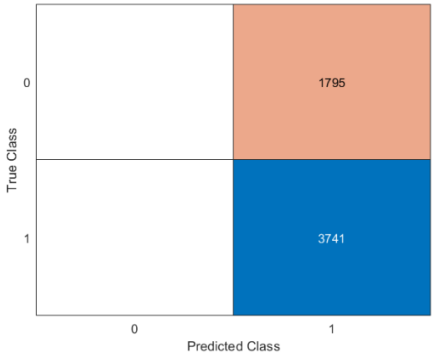
Confusion Matrices



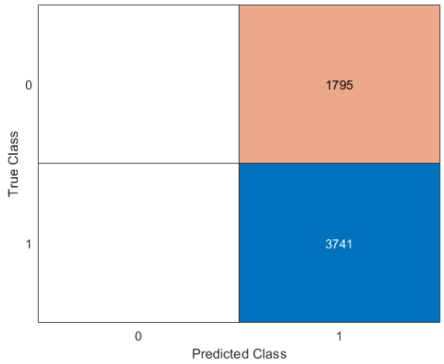
(a) Decision Trees



(b) K Nearest Neighbour

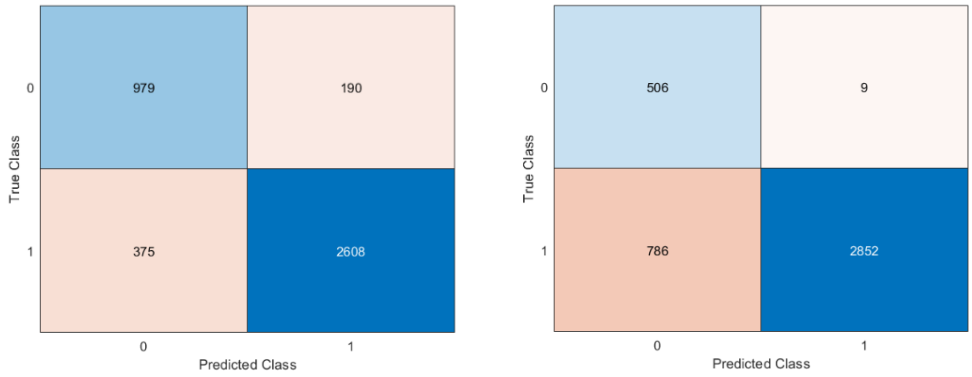


(c) Support Vector Machine



(d) Naive Bayes

Figure B.1: ML Models' Confusion Matrices



(a) Pattern Recognition Neural Network

(b) Deep Neural Network

Figure B.2: DL Models' Confusion Matrices

Appendix C

C Code

```
/*********************************************************************\
 ***
 *** Simulink model     : NeuralNetworkTL_frame
 *** TargetLink subsystem : NeuralNetworkTL_frame/NeuralNet
 *** Codefile           : NeuralNet.c
 ***
 *** Generated by TargetLink, the dSPACE production quality code
     generator
 *** Generation date: 2021-07-09 16:18:49
 ***
 *** CODE GENERATOR OPTIONS:
 *** Code generation mode                 : Standard
 *** Compiler                             : <unknown>
 *** Target                               : Generic
 *** ANSI-C compatible code               : yes
 *** Code Optimization                    : enabled
 *** Constant style                       : decimal
 *** Clean code option                    : disabled
 *** Logging mode                         : According to block-specific
     data
 *** Code Coverage                        : disabled
 *** Generate empty conditional branches  : disabled
 *** Loop unroll threshold                : 5
 *** Shift mode                           : automatic
 *** Handle unscaled SF expr. with TL type : enabled
 *** Assignment of conditions             : AllBooleanOutputs
 *** Scope reduction only to function level : disabled
 *** Exploit ranges if not erasable       : disabled
 *** Exploit Compute Through Overflow     : optimized
```

```
*** Linker sections                      : enabled
*** Enable Assembler                     : disabled
*** Variable name length                 : 31 chars
*** Use global bitfields                 : disabled
*** Stateflow: use of bitfields          : enabled
*** State activity encoding limit        : 5
*** Omit zero inits in restart function  : disabled
*** Share functions between TL subsystems : disabled
*** Generate 64bit functions             : enabled
*** Inlining Threshold                   : 6
*** Line break limit                     : 100
*** Target optimized boolean data type   : enabled
*** Keep saturation elements             : disabled
*** Extended variable sharing            : disabled
*** Extended lifetime optimization       : enabled
*** Style definition file                : D:\LegacyApp\dSPACE
    TargetLink 4.3\Matlab\Tl\Config\
***                                       codegen\cconfig.xml
*** Root style sheet                     : D:\LegacyApp\dSPACE
    TargetLink 4.3\Matlab\Tl\XML\Cod
***
    eGen\Stylesheets\TL_CSourceCodeSS.xsl
***
*** SUBSYS       CORRESPONDING SIMULINK SUBSYSTEM
*** SNNet1       NeuralNet
*** SNNet2       NeuralNet/NeuralNetwork
*** SNNet3       NeuralNet/NeuralNetwork/Layer 1
*** SNNet4       NeuralNet/NeuralNetwork/Layer 2
*** SNNet5       NeuralNet/NeuralNetwork/Process Input 1
*** SNNet6       NeuralNet/NeuralNetwork/Process Output 1
*** SNNet7       NeuralNet/NeuralNetwork/Layer 1/Delays 1
*** SNNet8       NeuralNet/NeuralNetwork/Layer 1/IW{1,1}
*** SNNet9       NeuralNet/NeuralNetwork/Layer 1/tansig
*** SNNet10      NeuralNet/NeuralNetwork/Layer 1/IW{1,1}/DotProdMat
*** SNNet11      NeuralNet/NeuralNetwork/Layer 1/IW{1,1}/DotProdMat1
*** SNNet12      NeuralNet/NeuralNetwork/Layer 1/IW{1,1}/DotProdMat2
*** SNNet13      NeuralNet/NeuralNetwork/Layer 1/IW{1,1}/DotProdMat3
*** SNNet14      NeuralNet/NeuralNetwork/Layer 1/IW{1,1}/DotProdMat4
*** SNNet15      NeuralNet/NeuralNetwork/Layer 1/IW{1,1}/DotProdMat5
*** SNNet16      NeuralNet/NeuralNetwork/Layer 1/IW{1,1}/DotProdMat6
*** SNNet17      NeuralNet/NeuralNetwork/Layer 1/IW{1,1}/DotProdMat7
*** SNNet18      NeuralNet/NeuralNetwork/Layer 1/IW{1,1}/DotProdMat8
*** SNNet19      NeuralNet/NeuralNetwork/Layer 1/IW{1,1}/DotProdMat9
*** SNNet20      NeuralNet/NeuralNetwork/Layer 2/Delays 1
*** SNNet21      NeuralNet/NeuralNetwork/Layer 2/LW{2,1}
*** SNNet22      NeuralNet/NeuralNetwork/Layer 2/logsig
*** SNNet23      NeuralNet/NeuralNetwork/Process Input 1/mapminmax
```

```
 ***
 *** SUBSYS       CORRESPONDING MODEL BLOCK (REFERENCED MODEL)
 ***
 *** SF-NODE      CORRESPONDING STATEFLOW NODE               DESCRIPTION
 ***
 *** TargetLink version    : 4.3p7 from 15-Jan-2021
 *** Code generator version : Build Id 4.3.0.32 from 2021-01-14 15:11:32
\*************************************************************/

#ifndef NEURALNET_C
#define NEURALNET_C

/*-----------------------------------------------------------*\
  DEFINES (OPT)
\*-----------------------------------------------------------*/
/*-----------------------------------------------------------*\
  INCLUDES
\*-----------------------------------------------------------*/

#include <math.h>
#include "NeuralNet.h"

/*-----------------------------------------------------------*\
  ENUMS
\*-----------------------------------------------------------*/
/*-----------------------------------------------------------*\
  DEFINES
\*-----------------------------------------------------------*/
/*-----------------------------------------------------------*\
  TYPEDEFS
\*-----------------------------------------------------------*/
/*-----------------------------------------------------------*\
  VARIABLES
\*-----------------------------------------------------------*/

/*************************************************************\
   SLGlobal: Default storage class for global variables | Width: 16
\*************************************************************/
Int16 SNNet1_OutPort;
Int16 SNNet1_SpeedAct;
Int16 SNNet1_TorqueAct;

/*-----------------------------------------------------------*\
  PARAMETERIZED MACROS
\*-----------------------------------------------------------*/
/*-----------------------------------------------------------*\
  FUNCTION PROTOTYPES
```

```
\*-----------------------------------------------------------*/
/*-----------------------------------------------------------*\
  INLINE FUNCTIONS
\*-----------------------------------------------------------*/
/*-----------------------------------------------------------*\
  FUNCTION DEFINITIONS
\*-----------------------------------------------------------*/

/*************************************************************\
 ***  FUNCTION:
 ***      NeuralNet
 ***
 ***  DESCRIPTION:
 ***
 ***
 ***  PARAMETERS:
 ***      Type              Name              Description
 ***      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 ***
 ***  RETURNS:
 ***      void
 ***
 ***  SETTINGS:
 ***
\*************************************************************/
void NeuralNet(void)
{
   /* SLLocal: Default storage class for local variables | Width: 32 */
   Float32 SNNet22_Reciprocal;

   /* SLLocal: Default storage class for local variables | Width: 16 */
   Int16 SNNet22_Sum;
   Int16 SNNet23_Sum1[2];
   Int16 SNNet3_netsum[10];
   Int16 SNNet9_Sum1[10];

   /* SLLocal: Default storage class for local variables | Width: 32 */
   Float32 Aux_F32;
   Int32 Aux_S32;

   /* SLLocal: Default storage class for local variables | Width: 16 */
   Int16 Aux_S16;

   /* Sum: NeuralNet/NeuralNetwork/Process Input 1/mapminmax/Sum1
      # combined # Gain: NeuralNet/NeuralNetwork/Process Input
          1/mapminmax/range y __ range x
```

```
   # combined # Sum: NeuralNet/NeuralNetwork/Process Input
      1/mapminmax/Sum */
SNNet23_Sum1[0] = (Int16) (((Int16) ((((Int32) (Int16)
    (SNNet1_TorqueAct - 29)) * 14551) >> 19))
 - 1);
SNNet23_Sum1[1] = (Int16) (((Int16) ((((Int32) (Int16)
    (SNNet1_SpeedAct - 792)) * 22733) >> 23))
 - 1);

/* Sum: NeuralNet/NeuralNetwork/Layer 1/netsum
   # combined # Sum: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat/Sum10
   # combined # Product: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat/Product10
   # combined # Product: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat/Product1 */
SNNet3_netsum[0] = (Int16) (((Int16) (((Int16) (-((Int16) ((((Int32)
    SNNet23_Sum1[0]) * 161443)
 >> 12)))) + ((Int16) (-((Int16) ((SNNet23_Sum1[1] * 11) >> 2))))))) -
     26);

/* Sum: NeuralNet/NeuralNetwork/Layer 1/netsum
   # combined # Sum: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat1/Sum10
   # combined # Product: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat1/Product10
   # combined # Product: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat1/Product1 */
SNNet3_netsum[1] = (Int16) (((Int16) (((Int16) ((((Int32)
    SNNet23_Sum1[0]) * 829883) >> 18)) +
 ((Int16) ((SNNet23_Sum1[1] * 35) >> 5)))) + 7);

/* Sum: NeuralNet/NeuralNetwork/Layer 1/netsum
   # combined # Sum: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat2/Sum10
   # combined # Product: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat2/Product10
   # combined # Product: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat2/Product1 */
SNNet3_netsum[2] = (Int16) (((Int16) ((SNNet23_Sum1[0] * 77) +
    ((Int16) (-((Int16)
 ((SNNet23_Sum1[1] * 15) >> 2)))))) + 41);

/* Sum: NeuralNet/NeuralNetwork/Layer 1/netsum
   # combined # Sum: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat3/Sum10
```

```
  # combined # Product: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat3/Product10
  # combined # Product: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat3/Product1 */
SNNet3_netsum[3] = (Int16) (((Int16) (((Int16) (-(SNNet23_Sum1[0] *
    113))) + ((Int16)
 (-(SNNet23_Sum1[1] * 197))))) + 133);

/* Sum: NeuralNet/NeuralNetwork/Layer 1/netsum
  # combined # Sum: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat4/Sum10
  # combined # Product: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat4/Product10
  # combined # Product: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat4/Product1 */
SNNet3_netsum[4] = (Int16) (((Int16) (((Int16) ((SNNet23_Sum1[0] *
    11) >> 1)) + ((Int16)
 ((((Int32) SNNet23_Sum1[1]) * 291831) >> 15)))) - 6);

/* Sum: NeuralNet/NeuralNetwork/Layer 1/netsum
  # combined # Sum: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat5/Sum10
  # combined # Product: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat5/Product10
  # combined # Product: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat5/Product1 */
SNNet3_netsum[5] = (Int16) (((Int16) (((Int16) (-(SNNet23_Sum1[0] *
    58))) + ((Int16) ((((Int32)
 SNNet23_Sum1[1]) * 305019) >> 14)))) - 19);

/* Sum: NeuralNet/NeuralNetwork/Layer 1/netsum
  # combined # Sum: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat6/Sum10
  # combined # Product: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat6/Product10
  # combined # Product: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat6/Product1 */
SNNet3_netsum[6] = (Int16) (((Int16) (((Int16) (-(SNNet23_Sum1[0] *
    102))) + ((Int16)
 (-(SNNet23_Sum1[1] * 163))))) + 118);

/* Sum: NeuralNet/NeuralNetwork/Layer 1/netsum
  # combined # Sum: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat7/Sum10
  # combined # Product: NeuralNet/NeuralNetwork/Layer
      1/IW{1,1}/DotProdMat7/Product10
```

```
   # combined # Product: NeuralNet/NeuralNetwork/Layer
       1/IW{1,1}/DotProdMat7/Product1 */
SNNet3_netsum[7] = (Int16) (((Int16) ((SNNet23_Sum1[0] * 242) +
    ((Int16) (-(SNNet23_Sum1[1] *
 368))))) - 202);

/* Sum: NeuralNet/NeuralNetwork/Layer 1/netsum
   # combined # Sum: NeuralNet/NeuralNetwork/Layer
       1/IW{1,1}/DotProdMat8/Sum10
   # combined # Product: NeuralNet/NeuralNetwork/Layer
       1/IW{1,1}/DotProdMat8/Product10
   # combined # Product: NeuralNet/NeuralNetwork/Layer
       1/IW{1,1}/DotProdMat8/Product1 */
SNNet3_netsum[8] = (Int16) (((Int16) (((Int16) ((((Int32)
    SNNet23_Sum1[0]) * 707403) >> 15)) +
 ((Int16) ((((Int32) SNNet23_Sum1[1]) * 149555) >> 12)))) - 25);

/* Sum: NeuralNet/NeuralNetwork/Layer 1/netsum
   # combined # Sum: NeuralNet/NeuralNetwork/Layer
       1/IW{1,1}/DotProdMat9/Sum10
   # combined # Product: NeuralNet/NeuralNetwork/Layer
       1/IW{1,1}/DotProdMat9/Product10
   # combined # Product: NeuralNet/NeuralNetwork/Layer
       1/IW{1,1}/DotProdMat9/Product1 */
SNNet3_netsum[9] = (Int16) (((Int16) (((Int16) (-(SNNet23_Sum1[0] *
    111))) + ((Int16)
 (-(SNNet23_Sum1[1] * 193))))) + 131);
for (Aux_S32 = 0; Aux_S32 < 10; Aux_S32++)
{
   /* Sum: NeuralNet/NeuralNetwork/Layer 1/tansig/Sum
      Vector 'SNNet9_Sum' replaced by 'Aux_S16'
      # combined # Math: NeuralNet/NeuralNetwork/Layer 1/tansig/Exp
      # combined # Gain: NeuralNet/NeuralNetwork/Layer 1/tansig/Gain */
   Aux_S16 = (Int16) (((Float32) exp((Float64) (Int16) (-((Int16)
       (SNNet3_netsum[Aux_S32] <<
    1))))) + 1.F);

   /* Math: NeuralNet/NeuralNetwork/Layer 1/tansig/Reciprocal */
   if (Aux_S16 != 0) {
      /* Vector 'SNNet9_Reciprocal' replaced by 'Aux_F32' */
      Aux_F32 = 1.F / ((Float32) Aux_S16);
   }
   else {
      /* NeuralNet/NeuralNetwork/Layer 1/tansig/Reciprocal: Numerator
          always greater than or equa
        l to zero.
        Vector 'SNNet9_Reciprocal' replaced by 'Aux_F32' */
```

```
      Aux_F32 = 3.402823466e+38F;
   }

   /* Sum: NeuralNet/NeuralNetwork/Layer 1/tansig/Sum1
      # combined # Gain: NeuralNet/NeuralNetwork/Layer 1/tansig/Gain1
         */
   SNNet9_Sum1[Aux_S32] = (Int16) (((Int16) (Aux_F32 * 2.F)) - 1);
}

/* Sum: NeuralNet/NeuralNetwork/Layer 2/LW{2,1}/Sum10
   # combined # Product: NeuralNet/NeuralNetwork/Layer
      2/LW{2,1}/Product1 */
Aux_S16 = (Int16) (-((Int16) ((((Int32) SNNet9_Sum1[0]) * 30367) >>
    10)));

/* # combined # Product: NeuralNet/NeuralNetwork/Layer
    2/LW{2,1}/Product2 */
Aux_S16 += ((Int16) ((((Int32) SNNet9_Sum1[1]) * 22091) >> 11));

/* # combined # Product: NeuralNet/NeuralNetwork/Layer
    2/LW{2,1}/Product3 */
Aux_S16 += ((Int16) (-(SNNet9_Sum1[2] * 46)));

/* # combined # Product: NeuralNet/NeuralNetwork/Layer
    2/LW{2,1}/Product4 */
Aux_S16 += ((Int16) (-(SNNet9_Sum1[3] * 53)));

/* # combined # Product: NeuralNet/NeuralNetwork/Layer
    2/LW{2,1}/Product5 */
Aux_S16 += (SNNet9_Sum1[4] * 48);

/* # combined # Product: NeuralNet/NeuralNetwork/Layer
    2/LW{2,1}/Product6 */
Aux_S16 += ((Int16) ((((Int32) SNNet9_Sum1[5]) * 343) >> 4));

/* # combined # Product: NeuralNet/NeuralNetwork/Layer
    2/LW{2,1}/Product7 */
Aux_S16 += ((Int16) (-((Int16) ((((Int32) SNNet9_Sum1[6]) * 4409) >>
    8))));

/* # combined # Product: NeuralNet/NeuralNetwork/Layer
    2/LW{2,1}/Product8 */
Aux_S16 += ((Int16) ((((Int32) SNNet9_Sum1[7]) * 25291) >> 11));

/* # combined # Product: NeuralNet/NeuralNetwork/Layer
    2/LW{2,1}/Product9 */
Aux_S16 += ((Int16) (-(SNNet9_Sum1[8] * 18)));
```

```
   /* Sum: NeuralNet/NeuralNetwork/Layer 2/logsig/Sum
      # combined # Math: NeuralNet/NeuralNetwork/Layer 2/logsig/Exp
      # combined # Gain: NeuralNet/NeuralNetwork/Layer 2/logsig/Gain
      # combined # Sum: NeuralNet/NeuralNetwork/Layer 2/netsum
      # combined # Product: NeuralNet/NeuralNetwork/Layer
          2/LW{2,1}/Product10 */
   SNNet22_Sum = (Int16) (((Float32) exp((Float64) (Int16) (-((Int16)
       (((Int16) (((UInt16) Aux_S16)
    + ((UInt16) (SNNet9_Sum1[9] * 54)))) + 18))))) + 1.F);

   /* Math: NeuralNet/NeuralNetwork/Layer 2/logsig/Reciprocal */
   if (SNNet22_Sum != 0) {
      SNNet22_Reciprocal = 1.F / ((Float32) SNNet22_Sum);
   }
   else {
      /* NeuralNet/NeuralNetwork/Layer 2/logsig/Reciprocal: Numerator
          always greater than or equal t
         o zero. */
      SNNet22_Reciprocal = 3.402823466e+38F;
   }

   /* TargetLink outport: NeuralNet/OutPort
      # combined # Logical: NeuralNet/Logical Operator1 */
   SNNet1_OutPort = (Int16) (SNNet22_Reciprocal < 0.55F);
}

/*-----------------------------------------------------------------*\
  MODULE LOCAL FUNCTION DEFINITIONS
\*-----------------------------------------------------------------*/
#endif /* NEURALNET_C */
/*-----------------------------------------------------------------*\
  END OF FILE
\*-----------------------------------------------------------------*/
```