UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE SISTEMAS ELETRÔNICOS

RAFAEL CANAL

MACHINE LEARNING APPLIED IN AUTOMOTIVE ECUS FOR REAL-TIME ENGINE BEHAVIOR ANALYSIS

DISSERTAÇÃO DE MESTRADO

Joinville
2023

Rafael Canal

MACHINE LEARNING APPLIED IN AUTOMOTIVE ECUS FOR REAL-TIME ENGINE BEHAVIOR ANALYSIS

Dissertação submetida ao Programa de Pós-Graduação em Engenharia de Sistemas Eletrônicos da Universidade Federal de Santa Catarina para a obtenção do título de Mestre em Engenharia de Sistemas Eletrônicos.
Supervisor:: Prof. Giovani Gracioli, Dr.
Co-supervisor:: Prof. Gustavo Medeiros de Araújo, Dr.

Joinville
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Canal, Rafael
Machine learning applied in automotive ECUs for real time engine behavior analysis / Rafael Canal ; orientador, Giovani Gracioli, coorientador, Gustado Medeiros de Araújo, 2023.
150 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Campus Joinville, Programa de Pós-Graduação em Engenharia de Sistemas Eletrônicos, Joinville, 2023.

Inclui referências.

1. Engenharia de Sistemas Eletrônicos. 2. Aprendizado de Máquina. 3. Análise do perfil de condução e previsão de consumo de combustível. 4. Detecção de misfire. 5. Aquisição de dados automotivos em tempo real via ECU. I. Gracioli, Giovani. II. Medeiros de Araújo, Gustado. III. Universidade Federal de Santa Catarina. Programa de Pós Graduação em Engenharia de Sistemas Eletrônicos. IV. Título.

Rafael Canal

Machine learning applied in automotive ECUs for real-time engine behavior analysis

O presente trabalho em nível de Mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Mateus Grellert da Silva, Dr.
PPGCC/UFSC

Prof. Max Mauro Dias Santos, Dr.
PPGCC/UTFPR

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de Mestre em Engenharia de Sistemas Eletrônicos.

ICP-Edu
Documento assinado digitalmente
Lucas Weihmann
Data: 09/10/2023 08:53:24-0300
CPF: ***.676.080-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Lucas Weihmann, Dr.
Coordenador do Programa

ICP-Edu
Documento assinado digitalmente
Giovani Gracioli
Data: 07/10/2023 18:45:20-0300
CPF: ***.838.060-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Giovani Gracioli, Dr.
Orientador

Joinville, July 12, 2023.

I believe that the purpose of science and technology is social and that innovation must exist for the development of the collective so that everyone has a dignified and facilitated life. Therefore, I do not dedicate this work to anyone in particular but treat it as a symbol of personal achievement, of someone who believes that making a difference should be a premise, whether in the application and development of technologies or our daily lives.

ACKNOWLEDGEMENTS

Thanks to all the members involved in the IASE project. To the professors, thank you for the guidance, advice, and attention in these last two years of work. Without this, the evolution I had during this period would not be possible and which I will certainly take into my personal, professional, and academic life. Thank you to the other members and colleagues for your dedication to their tasks. The commitment of each one made the difference for us to obtain success as a collective and in our objectives.

Furthermore, I would like to thank the team at Renault do Brasil for the constant exchange of knowledge and ideas and the Federal University of Santa Catarina for the opportunity to study in a free and quality institution. I believe that a better country can be built by joining the incentive of private initiative with the effort and human value of public universities.

This research was supported by Fundação de Desenvolvimento da Pesquisa - Fundep Rota 2030/Linha V 27192.02.01/2020.09-00.

RESUMO

A Unidade de Controle Eletrônico (ECU) centraliza a leitura de diversos sensores que adquirem dados sobre o funcionamento do motor. A leitura desses dados da ECU automotiva é capaz de gerar uma quantidade expressiva de dados. Quando monitoradas e tratadas corretamente, essas informações podem auxiliar no entendimento de padrões e problemas nos diversos subsistemas dos motores, indicando pontos que devem e podem ser melhorados. Além disso, o acompanhamento desses dados pode servir para diversos setores dentro da área, como segurança, rastreamento, redução do número de protótipos de testes e por consequência redução de custo, melhorar a interação com o motorista e ser mais uma ferramenta na mão dos engenheiros para o desenvolvimento de novas tecnologias. Neste trabalho, desenvolvemos modelos de aprendizado de máquina para a construção de três análises inteligentes: detecção da falha de ignição, classificação do perfil de condução do motorista e predição do consumo de combustível. Para isso, utilizamos um hardware embarcado no veículo que se comunica com a ECU, aquisitando as informações solicitadas e enviando elas para um servidor na nuvem, enquanto o carro está em operação. Os modelos foram treinados com variáveis fortemente relacionadas com a análise em questão, sendo feito um processo extenso de seleção de atributos para a sua escolha. As análises são feitas em tempo real e qualquer interessado pode fazer o acompanhamento por meio de aplicações locais ou dashboards web. Comparado com os trabalhos relacionados, os resultados obtidos são similares ou melhor, com a diferença de serem obtidos por meio de algoritmos menos custosos computacionalmente, com dados exclusivos da ECU e em tempo real.

Palavras-chave: Aprendizado de Máquina; detecção de falha de ignição; análise do perfil de condução; previsão de consumo de combustível; unidade de controle eletrônico (ECU); aquisição de dados automotivos; monitoramento do comportamento do motor; análise em tempo real.

RESUMO EXPANDIDO

Introdução

Para garantir a eficiência e o desempenho de um veículo em operação é crucial monitorar o motor de combustão para evitar problemas que possam causar danos, diminuir a vida útil e aumentar as emissões de gases e o consumo de combustível. Lendo dados da Unidade de Controle Eletrônico (ECU) do veículo, é possível monitorar diferentes variáveis do motor em tempo real durante as fases de desenvolvimento do carro, garantindo e melhorando sua qualidade. A ECU do motor centraliza a leitura de vários sensores que adquirem dados sobre a operação do motor, incluindo possíveis falhas. Nesse sentido, novas estratégias baseadas em dados estão sendo aplicadas no estudo do motor e de suas peças, tanto para melhorar a interatividade com o motorista, fornecendo feedback e informações relevantes do carro, quanto para impulsionar processos industriais como calibração, controle de emissões de gases e redução do consumo. Algoritmos de aprendizado de máquina (ML) ganharam notoriedade em numerosas pesquisas e estudos devido à sua capacidade de detectar anomalias, falhas ou mudanças no padrão de comportamento do motor ou de um conjunto de variáveis. Entre os fenômenos que podem ser analisados a partir dos dados da ECU usando ML, podemos citar: (1) detecção de falhas de ignição, (2) classificação do comportamento de condução e (3) previsão de consumo de combustível.

A falha de ignição acontece quando a combustão não ocorre corretamente devido a erros na ignição ou no sistema de combustível e é uma das mais comuns em motores de combustão, podendo ser causada por fatores mecânicos, elétricos e ambientais. Além de interferir na operação do motor, reduz a economia de combustível e aumenta a poluição devido à maior concentração de toxinas causadas pela má combustão; esses problemas tornam necessária a redução da sua ocorrência. No entanto, sua detecção é difícil devido ao grande número de variáveis envolvidas no processo. Assim, algoritmos de ML podem contribuir identificando a relação entre as variáveis envolvidas no problema e construindo um modelo para detecção e diagnóstico de falhas, usando variáveis críticas e aplicando as melhores técnicas para desenvolver um algoritmo capaz de integrar conhecimento e atender às necessidades de detecção de maneira sustentável e prática.

O monitoramento e análise do perfil do motorista pode definir padrões para revelar condutas inadequadas de condução que possam causar acidentes de trânsito ou sinais de falha grave do motor, tornando possíveis ações preventivas para evitar perigos mais significativos. Além disso, oferece a oportunidade de monitorar os principais parâmetros e variáveis que afetam o controle de emissões de gases, vazamento de substâncias e consumo excessivo de veículos.

A previsão de consumo de combustível apresenta benefícios potenciais, tanto para fins econômicos quanto ambientais, principalmente relacionados ao transporte. Entre eles, a minimização do consumo em si e a redução das emissões de gases de compostos químicos tóxicos para o meio ambiente. A previsão de consumo também ajuda a calibrar o motor e seus sistemas, um processo complexo que requer muitos testes e protótipos físicos dos veículos. Algoritmos de ML podem fornecer uma previsão mais precisa do consumo de combustível do mundo real com base em uma gama mais ampla de variáveis do que os modelos tradicionais de regressão linear e reduzir a necessidade de testes físicos sem novas calibrações.

Este trabalho aborda três aplicações de ML em dados da ECU automotiva para análises inteligentes do motor de combustão e do motorista. A primeira com objetivo de detectar falhas de ignição usando variáveis comuns a veículos e sem o uso de

sensores caros, como o sensor de vibração, para o qual foram usados seis modelos de ML. A segunda aspira a classificar os perfis de direção dos motoristas, econômicos ou não, usando três algoritmos (não supervisionados e supervisionados). A terceira busca prever o consumo de combustível, aplicando três modelos de regressores. As principais diferenças em nosso trabalho em comparação com trabalhos relacionados é: (1) adquirimos dados em tempo real diretamente da ECU do carro e os enviamos para um servidor na nuvem e (2) aplicamos modelos de ML no servidor à medida que os dados chegam em tempo de execução, fornecendo feedback instantâneo para engenheiros ou usuários.

Objetivos

O principal objetivo deste trabalho é monitorar o motor de combustão para detectar falhas de ignição, analisar o perfil de condução e prever o consumo de combustível, usando exclusivamente variáveis extraídas diretamente da ECU, aplicando técnicas de ML para otimizar o desempenho dos sistemas atuais, gerar informações relevantes para engenheiros e fornecer feedback aos motoristas

Metodologia

Para coletar dados reais para as análises propostas neste trabalho, utilizamos um carro fornecido pela Renault do Brasil, um Sandero 2019, modelo 1.0, com um motor a gasolina de quatro cilindros e quatro tempos com ignição por centelha. Um hardware de aquisição desenvolvido no projeto Intelligent Acquisition and Analysis System for ECUs (IASE) foi conectado à ECU do motor. Assim, com o carro disponível, foi possível coletar uma quantidade significativa de dados reais, que foi usada para treinar e validar os modelos de ML. Não utilizamos dados de simulação ou conjuntos de dados públicos, um dos pressupostos que diferencia este trabalho de trabalhos relacionados. O sistema de aquisição desenvolvido no IASE nos permite selecionar e organizar as variáveis alvo da ECU por meio de uma interface gráfica do usuário, criando um arquivo de configuração que chamamos de "ensaio", que é carregado no hardware desenvolvido. Este hardware, responsável por coletar os dados desejados diretamente da ECU, foi incorporado ao veículo por meio do Protocolo de Calibração da Rede de Área de Controle (CAN). Sobre o barramento CAN, as ECUs normalmente suportam o Protocolo de Calibração CAN (CCP) ou o Protocolo Universal de Medição e Calibração (XCP), dependendo do modelo e fabricante da ECU, para realizar a aquisição de dados. Assim, o ensaio para ler as variáveis de uma ECU deve respeitar os limites de largura de banda impostos pelos protocolos. Por exemplo, ao usar o protocolo CCP em nosso carro, a ECU do motor se comunica com a placa a taxas de amostragem limitadas a 4, 5, 10 e 100ms. Cada ensaio de aquisição também pode ler cerca de 160 variáveis usando as taxas de amostragem disponíveis. Este dispositivo se comunica via 4G com o servidor do LISHA na nuvem, para que os dados sejam processados e/ou aplicados diretamente a modelos de ML ou qualquer outro tipo de tratamento e manipulação, em tempo real, com maior capacidade de processamento disponível, e armazenados em um banco de dados para uso posterior, incluindo em novos estudos e projetos. Em nosso servidor, os algoritmos de ML são chamados de “workflows” e são implementados em Python 3.8. Quando o dispositivo de hardware é configurado para ler uma variável da ECU do motor, as variáveis são marcadas com workflows que devem ser aplicados. Quando o servidor recebe os dados adquiridos, ele identifica quais algoritmos devem ser aplicados para esses dados específicos. Além

disso, os dados também são salvos no banco de dados para análises e/ou visualização posterior em um painel.
Ao longo de um período de dois anos, mais de cinquenta experimentos foram conduzidos para construir os conjuntos de dados. Cada tipo de análise apresentado neste trabalho tinha suas particularidades que exigiam um foco diferente, mas, em geral, um dos principais objetivos de criação dos conjuntos de dados era obter um grande número de variáveis diferentes. Isso permitiu que um processo de seleção de características (FS, na sigla em inglês) identificasse as variáveis mais relevantes para a análise em questão (detecção de falhas de ignição ou análise de consumo de combustível). Ao incorporar uma ampla gama de variáveis, a construção do conjunto de dados facilitou a identificação de relacionamentos-chave para essas análises, possibilitando o treinamento e teste de vários modelos de ML. Para isso, foi seguida uma metodologia de aquisição de dados usada em ambas as análises. Para iniciar o processo de aquisição de dados, o hardware incorporado ao veículo deve ser carregado com o arquivo de ensaio. Esse arquivo indica a localização do sinal na ECU, incluindo a definição de Smartdata (Unidade, Dispositivo, X, Y, Z, Assinatura, Tipo, Período, Fluxo de Trabalho) e os valores mínimo e máximo de limite dos sinais (predefinidos para verificar se a medição está conforme o esperado). Os dados do ensaio são enviados para o servidor. A aquisição de dados usada neste trabalho foi acionada pelo tempo. Durante essa fase, os dados são armazenados em buffer e preparados para upload serial via 4G para a plataforma IoT usando o formato SmartData. Com o dispositivo incorporado ao veículo, inúmeros ensaios podem ser carregados nele. Isso permite a realização de uma ampla gama de testes com diferentes objetivos. Cada um desses ensaios pode levar em consideração abordagens diferentes para a análise, o que é excelente para comparar e avaliar os algoritmos desenvolvidos. Diferentemente da maioria dos trabalhos relacionados, que realizam simulações de falhas em bancos de teste, as medições no projeto ocorreram no carro em operação, dirigido pelos membros do projeto, na pista de teste limitada dentro do Parque Industrial Ágora em Joinville, Brasil, onde está localizada a Universidade Federal de Santa Catarina. Considerando o número limitado de variáveis por ensaio, FS é importante para reduzir o conjunto de variáveis que representa o fenômeno em análise. Além disso, ela evita o overfitting dos modelos de ML, remove variáveis prejudiciais, redundantes ou ruidosas, gera uma melhor generalização, reduz o custo computacional e melhora o desempenho. Foram escolhidas estratégias para criar subconjuntos dos atributos disponíveis, avaliando variáveis individuais de acordo com critérios predefinidos ou selecionando-as diretamente. Em seguida, foi feita uma avaliação estatística para verificar quais delas eram mais qualificadas; esse processo geralmente tem uma condição de parada, e o melhor resultado é o que prevalece até uma avaliação adicional. Como resultado, entre todos os subconjuntos ou variáveis testadas, algumas superaram o conjunto completo e foram usadas em seu lugar para treinamento e teste dos algoritmos relacionados a cada um dos objetivos de análise do trabalho.

Resultados e Discussão

Para avaliar os modelos de detecção de falha, todos os dados dos experimentos foram reunidos em um único conjunto e balanceados. Isso ocorre devido ao fato de o número de amostras com falhas de ignição ser muito menor do que o número de amostras saudáveis, uma vez que a ocorrência de uma falha de ignição não pode ser facilmente controlada e ocorre esporadicamente. Posteriormente, dividimos os dados em conjuntos de treinamento e teste, embaralhando-os e dividindo-os em 70% para treinamento e 30% para teste, e em seguida, aplicamos os modelos de

ML. Essa técnica nos permite verificar a generalização dos modelos e comparar os diferentes resultados dos modelos treinados. Para verificar o desempenho geral dos modelos, calculamos sua precisão, recall e pontuação F1 no conjunto de teste. Em geral, o Classificador XGBoost mostrou ser o melhor modelo para identificar falhas de ignição, não apenas tendo o melhor resultado geral, mas também com uma média melhor em diferentes conjuntos de recursos obtidos por meio dos métodos de FS. O algoritmo alcançou uma precisão de até 92,40%, um recall de 96,16% e uma pontuação F1 de 94,24%. Em média, ele alcança uma precisão de 87,55%, recall de 92,25% e uma pontuação F1 de 89,79%, com um desvio padrão de 7,38%, 4,30% e 5,90%, respectivamente. Portanto, o XGBoost é o algoritmo que, em média, obteve a maior pontuação F1, que é a métrica escolhida para classificar os algoritmos.

Para avaliar os classificadores de perfil de condução, realizamos uma validação cruzada, dividindo os conjuntos de dados em conjuntos de treinamento e teste. O método de divisão escolhido considerou um experimento de uma "fase" como conjunto de teste e os demais como conjunto de treinamento. Essa técnica nos permite verificar a generalização dos modelos, além de aplicar as métricas de precisão e recall. Dessa forma, obtivemos o desempenho geral dos modelos e quantas classificações corretas foram feitas, visualizando o número de falsos positivos e negativos. Para a avaliação, dividimos os experimentos em fases, nas quais aplicamos mudanças nas variáveis selecionadas devido ao conhecimento adquirido ao longo dos experimentos (ou seja, conhecimento específico do domínio) e ao processo de FS feitos. Ao comparar as fases dos experimentos, a Fase 1 mostra desempenhos mais baixos, variando de 80% a 95% de precisão, enquanto as segunda e terceira fases apresentam resultados melhores, variando de 81% a 99% de precisão. Além disso, a precisão dos experimentos mais recentes é significativamente maior, chegando a 100% em alguns casos, embora um recall menor seja obtido. Isso mostra uma melhoria no conjunto de variáveis atualizado para classificar os níveis de consumo, mas ainda com bons resultados com o conjunto mais antigo. No geral, os resultados obtidos mostraram que os métodos de classificação tiveram boa precisão na determinação do nível de consumo a partir das variáveis corretamente selecionadas, atingindo até 100% de pontuação para todas as métricas em algumas divisões de treinamento e teste. Em geral, o XGBoost mostra uma pontuação ligeiramente melhor do que a Regressão Logística. O XGBoost teve desempenhos mais persistentes, variando entre médias elevadas e baixos desvios padrão, mostrando médias de pontuação maiores e variações menores, demonstrando persistência nelas.

Para avaliar os modelos de previsão de consumo de combustível, as métricas de erro médio quadrático (MSE), erro médio absoluto (MAE) e coeficiente de determinação (R^2) foram utilizados, para permitir uma melhor comparação com trabalhos relacionados e porque eles se complementam nas análises. Para avaliar os algoritmos de regressão, realizamos uma análise semelhante com o mesmo procedimento de validação cruzada explicado na classificação de perfil. Para as duas primeiras fases, separamos em cada rodada um dos experimentos para testes e o restante para treinamento. Para a terceira fase, realizamos validação cruzada com 3 conjuntos de dados, onde os dados foram divididos aleatoriamente em 3 conjuntos. Realizamos a validação cruzada entre eles, selecionando um conjunto para teste e o restante para treinamento. O modelo do XGBoost obteve os resultados mais consistentes e bons em geral para todas as fases e conjuntos de recursos. Ele manteve resultados semelhantes nas Fases 1 e 2, apresentando médias de MAE e R^2 acima de 0,9 e abaixo de 0,9, respectivamente. Além disso, na Fase 3, ele mostrou um desempenho geral melhor em comparação com todos os modelos (R^2 acima de 0,96, média de MSE e MAE respectivamente abaixo de 0,5

$(l/h)^2$ e 0,4 (l/h), com resultados consistentes mostrados pela pequena variação nas pontuações). Isso indica a capacidade do modelo de obter boas previsões na maioria dos casos.

Comparado a trabalhos relacionados, nos destacamos devido à abordagem inovadora de realizar análises em tempo real usando dados diretamente da ECU. Essa metodologia pode ser aplicada a outros veículos com as mesmas variáveis. Além disso, mesmo obtendo resultados melhores, utilizamos algoritmos menos complexos e mais fáceis de compreender, que são executados na nuvem. Melhoramos os resultados globais, tendo métricas com valores superiores em todos os objetos de análise, com um extenso trabalho de FS que identifica com precisão a relação da falha de ignição, do motorista e do consumo de combustível com as principais variáveis do motor, e garantindo a integridade dos dados usados.

Considerações Finais

Este estudo alcançou com sucesso seu objetivo principal de monitorar o motor a combustão para detectar falhas, analisar perfis de condução e prever o consumo de combustível por meio de técnicas de ML. Foram implementados seis modelos de ML para detectar falhas de ignição, estabelecendo a eficácia desses modelos em classificar com precisão essa falha específica, com precisão superior a 90%. Além disso, foram validados dois cenários relacionados ao consumo de combustível, comparando três modelos de ML para classificar o perfil de condução em relação ao consumo de combustível e, comparando três modelos de regressão de ML para prever o consumo de combustível, comprovando sua validade em ambos os assuntos com resultados que indicam uma categorização correta do motorista em econômico ou não econômico e uma previsão precisa do consumo. Nos classificadores, obtivemos valores de até 100% de accuracy, precision e recall, em modelos como o XGBoost e Regressão Logística. Representando uma melhoria em relação a outros trabalhos, nos quais foram obtidas pontuações menores, apesar do uso de modelos mais complexos e caros ou de dados não coletados pelos autores. Além disso, em relação aos regressores, nossos modelos foram capazes de apresentar pontuações R^2 de até 0,99, um MAE de 0,23 (l/h), além de uma média de MSE de 0,49 $(l/h)^{2,}$ chegando a 0,28 $(l/h)^2$ em alguns testes. Comparados a trabalhos semelhantes, esses resultados também representam avanços na obtenção de boas pontuações com modelos semelhantes ou até mais baratos, além do uso de dados reais coletados para treinamento e testes. A avaliação dos modelos, envolvendo tanto as fases de treinamento quanto de teste, utilizou dados autênticos coletados da ECU de um veículo, e a aplicação dos workflows ocorreu em um servidor em nuvem. Com processamento de dados em tempo real, durante a operação do veículo, as partes interessadas podem monitorar os resultados por meio de aplicativos locais ou web. Portanto, esta pesquisa tem aplicabilidade prática em diversos setores automotivos, incluindo indústria, engenharia e transporte em geral. Dado a crescente ênfase na aquisição de dados e na multidisciplinaridade envolvida no setor, espera-se que as ECUs e os componentes adicionais dos veículos gerem ainda mais dados, criando assim mais oportunidades para análises baseadas em ML.

Palavras-chave: Aprendizado de Máquina; detecção de falha de ignição; análise do perfil de condução; previsão de consumo de combustível; unidade de controle eletrônico (ECU); aquisição de dados automotivos; monitoramento do comportamento do motor; análise em tempo real.

ABSTRACT

The automotive ECU data readout can generate a significant amount of data. When properly monitored and processed, this information can help understand patterns and problems in the various engine subsystems, highlighting areas for potential improvement. Additionally, tracking this data can benefit multiple sectors within the field, including safety, tracking, reducing prototype testing, and subsequently lowering costs. It can also enhance driver interaction and serve as a valuable tool for engineers in developing new technologies. In this work, we develop machine learning models for three intelligent analyses: misfire detection, driver driving profile classification, and fuel consumption prediction. To accomplish this, we use hardware embedded in the vehicle that communicates with the ECU, acquiring the requested information and transmitting it to a cloud server while the car operates. The models were trained using variables strongly related to each respective analysis, and an extensive feature selection process was conducted to determine their relevance. Analyzes are performed in real-time, and interested parties can monitor them through local applications or web dashboards. Compared with related works, the obtained results are either similar or superior, differentiating by being obtained through computationally less costly algorithms, utilizing data exclusively from the ECU, and providing real-time analysis.

Keywords: Machine Learning. Misfire Detection. Driving Profile Analysis. Fuel Consumption Prediction. Electronic Control Unit (ECU). Automotive data acquisition. Engine Behavior Monitoring. Real-time Analysis

LIST OF FIGURES

LIST OF TABLES

LISTA DE ABREVIATURAS E SIGLAS

CONTENTS

1 INTRODUCTION

To guarantee the efficiency and performance of a vehicle in operation, as demanded by customers, market, and homologation, it is crucial to monitor the combustion engine to avoid problems that can cause damage, decrease the lifetime, and increase its gas emissions and fuel consumption. By reading data from the vehicle's Electronic Control Unit (ECU), it is possible to monitor different engine variables at runtime during the car development phases, thus ensuring and improving its quality [Bedretchuk et al. 2023]. The engine ECU centralizes the reading of several sensors that acquire data about the engine's operation, including possible failures. This diagnostic unit has a solid engineering foundation, crucial practical value, and deep theories.

In the current automotive sector, most of the market value of an auto is related to the technology used by the industry, and 60% of its executives believe that companies are well prepared for Industry 4.0 technologies, such as artificial intelligence [KPMG 2021]. In this sense, new data-based strategies are being applied in the study of the motor and its parts, both to improve interactivity with the driver, providing feedback and bringing relevant information from the car, as well as to boost industrial processes such as calibration, gas emission control, and consumption reduction [Claßen et al. 2021].

Machine learning (ML) algorithms gained notoriety in numerous research and studies due to their ability to detect anomalies, faults, or changes in the behavior pattern of the engine or a set of variables [Ly, Martin e Trivedi 2013, Toledo e Shiftan 2016, Yang et al. 2022]. Among the phenomena that can be analyzed from the ECU data using ML, we can cite: (1) misfire detection [Wu e Liu 2009], (2) classification of driving behavior [Shahverdy et al. 2020], and (3) fuel consumption prediction [Katreddi e Thiruvengadam 2021].

1. An engine can suffer from several problems and failures, such as a misfire, which happens when combustion does not occur correctly due to ignition or fuel system errors. When a misfire occurs, the engine's balance is destroyed, and its center of gravity is shifted [Hmida et al. 2021]. This failure is one of the most common in combustion engines and can be caused by mechanical, electrical, and environmental factors. Besides interfering with engine operation, it reduces fuel economy and increases pollution because of the higher concentration of toxins caused by poor fuel combustion; these problems make reducing the occurrence of this fault necessary. However, its detection is difficult due to the number of variables involved in the process [Sharma, Sugumaran e Devasenapati 2014]. Even the detection made by the onboard computer is still tricky, given the complexity of the fault. In general, a misfire can not be analyzed from a single point of view, as explained further (in Section 2.3.1), either from a single variable or just one engine component. Several sensors and different techniques are required, often making

fault detection more expensive or consuming much computational power, which is limited in the ECU. Thereby, it is not feasible to build a simple physical model to represent this phenomenon, given all the factors that can cause the failure.

The detection of engine misfire is a requirement of the Onboard Diagnostics II (OBDII) standard [Merkisz, Bogus e Grzeszczyk 2001], present in the legislation of several countries, including Brazil. According to [Ye 2009], fault diagnoses have produced enormous benefits since they were developed, whether economic or environmental and have become an object of research in many countries. In this sense, machine learning algorithms can contribute by identifying the relationship between the variables involved in the problem and building a model for fault detection and diagnosis [Chen et al. 2018, Liu et al. 2013], using critical variables, and applying the best techniques to develop an algorithm capable of integrating knowledge and meeting the detection needs sustainably and practically.

Previous studies of misfire detection have used various methods precisely because of their different causes. Mechanically, [Kiencke 1999] analyzed signals related to the angular velocity of the motor, and [Tinaut et al. 2007] looked at the difference in kinetic and potential energy in samples with failure conditions. On the other hand, observing vibration signals, [Jafarian et al. 2018] performed frequency analysis, and [Rath et al. 2019] used harmonic vibrations to extract relevant information from the engine under fault conditions. [Wu e Liu 2009] applied artificial intelligence techniques in multi-resolution analysis, and [Qin et al. 2021] developed a method for automatically extracting time-domain, time-frequency-domain information, and hand-craft time-domain statistical features to classify read data into fault or regular condition classes.

2. Driver profile monitoring can identify vehicle thefts, which is interesting for insurers and transport companies regarding security and personalized insurance plans [Händel et al. 2014, Huang e Meng 2019]. Selling data to insurance companies is a growing market. 43% of automakers will sell driver and vehicle data to insurance companies in a short time [KPMG 2021]. Furthermore, profile analysis can define patterns to reveal improper driving conduct that may cause traffic accidents or signs of severe engine failure, making preventive actions possible to avoid more significant dangers [Hamed, Khafagy e Badry 2021]. In addition, it provides the opportunity to monitor the main parameters and variables that affect the control of gas emissions, substance leakage, and excessive vehicle consumption [Kalogirou 2003, Rahnama, Arab e Reitz 2020, Wang et al. 2021].

   Related works have been proposed to identify and classify driving conduct by labeling driving habits as safe or unsafe [Lattanzi e Freschi 2021], applying cluster algorithms to classify the driver as eco-friendly or not [Ly, Martin e Trivedi 2013,

Peppes et al. 2021], and verifying the impact of monitoring on fuel consumption [Liimatainen 2011, Toledo e Shiftan 2016].

3. Fuel consumption prediction has potential benefits, whether for economic or environmental purposes, mainly in works related to transport [Katreddi e Thiruvengadam 2021], which has been using ML to develop predictive models widely in recent years [Hamed, Khafagy e Badry 2021, Yao et al. 2020, Kanarachos, Mathew e Fitzpatrick 2019]. AI-based model's benefits include the minimization of fuel consumption regarding the drop of gas emissions of toxic chemical compounds to the environment. Moreover, the fuel consumption prediction can help calibrate the engine and its systems, a complex process that requires many tests and physical prototypes of the vehicles [Claßen et al. 2021]. This is an expensive and time-consuming step in building/testing new engines, making it difficult to comply with legislation. ML has several strengths over traditional methods, including its ability to handle large amounts of data and complex relationships between variables [Ziółkowski et al. 2021]. ML algorithms can provide a more accurate prediction of real-world fuel consumption based on a broader range of variables than traditional linear regression models [Pavlovic et al. 2020] and reduce the need for physical testing without new calibrations [Ma, Shahbakhti e Chigan 2023].

   For fuel consumption regression, studies proposed models based on driving behavior data collected by smartphones and OBD terminals [Yao et al. 2020], applied Support Vector Machine [Hamed, Khafagy e Badry 2021], and developed Deep Neural Network models [Kanarachos, Mathew e Fitzpatrick 2019] to predict fuel consumption. In general, the works obtained good results. However, with issues that can be improved. Whether in the sets of variables used, which may not be present in all vehicles [Lattanzi e Freschi 2021], not associating the models with factors of intrinsic fuel consumption, such as acceleration [Ly, Martin e Trivedi 2013], in the training data, which were not collected by the authors [Lattanzi e Freschi 2021, Yang et al. 2022], using equipment with lower sampling frequencies [Yao et al. 2020], such as OBD terminals, developing algorithms that require more computational power [Kanarachos, Mathew e Fitzpatrick 2019], or in the execution time, that not occur at runtime [Liimatainen 2011].

The project on which this thesis was developed resulted from a partnership between Renault and the Federal Government's Rota 2030 program. Called Intelligent Acquisition and Analysis System for ECUs (IASE), the project was a joint effort of the Software/Hardware Integration Lab (LISHA) and Renault, and its main goal is to explore the potential of applying AI within the context of the Internet of Things (IoT) to optimize the operation of Combustion Engines, mainly in matters related to the calibration of controller parameters and anomaly detection. The project team developed an embedded

system that was added to a vehicle provided by Renault do Brasil, and experiments were carried out on it to capture data in natural and non-simulated conditions.

In this work, data was captured from the ECU through experiments generated over two years, acquiring a considerable amount of data and enabling this work and also future works since the data was saved in the cloud. Furthermore, feature selection (FS) techniques were applied to define the main variables in each analysis addressed in this document. Having the data available, several machine learning algorithms were developed (Gradient Boost, Logistic Regression, Support Vector Regression, Multilayer Perceptron, Autoencoder, among others) based on the related works and also on the understanding of the engine to analyze these data and generate information, and applications relevant to the project and to Renault. In this way, the data of interest is taken directly from the ECU and applied to the ML algorithms. The results were compared and explored to understand which models best adapt to the proposed problems and how they contrast with the literature.

This work deals with three approaches to applying ML algorithms in automotive ECU data for intelligent analyses of the combustion engine and the driver. The first aim was to detect misfires using variables common to vehicles and without using expensive sensors such as the vibration sensor, for which six ML models were used. The second aspires to classify drivers' driving profiles, whether economical or not, using three ML algorithms (unsupervised and two supervised). The third seeks to predict fuel consumption, applying three ML models of regressors. The main difference in our work compared to related work is twofold: (1) we acquire real-time data directly from the car's ECU and send it to a cloud server, and (2) we apply ML models on the server as data arrives at runtime, providing instant feedback to engineers or users.

1.1 GOALS

The main objective of this work is to monitor the combustion engine to detect misfires, analyze the driving profile and predict fuel consumption, using exclusively variables taken directly from the ECU, applying different machine learning techniques to optimize the performance of current systems, generate relevant information for engineers and provide feedback to drivers. The following specific objectives have been defined to assist in the development of the main goal:

1. Study the operation of internal combustion engines;
2. Understand the causes and symptoms of the misfire and how to identify it through the data collected from the ECU;
3. Identify variables strongly related to fuel consumption, both for driving profile analysis and for consumption regression;

4. Collect and build datasets with real data applying data engineering to validate and optimize them;

5. Implement machine learning algorithms for misfire detection, profile classification, and fuel consumption regression, evaluating and comparing their performance with related work;

6. Develop a complete flow for real-time fault detection, driving profile classification, and fuel consumption prediction in running cars.

1.2 CONTRIBUTIONS

In summary, we make the following contributions:

1. We developed and compared six ML-based models (Xgboost, Gradient Boosting, K-Means, K-Neighbors, Logistic Regression, and Support Vector Classification), representing different classes of algorithms, to detect misfires at runtime. [Shahid, Ko e Kwon 2022, Qin et al. 2021] also used ML algorithms and obtained similar results. However, the authors implemented computationally more expensive algorithms, such as neural networks with multi-domain input and convolutional neural networks [Dreiseitl e Ohno-Machado 2002];

2. We implemented and compared three ML-based algorithms (K-Means, Logistic Regression, and XGBoost), representing different classifiers, to categorize driving behavior as fuel-friendly or not. Likewise, [Peppes et al. 2021] compared algorithms and obtained similar results. However, implemented computationally more expensive algorithms, such as neural networks [Dreiseitl e Ohno-Machado 2002];

3. We developed and compared three algorithms (Ridge Regression, Support Vector Regression, and XGBoost) for fuel consumption prediction. [Yao et al. 2020, Kanarachos, Mathew e Fitzpatrick 2019] also worked on predicting consumption. However, they used neural networks and smartphones, requiring more resources and greater computational power [Dreiseitl e Ohno-Machado 2002];

4. We trained all models with real data captured from a test vehicle and compared the performance of the algorithms with changes in driving and environment. Our results indicate that we can detect misfires with a precision of 92.4%, recall of 96.16 %, and F1 Score of 94.24%. Furthermore, we can classify the driving behavior with up to 100% accuracy, recall, and precision, besides predicting consumption with an average of the square of errors between predicted and measured values of 0.28 (l/h^2), mean absolute error of 0.23 (l/h) and coefficient of determination of 0.99;

5. We conducted a comprehensive literature review and compared existing works in terms of resources used, techniques applied, and results obtained, summarizing this in three tables of related works (3.1.1 Summary of misfire detection techniques, 3.2.1 Summary of related works that use classification techniques, and 3.3 Summary of related works that use regression techniques).

1.3 STATEMENT OF CONTRIBUTION

The content of this thesis is based on the work presented in the following articles:

[Canal, Riffel e Gracioli 2023]: Rafael Canal, Felipe Kaminsky Riffel, and Giovani Gracioli. "Driving Profile Analysis Using Machine Learning Techniques and ECU Data". 32nd International Symposium on Industrial Electronics (ISIE), 2023, IEEE.

[Bedretchuk et al. 2023]: João Paulo Bedretchuk, Sergio Arribas García, Thiago Nogiri Igarashi, Rafael Canal, Anderson Wedderhoff Spengler, and Giovani Gracioli. "Low-Cost Data Acquisition System for Automotive Electronic Control Units". v. 23, n. 4, 2023.

[Canal et al. 2023]: Rafael Canal, Felipe Kaminsky Riffel, João Paulo Araujo Bonomo, Rodrigo Santos de Carvalho, and Giovani Gracioli. "Misfire Detection in Combustion Engines Using Machine Learning Techniques". XIII Brazilian Symposium on Computing Systems Engineering (SBESC), 2023, IEEE.

[Canal, Riffel e Gracioli 2023]: Rafael Canal, Felipe Kaminsky Riffel, and Giovani Gracioli. "Machine learning for real-time fuel consumption prediction and driving profile classification based on ECU data". Submitted for publication.

Due to the relationship to published work, this thesis contains significant material from [Canal, Riffel e Gracioli 2023, Canal, Riffel e Gracioli 2023, Canal et al. 2023, Bedretchuk et al. 2023]. We thank all co-authors for their precious help in completing this research. In particular, we would like to thank Felipe Kaminsky Riffel for his help in applying the feature selection algorithms and in refining the algorithms developed in the work to improve the results.

1.4 DOCUMENT ORGANIZATION

The document is organized as follows. Section 2 grounds the theory needed to understand the work. The 3 section reviews, summarizes, and compares related works. The section 4 describes the data acquisition methodology we used in this work, detailing each step of the process from designing experiments, algorithms and parameters used, training and testing steps, to real-time analysis. The section 5 presents the results achieved, comparing them with the literature. Finally, Section 6 concludes the work, showing its relevance for the automotive area.

2 BACKGROUND

The following subsections discuss the main theoretical concepts important for understanding and developing this work.

2.1 ELECTRONIC CONTROL UNIT

The ECU is a critical component in modern electronic systems used in various applications, including automotive, aerospace, industrial automation, and consumer electronics. The ECU serves as the central processing unit that manages and controls the operations of different subsystems within a more extensive system [Bedretchuk et al. 2023].

The electronics present in automobiles include, among other components, transmission and engine controls, safety systems, measurement, and diagnostic modules. The ECU is a microprocessor responsible for calibrating and controlling the essential functions of the engine and its components [Kruse, Kurz e Lang 2010]. It is a critical system that regulates vehicle gas emissions, improves average fuel economy, increases reliability, and reduces costs. The ECU allows monitoring of various mechanical and electrical systems in the vehicle, where each part of the vehicle communicates internally. In general, the Controller Area Network(CAN) performs the vehicle communication network due to the advances in this technology related to communication speed and data transmission [Nguyen, Cheon e Jeon 2014].

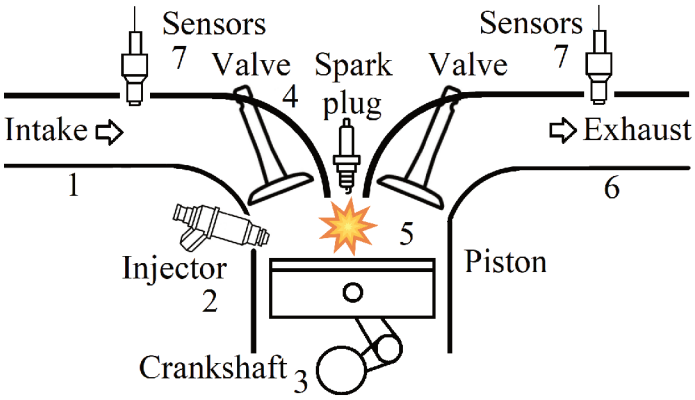
As technology advances, ECUs evolve, enabling more sophisticated functionality and addressing challenges associated with software complexity [Nag, Ghanekar e Harmalkar 2019] and system integration. The future of ECUs lies in their integration with emerging technologies, such as artificial intelligence and connectivity, to enhance performance, efficiency, and user experience in a wide range of applications [Weber et al. 2019, Hafeez et al. 2021]. Integration like the one proposed in this work, applying machine learning techniques in an industrial scenario for real-time analysis to help engineers optimize the engine's performance and its components and drivers to improve their driving style.

2.2 COMBUSTION ENGINE

Most automobiles are equipped with a 4-stroke combustion engine, so called because its operation is based on four different stages: intake, compression, combustion, and exhaust, where combustion occurs once every two revolutions [Lilly 1999]. Figure 2.1 demonstrates the combustion process in one cylinder. In the first revolution, the first stage (1) begins at the top dead center (TDC); the engine piston moves down and pulls the fuel and atmospheric air mixture through the intake valve. The fuel is injected by the

injector (2). In the second stage, beginning at the bottom dead center (BDC) and with the combustion chamber filled, the intake and exhaust valves are closed, and the piston rise, compressing this mixture. In the third stage, starting the second revolution when the piston reaches the maximum point (still in TDC), the spark plug emits an electric spark (4) that causes the explosion (5) and forces the piston back to BDC, producing mechanical work and moving the piston downwards to turn the crankshaft (3). This part is essential because the chemical energy (from combustion) is transformed into mechanical energy (which moves the car). In the fourth stage, returning from BDC to TDC, the piston rises again, and the exhaust valve opens (6), releasing the gases formed in the combustion. When this valve closes, the intake valve opens again, and the process restarts.

Figure 2.1 – Engine cylinder schematic demonstrates a misfire failure and its main parts and sensors.



Source: Author (2023).

It is substantial that the fuel withstands the compression and does not cause the knock-noise fault (does not explode before the third stage) and also that a misfire fault does not happen (does not delay or even cancel the explosion on the fourth stage), if this occurs, the engine power will be reduced. In this sense, it is necessary that the maintenance of the components involved in the process is up to date and that the gasoline used is of good quality. These two faults are common in this type of engine. In this scenario, the understanding of the misfire and its detection are the main objectives of this work. The sensors inserted into the car's engine and cylinders (7) and their respective monitoring through the ECU can help with this procedure.

2.3 MISFIRE

A misfire occurs when the burning process is not done correctly in the combustion chamber, either with the delay in combustion or the lack of it. In this case, the engine suffers from energy loss, unbalancing its operation until a new combustion cycle begins. The California Air Resources Board (CARB) [Board 1991] regulations define engine misfire as a lack of combustion in the cylinder due to the absence of spark, poor fuel metering, poor compression, or any other cause.

Misfire is one of the common failures of multi-cylinder engines [Qin et al. 2021]. It affects the engine's output power, which can reduce its power capacity by up to 25% [Sharma, Sugumaran e Devasenapati 2014], and save engine operation, leading to problems of weak acceleration, increased fuel consumption, and pollutant emissions, due to higher concentration of unspent hydrocarbons present in the exhaust gases. Consequently, and supported by the OB-DII legislation, many companies and researchers have proposed practical methods to detect and avoid misfires, as shown in section 3.

2.3.1 Causes

As the combustion process involves many other parts, be they mechanics, environment, driver experience, maintenance, and fuel quality, the source of misfires can be challenging to find. The failure can also occur constantly or intermittently due to ignition system failure, unbalanced air-fuel ratio, faulty spark plug, blown head gasket, clogged catalytic converter, lack of compression, or even failure of the gas re-circulation system of escape [Boguś e Merkisz 2005]. Next, studies that analyzed the causes of misfires are described, and at the end of the subsection, a table is presented summarizing the principal causes of failure.

The **ignition system failure** occurs from failures in sensors related to pressure in the suction manifold, crankshaft position, cooling liquid temperature, sucked-in air temperature, throttle position, lambda, and other work elements like the valve of fuel vapor absorbent, fuel pump, inertial switch, and pintle injectors [Dziubinski et al. 2017].

A **foul or a bad spark plug** can be coated with a material like tar, gasoline, or carbon, and a misfire could be caused by pre-ignition from the spark plug gap fault. For optimal engine performance, the spark plugs must be clean (with the electrodes intact); otherwise, the engine's running would be changed [Azrin et al. 2021].

A **blown head gasket** can be caused by the compression in the cylinder because of the use of aluminum rather than iron cylinder heads [Komorska 2011]. Aluminum has a higher thermal expansion rate than iron, causing much more stress on the head gasket, which can cause compression leakage between the cylinders if damaged.

An **imbalance air-fuel ratio** occurs if the mixture in one or more cylinders is different from the others [Nakagawa, Fukuchi e Numata 2012]. This malfunction can

be caused by a problem with the fuel injector, an intake leak in one of the cylinders, or even a flow problem in the exhaust gas re-circulation of a cylinder.

A **clogged catalytic converter** happens when the engine temperature rises above 1000℃ [Klenk et al. 1993]. Under normal conditions (without misfire), even at high speed, the engine reaches up to about 950℃. If it reaches a higher temperature, the catalytic converter can melt and clog due to overheating.

A **lack of compression** can be caused by sloughing off piston and piston rings, damaged valve seals, defect in the hydraulic lifter, camshaft re-synchronization, or by penetration of the air/fuel mixture into the crankcase [Ting e Mayer J. E. 1974].

An **exhaust gas re-circulation** system failure occurs when gas flow is insufficient [Wei et al. 2012]. It can be caused by airflow obstruction, valve sticking, a differential pressure feedback sensor, or a vacuum-switch valve failure.

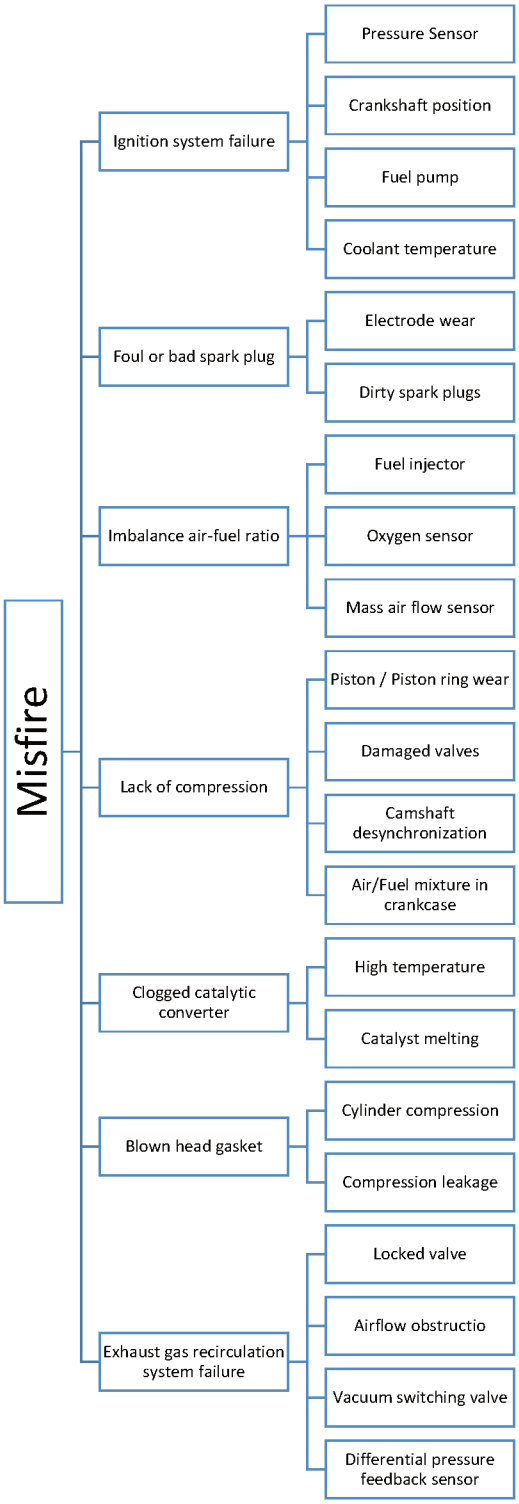
Table 2.1 provides a summary of the causes described above and, in addition, Figure 2.2 illustrates the misfire tree and the failure chain of each of these causes, according to the revised literature.

Table 2.1 – Summary of causes

Misfire Causes	**Engine conditions**
Ignition system failure	Faulty sensors and working elements that are part of the injection system [Dziubinski et al. 2017].
Faulty spark plug	Spark plug electrode wear [Azrin et al. 2021].
Blown head gasket	Compression in the cylinder due to use of incorrect material [Komorska 2011].
Imbalanced air-fuel ratio	Faulty oxygen sensor, faulty mass air flow sensor, fuel injector malfunction, or intake leak in a cylinder [Nakagawa, Fukuchi e Numata 2012].
Clogged catalytic converter	Too high temperature [Klenk et al. 1993].
Lack of compression	Piston rings and pistons have worn out, or air/fuel mixture reaches crankcase [Ting e Mayer J. E. 1974].
Exhaust gas re-circulation system failure	Glued or charred exhaust gas re-circulation valves [Wei et al. 2012].

Source: Author (2023).

Figure 2.2 – Misfire causes tree.



Source: Author (2023).

2.3.2 Symptoms

The symptoms caused by misfires are subtle and often imperceptible. Detecting the failure by perceiving the macro symptoms alone is difficult, but it is important for the general understanding of the failure that they are known. Thus, based on the aforementioned authors, the following list was made.

- **Brute acceleration**: the car moves in a jolt when stepping on the acceleration pedal; it is possible to feel a strong tremor coming from the engine. The most common situation to detect misfires is in high gear, low rotation, and the throttle on the ground. Raw throttle is a typical sign that the engine is failing;
- **Rough Idle**: a very irregular idle, which can cause the engine to shut down. Therefore, the engine sensors will have faulty values, and the fuel mixture will be confused;
- **Vibrations**: when one or more engine cylinders do not fire correctly, the engine becomes unbalanced, causing strong vibrations inside the cabin when accelerating or idling.
- **Engine light**: if a sensor has failed or detected something wrong with the engine, it will send the information to the ECU. Thus, it is up to the unit to decide whether the problem is serious. Also, if the problem occurs repeatedly, the ECU will turn on the check light. When the ECU detects misfires, it is widespread for it to turn on the engine light and store a fault code relating it to the cylinder in which it occurred;
- **Slow Acceleration**: O2 sensors can receive incorrect information and generate either very lean or very rich mixtures, which can cause a decrease in acceleration;
- **Strong smell** of not burnt fuel in the exhaust;
- **Change of engine sound**.

2.3.3 Detection

The literature includes different approaches for the detection of misfires in combustion engines, such as linear and non-linear analysis of engine vibration [Syta, Czarnigowski e Jakliński 2021], automatic diagnosis based on torsional vibration and block rotation [Chen et al. 2012] or based on simulation models [Chen e Bond Randall 2015], crank speed monitoring [Wang et al. 2022] and even real-time torque estimation [Zheng et al. 2019].

In addition to external approaches, the ECU can detect some faults. The first information the ECU provides is the anomaly light flashing on the car's dashboard at

the time of failure. To identify the defective cylinder, it is necessary to connect a reader to the diagnostics connector (Data Link Connector), which is used to access onboard diagnostics and live data streams from the ECU. Using it, it is possible to access the vehicle's ECU and capture the fault codes recorded during the event. P0301 is an example code for misfire or combustion failure. Where P stands for power-train and is related to power-train system failures, 030 indicates misfire, and 1 indicates cylinder one has or has had the fault [McCord 2011].

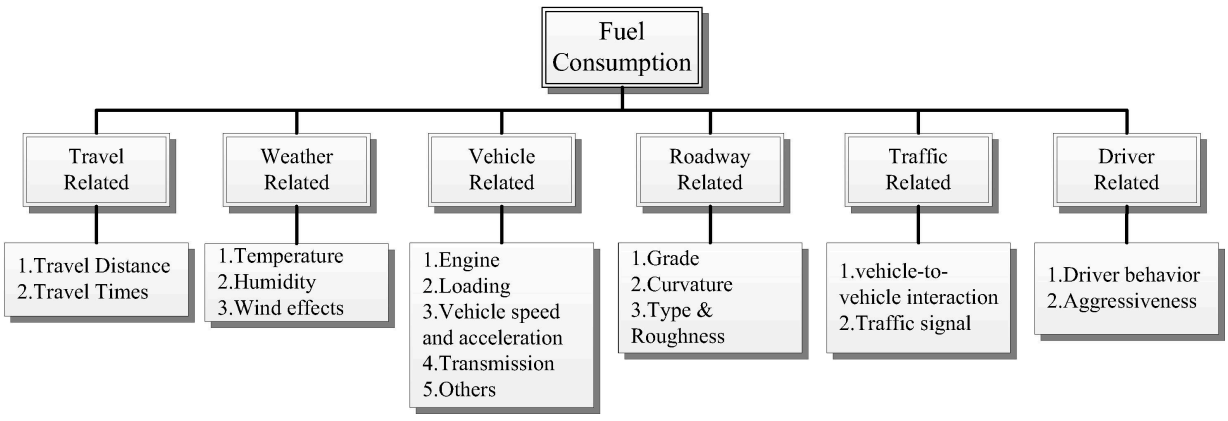
It is possible to apply techniques using the crankshaft angular velocity measured at the flywheel, brake torque, engine vibrations, combustion chamber pressure, exhaust gas pressure, and ionic current between spark plug electrodes, among others, to detect the misfire fault. However, based on what has been seen so far (2.3, 2.3.1, 2.3.2 and 2.3.3), it is possible to use information from the cylinder/engine sensors generated at run-time, to detect a misfire. Moreover, new techniques are being implemented, mainly due to their high failure prediction accuracy, such as the application of machine learning (ML) algorithms [Singh, Potala e Mohanty 2018].

2.4 FUEL CONSUMPTION

Environmental sustainability and energy security are concerns that have grown considerably. Therefore, understanding fuel consumption in the transport sector becomes increasingly important. Fuel consumption affects transport systems' economic viability and has significant environmental impacts, including greenhouse gas emissions and air pollution [Ali et al. 2022, Abokyi et al. 2019].

Figure 2.3 represents the main factors that impact fuel consumption [Zhou, Jin e Wang 2016].

Figure 2.3 – A structure diagram of the factors affecting vehicle fuel consumption.



Source: [Zhou, Jin e Wang 2016]

Travel-related factors significantly impact vehicle energy consumption and emissions. The distance traveled directly affects fuel consumption, with longer trips generally resulting in higher energy usage. Also, urban driving can be less efficient compared to highway driving. Moreover, driving at higher speeds increases aerodynamic drag and engine workload, leading to higher fuel consumption and emissions. Therefore, considering trip distances and optimizing driving routes can help minimize energy consumption and emissions.

Weather-conditions play a crucial role in vehicle energy consumption. Temperature can significantly impact fuel efficiency due to more extended engine warm-up times. Additionally, strong headwinds encountered during windy conditions can create higher aerodynamic drag, resulting in more energy required to maintain speed.

Vehicle-related factors encompass various aspects that influence fuel consumption and emissions. A vehicle's weight, size, and shape play a role in determining its aerodynamic efficiency. The type and size of the engine, quality of parts, monitoring of components, and embedded technology exhibit varying fuel consumption rates and can positively impact energy efficiency. Additionally, regular vehicle maintenance is essential.

Roadway-conditions significantly influence vehicle energy consumption. Uneven or poorly maintained roads increase rolling resistance, requiring more energy to overcome; well-maintained roads and planning routes with minimal congestion can help optimize energy efficiency.

Traffic-related factors impact both fuel consumption and emissions. Smooth traffic flow at a constant speed is more efficient than frequent stops and starts. Congested traffic, with high vehicle density, can lead to lower average speeds and increased fuel consumption.

Driver-related factors can significantly influence vehicle energy consumption and emissions. Driving behavior plays a crucial role, as aggressive driving habits such as rapid acceleration, harsh braking, and excessive speeding can significantly increase fuel consumption. Drivers can optimize fuel efficiency by practicing smooth and gradual acceleration, gentle braking, and adhering to speed limits.

As most of the factors mentioned above cannot be easily controlled in a research environment, our work focuses on two of the main ones: related to the vehicle and the driver. Thus, subsequent sections will explore fuel consumption, examining some of its determinants and consequences, mainly related to vehicle technology, engine monitoring, and driver behavior.

2.4.1 Driving and Consumption Profile

A vehicle's driving and consumption profile encompasses the driving behavior attributes and fuel consumption patterns demonstrated by a car [Lois et al. 2019].

Monitoring and finding critical aspects of the driving and consumption profile is essential to assess the vehicle's efficiency and environmental impact and identify possible paths for improvement. This section details the fundamental components of a driving and consumption profile, including driving style, driving conditions, and fuel consumption patterns.

Studies have emphasized the importance of driving style as a determinant of fuel consumption variations between vehicles [Weller et al. 2019, Yao et al. 2020, Holden, Gilpin e Banister 2019]. Aggressive driving behaviors such as rapid acceleration, abrupt deceleration, and excessive speed have been found to increase fuel consumption and emissions significantly. On the other hand, adopting a more eco-friendly driving style, characterized by smoother accelerations, gradual deceleration, and compliance with speed limits, can generate substantial fuel savings [Gao et al. 2019]. Driving conditions also have a considerable influence on a vehicle's fuel consumption. For example, stop-and-go traffic conditions can increase idling and frequent braking, increasing fuel consumption [Huang et al. 2019].

Analyzing fuel consumption patterns is vital to understanding a vehicle's efficiency and performance. This involves monitoring fuel consumption rates at specific intervals or during different driving modes (e.g., city or highway driving, dry or wet weather, high or low temperature). Allows the identification of trends, anomalies, and areas for improvement. Advanced technologies such as onboard fuel consumption gauges and more technological cloud-based systems can provide real-time data and analytics on fuel usage [Moradi e Miranda-Moreno 2020, Zheng et al. 2020], enabling drivers and fleet operators to optimize fuel efficiency.

By understanding driving and consumption profiles, researchers and policymakers can formulate targeted strategies to increase fuel efficiency, reduce emissions and promote green driving practices. This knowledge can guide the development of training programs, eco-driving campaigns, and policy interventions to promote sustainable transport systems [Li et al. 2021, Tsakalidis et al. 2020, Holden, Gilpin e Banister 2019].

2.4.2 Consumption Estimation

In automotive and engineering fields, regression models and fuel consumption forecasting are vital in optimizing the efficiency of various systems such as combustion engines, vehicles, and industrial machines. These models provide insight into fuel consumption patterns, allowing engineers to make informed decisions about design improvements, operating strategies, and energy management. This section presents an overview of fuel consumption regression and prediction models, focusing on their applications and methodologies.

Modeling fuel consumption in combustion engines is crucial to understanding and optimizing routine. Multiple regression techniques have been used to develop em-

pirical fuel consumption models [Slavin et al. 2013, Miri, Fotouhi e Ewin 2021], considering various engine parameters such as engine speed, load, air-fuel ratio, and ignition timing [Çapraz et al. 2016]. These models, often based on experimental data, allow engineers to estimate fuel consumption under different operating conditions and guide the development of efficient engine designs.

In the automotive industry, predicting fuel consumption is essential for evaluating vehicle performance, optimizing energy efficiency, and meeting regulatory standards [Wang et al. 2018]. Statistical regression models such as multiple linear regression and artificial neural networks [Xie et al. 2023] are commonly employed to predict vehicle fuel consumption based on speed, weight, aerodynamics, and driving conditions [Chen et al. 2017, Moradi e Miranda-Moreno 2020]. These models help in the design of fuel-efficient vehicles, in the development of eco-driving techniques, and in the evaluation of new technologies such as hybrid and electric engines [Zhang et al. 2020, İnci et al. 2021].

Regression models and fuel consumption prediction are also employed in mechanical engineering for energy management in industrial machines [Rahman e Smith 2017, Yao et al. 2020]. By analyzing historical factors such as fuel consumption data and considering equipment load, operating time, and maintenance schedules, engineers can develop models to optimize energy use, reduce fuel waste and improve the system's overall efficiency.

As automotive engineering advances, several trends and challenges arise in the regression and prediction of fuel consumption. Integrating real-time data acquisition systems, advanced sensors, and machine learning algorithms allows the development of more accurate and adaptive fuel consumption models. Furthermore, incorporating emerging technologies such as the Internet of Things (IoT) and big data analytics offers new opportunities for data-driven fuel consumption optimization in mechanical systems [BULUT e ILHAN 2019, Hawkins et al. 2021].

2.5 MACHINE LEARNING

In recent years (as can be seen from the dates of the related works (3), ML algorithms have been extensively explored in the field of research related to fault detection and have developed rapidly. Compared to algorithms with human-designed indicators, ML algorithms extract more failure features from a signal and can process more signals simultaneously [Wang et al. 2022].

ML use in fault detection is interesting because many techniques do not require a lot of computational resources or affordably have these resources, having great accuracy and being reliable [Sharma, Sugumaran e Devasenapati 2014]. In addition, the algorithms can be trained numerous times to follow the system changes, either manually, where external data sets are applied to the algorithm, or automatically, implementing algorithms capable of updating themselves continuously as the car's engine

conditions change.

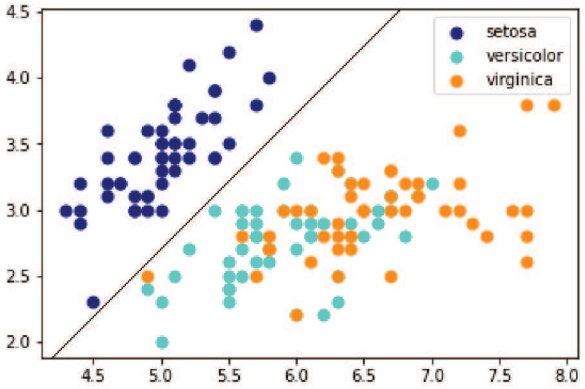
ML algorithms adapt their attributes to become more effective in performing the applied task. This adaptation is initially achieved in training, an important part of the algorithm learning process. The algorithm can respond to training in a few ways, either with variable parameters adjusted through iterative optimization, with possible classification paths, or with the probability distribution, from the input data. These algorithms must generally be oriented to learning by studying, experimenting, or being taught. Among these forms are two broad classifications: supervised and unsupervised learning, defined by Murata et al. 2002 and Zhao e Liu 2007 as described below.

- **Supervised**: the training includes the input data and the expected results so that the algorithm learns to make this association. In this type of learning, examples are passed to the program with the data and the expected answer. The main objective is for it to learn and make a good generalization of the relationship between input and output so that it can classify correctly when it receives new data never seen before. This method is commonly used in classification problems, where algorithms try to categorize data (decision trees, support vector machines (SVM), and many distance-based algorithms), and regression, in which algorithms understand the relationship between independent and dependent variables of the data (linear regression and also FS.

- **Unsupervised**: training does not include expected results. It is mainly used in clustering algorithms to group input data into classes using more statistical information. The purpose of this type of training is to make the model learn the patterns among the data by itself and is mostly used in clustering algorithms, which group data into categories according to their similarity; association algorithms, which make the association of data precisely by some defined base rule, and dimensionality reduction, where the number of dimensions of the data is reduced in terms of complexity or number of characteristics or input data

Next, the most applied ML algorithms in failure detection found in the literature are briefly explained. Also, in order not to clutter the background section with code examples, an application example of each algorithm explained below was added to Appendix A, demonstrating how programming is made easier with the help of Python libraries, the dataset used was the Iris Data Set from Dua e Graff 2017. The dataset consists of 4 attributes plus 3 classes of 50 samples each, which refers to a type of iris plant (setosa, versicolor, or virginica). Among them, one class is linearly separable, and the others are not separable from each other (Figure 2.4).

Figure 2.4 – Iris dataset classes.



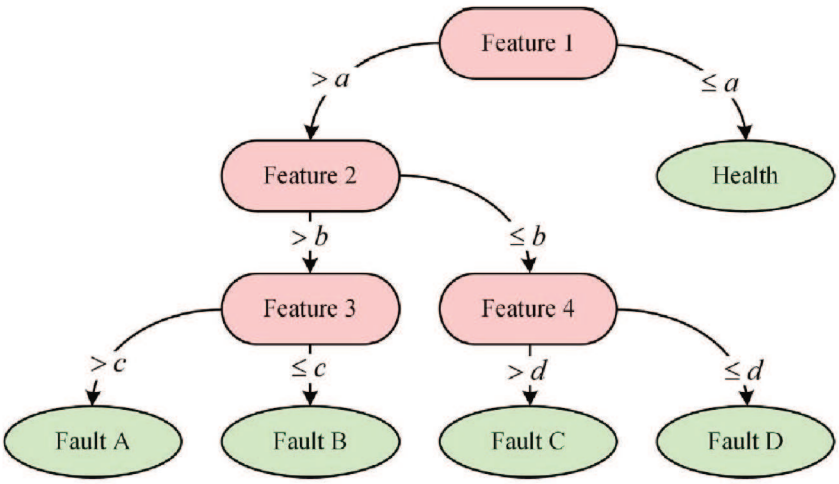
Source: Author (2023).

2.5.1 Clustering and Classifiers Algorithm

The theory and algorithms used for clustering and classification will be presented in this topic.

A decision tree is a graphical model used in machine learning to predict outcomes based on data features. It starts with a root node and branches into different nodes, representing possible decisions or outcomes (Figure 2.5). Each node is determined by an attribute or feature, leading to further branches until reaching terminal nodes, which correspond to outcomes. Decision trees are useful for classification and regression tasks, providing interpretable and efficient models for data analysis.

Figure 2.5 – Illustration of the decision tree.



Source: Lei et al. 2020.

Decision tree algorithms are a supervised learning method widely used in clas-

sification, establishing the relationship between the class and the attributes. In machine learning, the root is a database attribute, and the leaf node is the class or value that will be generated in response. They are used to extract statistical features (bias, variance, mean, etc.), selecting those that most contribute to identifying fault classes. They can be used to predict discrete categories (yes or no, for example) and to predict numerical values (estimated expenses for the month, for example) [Russell 2010]. They fit into a group of well-developed machine learning algorithms that efficiently implement the obtained ruleset as fuzzy logic in many systems [Lei et al. 2020]. This algorithm is simpler to understand than other machine learning techniques because it is possible to know what happens in each part of the process, unlike the black boxes formed in neural networks.

If-then rules exist in the links between nodes to understand how this algorithm works. For example, when data arrives at node X, the algorithm applies a defined rule, such as: if the characteristic of the analyzed data is greater or equal to Y. If it is, then it goes to one side of the tree. Otherwise, it goes to another. This logic is applied to the following nodes until it reaches the leaf node and has a final result. The tree's primary function is to find the nodes that will be fitted in each position, who will be the root node, then who will be the left node, the right node, and so on.

Although it is significant to know how a decision tree works and how to build one, it is interesting to use the libraries provided by the language in which the code is being written to optimize time and resources.

K-Means is an unsupervised clustering algorithm that divides the data into classes or clusters, with a number K of clusters informed as a parameter. In this method, the clusters are determined by centroids, K points in the data space, where a point belongs to a cluster if the respective centroid is the closest from it [Aggarwal et al. 2015].

K-means determines its clusters through an iterative optimization process, starting with the K centroids initially randomized. In each step, the centroids are updated to the mean array in each cluster. Formally defining, given clusters $S_1, \ldots, S_K$, and points $x^{(1)}, \ldots, x^{(n)}$, it aims to minimize the total variance:

$$\sum_{i=1}^{K} |S_k| \sum_{x^{(i)} \in S_k} \|x^{(i)} - \overline{x}^{(k)}\|^2 \tag{2.1}$$

where $\overline{x}^{(k)} = (\overline{x}_1^{(k)}, \ldots, \overline{x}_m^{(k)})$ is the array with means of each input x_i respective to the k-th cluster. This is done by replacing the centroids in each step with the respective mean array, repeating it until the centroids no longer change positions, which has guaranteed convergence [Hastie, Tibshirani e Friedman 2009].

In this work, we used the Python SKLearn implementation of K-Means, choosing $K = 2$ as a parameter, as also used in related works [Yang et al. 2018, Peppes et al.

2021].

Logistic Regression Classifier is a supervised regression method used for classification, as the obtained function is a sigmoid, which returns real values within an interval between 0 and 1. Given inputs $x = (x_1, \ldots, x_m)$, the predictor have the form:

$$\hat{p}(x) = \frac{1}{1 + e^{-g(x)}} \tag{2.2}$$

where $g(x) = x^T\theta$, being $\theta = (\theta_1, \ldots, \theta_m)$ a parameter array. These parameters are obtained through the minimization of the loss function:

$$L(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)}\log(\hat{p}^{(i)}) + (1 - y^{(i)})\log(1 - \hat{p}^{(i)})\right], \tag{2.3}$$

where $\hat{p}^{(1)}, \ldots, \hat{p}^{(n)}$ are the predicted values, $y^{(1)}, \ldots, y^{(n)}$ are the actual values (0 or 1). In the logistic regression, the predictor outputs probabilities, values in a range between 0 and 1, of the actual class being true. This can be used for binary classification, where the classes are determined if the function returns a value under or above 0.5 [Aggarwal et al. 2015].

K-Nearest Neighbors (KNN) classifier is an algorithm for classification tasks. It assigns a class label to a new data point based on the majority class label of its nearest neighbors in the training dataset. The algorithm involves selecting a value for K, finding the K nearest neighbors using a distance metric (usually Euclidean distance), and predicting the class label based on the majority of votes among the neighbors. KNN is an intuitive approach that assumes that similar data points tend to belong to the same class. SKLearning's KNeighborsClassifier is an example implementation of KNN, where K=5 by default.

Support Vector Classification (SVC) algorithm is a supervised learning method that aims to find an optimal hyperplane that separates different classes in the data. It works by mapping the data points into a higher-dimensional feature space and finding the hyperplane that maximizes the margin between the classes [Abe 2005]. The support vectors, the data points closest to the decision boundary, are crucial in determining the optimal hyperplane. SVC effectively handles non-linearly separable data using the kernel trick to transform the data into a higher-dimensional space implicitly. During prediction, SVC assigns new data points to classes based on which side of the decision boundary they fall. The key goal of SVC is to achieve a good generalization by finding the hyperplane that minimizes the classification error and maximizes the margin between classes in the transformed feature space.

Gradient Boosting is a supervised method that joins classification and regression trees, predictors which classify data through certain binary conditions upon thresholds and value ranges to determine the class it belongs to, associating it with some predicted score. This method works by joining trees in sequence, where a tree is trained

with the previous residual error, summing up their outputs to final results [Aggarwal et al. 2015].

Each tree can be written as a predictor in the form $\hat{y}_k = f_k(x)$, where x is the inputs array, f_k represents the k-th tree, and $\hat{y}_k$ represents the output score. In ensemble methods, as in the case of Gradient Boosting, the final outputs are determined by:

$$\hat{y} = \sum_{k=1}^{K} f_k(x), \tag{2.4}$$

being $f_1, f_2, \ldots, f_K$ the trees obtained by the algorithm. Gradient boosting obtains the trees and their parameters θ through the minimization of an objective function in the form:

$$obj(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \omega(f_k). \tag{2.5}$$

In this objective function, the first sum is a loss term, where $x_1, \ldots, x_n$ are the train inputs, $y_1, \ldots, y_n$ the target values, $\hat{y}_1, \ldots, \hat{y}_n$ the predicted values, and $l(y, \hat{y})$ a convex loss function between predicted and actual values. The second sum is a regularization term, being $f_1, \ldots, f_K$ the tree structures and $\omega(f)$ some measure of tree complexity. Gradient Boosting is distinguished from other ensemble methods by using a gradient-based approach, training each tree iteratively with the previous results.

2.5.2 Regression

Ridge Regression is a type of regularized linear regression. It consists of a linear estimator trained with a regularized loss function, using the called Ridge or Tikhonov Regularization technique, which aids in avoiding over-fitting and influence from outliers [Hastie, Tibshirani e Friedman 2009]. Given inputs $\mathbf{x} = (1\ x_1\ x_2\ \cdots x_m)^T$, the technique aims to obtain a regressor in the form:

$$\hat{y} = \sum_{i=0}^{m} \theta_i x_i = \theta^T x, \tag{2.6}$$

where $\theta = (\theta_0\ \theta_1\ \cdots\ \theta_m)^T$ are numerical parameters. Given $x^{(1)}, ..., x^{(n)}$ train feature arrays and $y_1, ..., y_n$ train target values, these parameters θ are obtained through minimization of the Loss Function:

$$L(\theta) = \sum_{j=0}^{n} |y_j - \theta^T x^{(j)}|^2 + \frac{1}{2}\alpha\|\theta\|^2 \tag{2.7}$$

The first term in Equation 2.7 represents the usual least-squares linear regression loss function, and the term $\frac{1}{2}\alpha\|\theta\|^2$ is called regularization term, being $\alpha > 0$ a fixed parameter and $\|\theta\| = \sqrt{\sum \theta_i^2}$ the usual Euclidean Norm.

Support Vector Regression (SVR) is a method that implements ideas from the Support Vector Machines into regression problems. The final estimator is in the form $\hat{y} = \theta_0 + \sum_{i=1}^{m} \theta_i x_i = \theta^T x$, being $x = (1\ x_1\ \cdots x_m)$ the input, $\hat{y}$ the predicted value and $\theta = (\theta_0\ \cdots\ \theta_m)$ an array of numerical parameters. In this model, instead of fitting an estimator which minimizes the error $|y - \hat{y}|$, the algorithm tries to fit an estimator such that the maximum possible inputs are in a specified $\varepsilon > 0$ error margin: the estimator fits a maximum quantity of $x^{(j)}$ inputs respecting $|y_j - \theta^T x^{(j)}| \leq \varepsilon$. The θ parameters are obtained by solving the minimization problem:

$$\min_{\theta,\zeta} \quad \frac{1}{2}\|\theta\|^2 + C\sum \zeta_i \tag{2.8}$$

$$\text{s.t.} \quad |y_j - \theta^T x^{(j)}| \leq \varepsilon + \zeta_j, \tag{2.9}$$

where $C > 0$ is a fixed regularization parameter, and $\zeta_i \geq 0$ are called slack variables, implemented to consider margin violation points.

Gradient boosting, used for the classification task, can be adapted for the regression. It applies the same principle of tree ensembles and training, but instead of a score related to a classification task, the prediction consists of the actual value for regression itself [Chen e Guestrin 2016].

2.5.3 Artificial neural network

Artificial neural network (ANN) was developed to copy the human brain's ability to process information in an attempt to make an algorithm capable of self-learning with what it has already analyzed. They can be used to analyze large amounts of data collected during the testing process and identify patterns or anomalies that may indicate faulty products or performance issues to assist in predicting product quality, estimating failure rates, and optimizing industrial procedures [Lei et al. 2020].

In an overview, an ANN is essentially conceived by two components: its architecture and its learning algorithm. The first is based on the number of attributes that will be analyzed and the desired outputs. The second has the function of generalizing the input data, memorizing what was understood within the adaptive parameters of the network (its weights). In this way, the developer must define which type of network is suitable to solve the problem in question and the best algorithm to train the network, which best adapts to the network weights.

An ANN is composed of neurons, but the amount and type of combination between them vary according to the chosen network, which may have one or several layers of neurons. The network topology can be composed by combining all inputs with all neurons from the subsequent layer, passing through the network weights, or by partially combining inputs with some neurons from the next layer, followed by passing the combination through an activation function.

Typically, the problem to be solved defines the constraints on the types of networks and possible learning algorithms. Networks can propagate information forward (data flows through the network from the input layer to the output layer performing computations and passing the results to the next layer) and/or backward (the network adjusts its internal parameters recurrently, with feedback from outputs to inputs). Conversely, algorithms can be divided into the supervised and unsupervised types previously described at 2.5. It is customary to initialize the network weights randomly, so the learning algorithm is responsible for updating them by going through a fixed number of iterations and/or until reaching a stop condition. Among the types of neural networks, the most used networks in the related works (3) and in this dissertation are explained in more detail below.

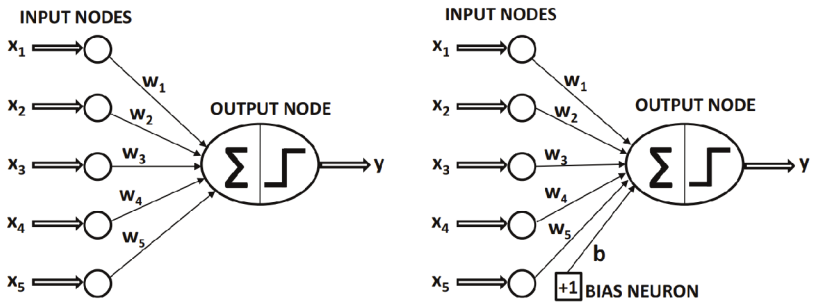
Each dataset has its peculiarity. For example, in the Iris Data Set, one of its three classes is linearly separable from the other two, while the others are not. In cases like this, if the linearly separable class is analyzed exclusively with one of the other two, it is possible to develop a neural network with a single layer, such as a perceptron. A larger number of layers in the network is necessary to separate the other two classes, such as a multilayer perceptron (MLP). The similarity between the two networks is that the neurons of the layers after the input have a direct relationship with all the neurons of the previous layer. The output of a neuron considers the values of all neurons in the previous layer, and each neuron produces only one output. Based on Haykin e Network 2004 and Aggarwal et al. 2018, the difference between the two networks is described below:

Perceptron: is the most simplified form of an ANN. It is formed by only one layer responsible for calculating the synaptic weights and the bias related to the dataset, as shown in Figure 2.6. It can classify only linearly separable datasets, and the number of output neurons is intrinsically related to how many classes the network can classify. In this network, the inputs (X) represent the information of the process to be mapped (which may contain a bias), and each of the inputs will have a synaptic weight (W) that represents the importance of each input concerning the output value desired (y). The sum of the weighted inputs will be added to the activation threshold and then passed as an argument of the activation function, resulting in the desired output. For binary classification (0 or 1), normally, the activation function used is the step function.

As seen before, the perceptron can only be used in binary classification problems with linearly separable classes, and the Iris dataset consists of three different classes with only one separable (Figure 2.4).

MultiLayer Perceptron: is a natural evolution of the perceptron. Its network contains one or more hidden layers, making it difficult to visualize its learning process. The architecture of multilayer neural networks is feed-forward; they successively feed each other in the direct direction from input to output (Figure 2.7). In its structure, each

Figure 2.6 – The basic architecture of the perceptron.



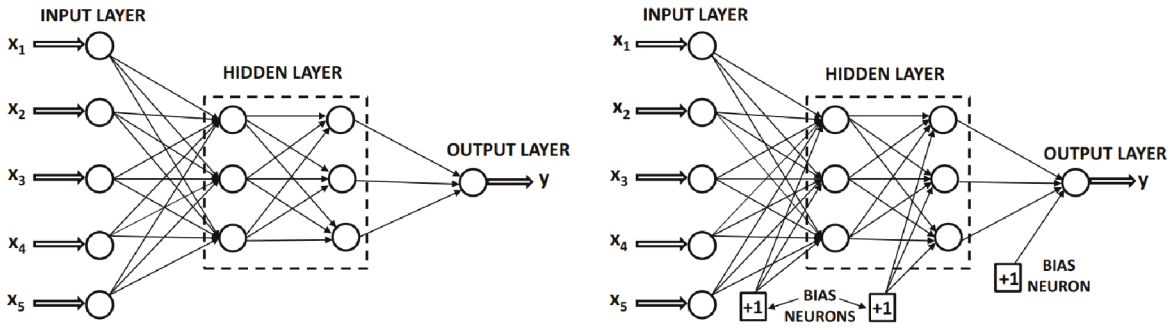
(a) Perceptron without bias (b) Perceptron with bias

Source: Aggarwal et al. 2018

layer has a defined function. The input layer carries the input attributes, and the hidden layers work as filters, where their weights encode the characteristics received in the input, allowing the network to create its representation of the problem, compiled and more complex. On the other hand, the output layer receives the stimulus from the hidden layer and reconstructs the created pattern, which will be the network response. In addition to the layers already mentioned, it is common to use linear outputs with a squared loss for real-valued prediction and soft-max outputs with a cross-entropy loss for discrete prediction.

Distinct from the perceptron, the MLP can be used in non-binary classification problems with non-linearly separable classes.

Figure 2.7 – The basic structure of a feed-forward network with two hidden layers and a single output layer.



(a) No bias neurons (b) With bias neurons

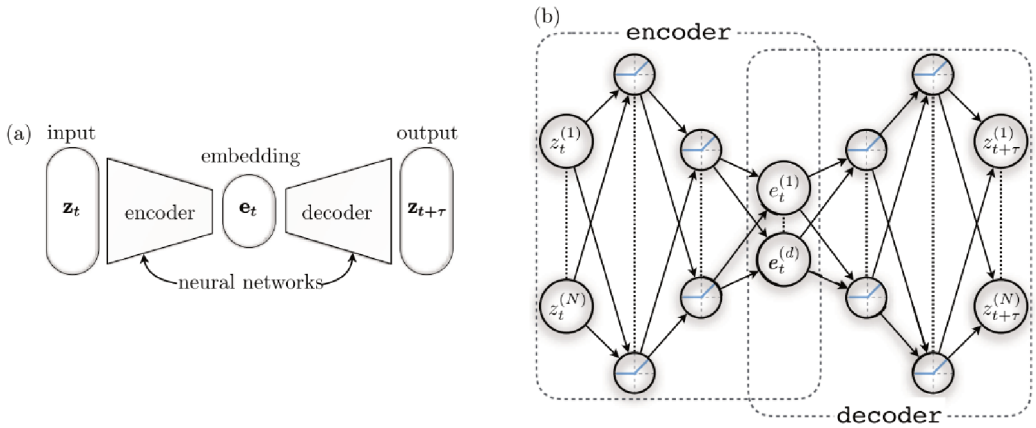
Source: Adapted from Aggarwal et al. 2018

Autoencoder (AE) is unsupervised learning neural network trained to copy their inputs to their outputs, not literally, but in such a way that it learns an approximation from the input to the output of the network. The AE generally has several hidden layers

and aims to compress the received data to find the basic relationship between them. Thus, hidden layers have fewer dimensions than their inputs and outputs. This behavior can be seen as a restriction of the network so that it does not exactly generate the input data as output but rather that it can make a correlation between them.

Fig 2.8 illustrates a general structure of an autoencoder. As can be seen, it is organized into two distinct parts, as if it were two mirrored neural networks, and is divided into two main phases: the encoder and the decoder [Goodfellow, Bengio e Courville 2016] [Wehmeyer e Noé 2018].

Figure 2.8 – Schematic of a time-lagged autoencoder.



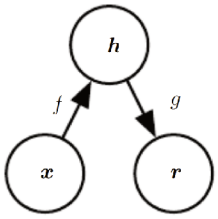
Source: Adapted from Wehmeyer e Noé 2018

Encoder: is the part of the network that compresses the input into a latent space representation (encoding the input). It can be represented by an encoding function h=j(x) (Figure 2.9). After encoding, the AE is expected to find the essence of the inputs, which is usually the information of interest to the application. In other words, the AE should learn only the necessary variations to reconstruct the input, finding the key information among all the data used in the training so that it has a good generalization to other input data.

Decoder: is the part that reconstructs the input from the latent space representation. It can be represented by a decoding function r=g(h) (Figure 2.9). For an application, it is expected that the AE is sensitive enough to the inputs to generate a good reconstruction at the output, considering a reconstruction loss, but, at the same time, not so sensitive to avoid overfeeding or an exact copy of the inputs.

Autoencoders can be used in numerous applications due to the encoding it does with the input data, correlating it accurately. After the model is built, if tested with a dataset under normal conditions (similar to training), the decoding error will be low, with a good data reconstruction. On the other hand, if tested with data under different conditions, the error in the reconstruction will be greater, serving to highlight anomalies,

Figure 2.9 – The general structure of an autoencoder.



Source: Goodfellow, Bengio e Courville 2016

for example. Autoencoder is commonly used in dimensional reduction, mainly in images, but also found in dataset attribute reduction.

Nonlinear AutoRegressive with Exogenous Inputs Recurrent Neural Network (NARX-NN) is a type of autoregressive model that uses previous predictions in addition to other features and its previous values to evaluate the future ones, integrated into Neural Networks architectures. NARX consists of an autoregressive model which supports nonlinear estimators and uses other features beyond the actual time series values, in this context called Exogenous Inputs, as described by [Kanarachos, Mathew e Fitzpatrick 2019].

Given a time series with values $y_1, \ldots, y_T$, exogenous values $x^{(1)}, \ldots, x^{(T)}$ with each $x^{(i)}$ having inputs on the form $x^{(i)} = (x_1^{(i)}, \ldots, x_m^{(i)})$, and a nonlinear estimator f, the NARX model is a predictor in the form:

$$\hat{y}_{T+1} = f(\hat{y}_T, \hat{y}_{T-1}, \ldots, \hat{y}_{T-N}, x^{(T+1)}, \ldots, x^{(T-N)}), \tag{2.10}$$

Where N is the lag or order of autoregression, specifying the number of previous inputs in use, and $\hat{y}_i$ is the series predicted values. The NARX Recurrent Neural Network (NARX-NN) uses a regressor neural network as the f estimator. The training of this model consists of the regular training of the f, considering all the listed inputs and setting the actual series values as the autoregressive training inputs [Kanarachos, Mathew e Fitzpatrick 2019].

2.6 FEATURE SELECTION

FS is an essential step in data and fault analysis. Without variables representing the studied phenomenon, its detection or abstraction becomes difficult [Silva, Gracioli e Araujo 2022]. The data analysis can be affected by either a lack or an excess of attributes, so it is interesting to build the smallest possible subset of features that represent all the vital attributes of the input data. This step aims to avoid overfeeding the machine learning models and also to clean the data sets to remove harmful, redundant, or noisy variables, to generate better generalization, learning, and scalability for the

applied algorithms, to reduce the computational cost and improve overall performance [Zebari et al. 2020] [Kiktova et al. 2014].

Like machine learning algorithms, FS algorithms must also be able to make a relationship between data. For this, the learning techniques (supervised and unsupervised) seen above also apply to these algorithms [Li et al. 2017].

The selection process is cyclic. First, a strategy is chosen for creating subsets of the available attributes. This can be done by evaluating individual variables according to pre-defined criteria or directly subsets [Kumar e Minz 2014]. A statistical evaluation takes place to verify which one is more qualified, this process usually has a stop condition, and the best result is the one that prevails until further evaluation. As a result, among all tested subsets or variables, one or more is expected to outperform the complete set to be used instead.

In this work, this process was implemented. It showed tangible differences, allowing the identification of the most relevant features for accurate predictions and providing valuable information about the underlying patterns and relationships in the data.

2.7 PARTIAL CONSIDERATIONS

The topics covered in this chapter make it possible to connect them and raise some considerations about what is treated during the work. The ECU is the control center for the vehicle's electronic systems, critical in managing engine performance, emissions, and overall functionality. Understanding its internal operations and utilizing the ECU's ability to present engine behavior is essential to reaping its benefits.

Combustion engines continue to be the predominant energy source in automobiles, and their efficiency and performance directly impact fuel consumption and emissions. By studying combustion engine fundamentals such as ignition processes and fuel combustion characteristics, valuable information can be gained to optimize fuel efficiency, reduce environmental impact, extend vehicle life, and ensure excellent safety for drivers.

Integrating machine learning algorithms in the automotive domain has immense potential to optimize various aspects such as fuel efficiency and fault detection. Taking advantage of the applicability of algorithms and including ECU data, models can be developed to improve the capabilities of automotive systems, mainly related to engine performance and driver driving.

Misfires can harm engine performance, fuel consumption, and emissions. Investigating the causes and methods of detecting misfires is crucial, and developing machine learning algorithms that can accurately identify and diagnose these problems in real time is a relevant alternative. This knowledge can lead to immediate maintenance and mitigation strategies to prevent further damage.

Fuel consumption is a significant concern for both economic and environmen-

tal reasons. Developing predictive models using machine learning algorithms allows accurate estimates and fuel consumption optimization. By considering various factors such as driving conditions, engine load, and vehicle characteristics, these models can suggest strategies to improve fuel efficiency and reduce carbon footprint.

In conclusion, integrating machine learning algorithms with a deep knowledge of automotive systems, considering ECU data, engine operation, misfire peculiarities, and fuel consumption, offers excellent potential to improve vehicle performance, reduce emissions and optimize fuel efficiency. By leveraging the insights gained from these considerations, further research and development can be undertaken to reap the full benefits of machine learning in the automotive industry, such as the work presented below.

3 RELATED WORK

ML algorithms have been explored in automotive-related research recently, mainly because of their relevance in developing fuel consumption models based on real-world driving data, which can improve the accuracy of engine-related parameters and reduce the need for physical tests [Pavlovic et al. 2020, Claßen et al. 2021, Ma, Shahbakhti e Chigan 2023].

In the following subsections, we review the literature considering three main related topics: (i) misfire detection [Qin et al. 2021, Devasenapati, Sugumaran e Ramachandran 2010, Rath et al. 2019]; (ii) classification of the driver's driving profile in terms of fuel consumption [Zheng et al. 2022, Yang et al. 2022, Peppes et al. 2021]; and (iii) regression models to estimate the fuel consumption, including analyses of related factors (environmental, traffic, driving behavior) [Li et al. 2019, Liu e Jin 2023, Rios-Torres, Liu e Khattak 2019]. Compared to human-designed analyses of engine data, ML algorithms speed up the processing with a greater amount of data [Ping et al. 2019].

3.1 MISFIRE DETECTION

Most mechanical methods use an angular velocity of the crankshaft or derivatives to detect misfires. A Kalman filtering can be applied to a signal from conventional angular speed sensors used by engine management systems in a 4-cylinders engine [Kiencke 1999]. These techniques require a state-space model of the signal and second-order statistics. Based on the premise that at high engine orders, the pressure amplitudes and load torque decrease, that the load of the cylinders affects the power density of the pressure torque, and that at each TDC of the engine, the pressure torque has vanished, a torque fluctuation was obtained in the application of this filter. The obtained model shows that during a deceleration phase, the engine has no differences in the pressure torque between the cylinders, and the engine's first-order inertia torque is zero. In the results, it was observed that in the case of a misfire, the drive-train oscillations were excited, and the torque fluctuations stimulated by this component were effectively suppressed. The approach estimation limit of engine speed is between 3000 and 4000 rpm because the oscillation of mass torques is bigger at higher speeds.

An algorithm based on analyzing the crankshaft speed fluctuation signal produced by a magnetic sensor placed on the engine shaft was used to detect misfire in a 4-stroke engine [Montani e Speciale 2006]. Four components related to the torque are expected to balance the engine, one positive corresponding to the cylinder that had expanded, one negative related to the compression phase, and two associated with the intake and exhaust phases. When a misfire happens, the positive one becomes smaller than usual, provoking an imbalance and reducing the engine speed suddenly. The proposed method is divided into some parts. First, a Continuous Wavelet Transform (CWT)

filter is used to suppress frequency components unrelated to the event and emphasize the misfire; in this way, a threshold value can be defined to determine the misfire. All the signal and their respective wavelet transforms are in the angular domain. However, the principal oscillation frequency is almost the same in the time domain. In the Wavelet domain, the engine speed and load are sufficient to categorize a post-misfire pattern. Considering the maximum amplitude of the signal that passes through the threshold, a normalization parameter can be obtained, and it represents the attenuation of the post-misfire transitory. A maximum correlation is made by comparing the post-misfire transitory and the calculated one to detect multiple misfires. The analysis of the delay of this maximum, respecting the first misfire instant, determines if and when a second misfire happened. The method was tested in a speed range between 1500 and 4000 rpm because the overall detection of the previously developed method was usually good at low speeds but difficult at high speeds. In this case, the algorithm was also good at high speeds, separating the misfire easily and unambiguously. For single misfires, 94% of the cases were detected, and the two types of multiple misfires tested succeeded in over 90% of the cases, independent of engine speed.

Considering the difference in kinetic and potential energy between samples and deriving the torque that causes the acceleration and deceleration of the crankshaft, an algorithm to detect misfire and compression fault through an energy model by estimating the indicated torque of the engine from instantaneous angular velocity was implemented [Tinaut et al. 2007]. It is important to highlight that the energy state of the pistons and the connecting rods are determined by the angular positions of the crankshaft and the mean angular speed, so changes in the pressure affect the instantaneous angular speed of the crankshaft and the pressure waveform can be obtained from this information. The kinetic energy from pistons, connecting rods, crankshaft, and load is explained and considered in the development of the energy model, and a deterministic function of the angular position and the total kinetic energy of the system becomes proportional to the square, angular speed to estimate the torque. The changes in kinetic energy should be equal for each cylinder in a perfect engine, and in a faulty cylinder, such as the one with a misfire, the energy was directly affected in it and the next cylinder in firing order. The model was evaluated with two indicators: one evaluating the relative change in kinetic energy at the compression stroke and the other at the expansion stroke. In a healthy engine, these indices should be equal to one for all cylinders, while in a faulty condition, the combustion energy index would decrease, and the compression index of the next cylinder in the firing sequences would increase as a result of less energy release in the combustion of the faulty cylinder. The energy indices were tested on a 4-cylinders engine by simulation of fault cases, and the proposed method was tested on a 4-cylinder 4-stroke engine at 1500 rpm. The influence of the misfire was clearly noticed at low mean angular speed, but at high mean angular speed, it was a lot more

complicated. Misfire was induced by decreasing the fuel quantity in the cylinder. This event decreases the energy index in the cylinder and increases the energy in the next one; this effect is seen in the greater use of the kinetic energy available for compression in the next cylinder. This result was also achieved in a simultaneous misfire in two consecutive cylinders and two alternative cylinders. The system behavior showed that in normal conditions, the expansion indices oscillate around 1, suggesting no significant loss of power in the cylinder, and, in an engine misfiring, the corresponding expansion index indicated the fault clearly. The algorithm could detect the faults correctly in an engine at low mean angular speed and at high mean angular speed or under transient conditions.

In perfect conditions, all cylinders operate identically, and the engine speed is a periodic function of crank angle, resulting in a waveform composed of the firing frequency and its harmonics, which are the major orders. In this way, a 6-cylinders diesel engine, with firing sequence 1-5-3-6-2-4 and 4-strokes, at a speed of 1200 rpm, was used for misfire analysis [Hu, Li e Zhao 2011]. Using two magnetic pickups, the instantaneous engine speed was used by measuring the time between consecutive pulses generated by a cycle reference signal. Other components result from the difference in the correct operation of all cylinders and can be filtered out to identify a fault, such as a misfire. Experiments were implemented, and misfire was simulated in cylinders 2 and 5. The idea was to observe the abnormal fluctuation signal (AFS) and the filtering process result. AFS showed concave trends where the misfire occurred, indicating that the torque was smaller than normal. It implemented an algorithm for misfire detection by partitioning the AFS into Z-dimensional vectors. These vectors were processed by multivariate statistical analysis methods, principal component analysis (PCA),) and the information obtained reveals the signal's N principal components (PCs), which explain the total variance of the vectors mentioned above. If the engine operates in normal conditions, the AFS would be just random noise, and the total variance would be distributed equally among the N PCs. Otherwise, under misfire occurrence, the variance would be explained by only the first PC, representing the most important trend of the vectors. To test the algorithm, two different multiple misfires were stimulated by slacking off the connector of the injection pump and the high-pressure fuel line and introducing a leakage in the fuel supply: a separated misfire (in cylinders 2 and 5) and an adjacent misfire (in cylinders 5 and 3). The results showed that if the cylinder does not completely work or the external load is high, almost all misfire events can be located correctly by the method, but in high speed and low loads, this percentage decrease to 90% since it becomes more difficult to identify adjacent misfiring cylinder events because it gets hidden by the second misfire. It is important to note that this method was established only for steady-state operating conditions.

A 4-cylinder inline 4-strokes engine model was mounted to simulate and under-

stand the misfire [Xu et al. 2019]. Four cases were tested at a 2000 rpm speed engine: normal conditions, single misfire, intermittent double-cylinder misfire, and continuous double-cylinder misfire conditions. The generalized force at the gravity center of an engine was calculated with information about gas explosion force, reciprocating inertial force, and rotating inertia force parameters, and the frequency, amplitude, and phase of the acceleration signals were found through the interpolation method of the discrete spectrum (Hanning window method). Having a firing order of 1, 3, 4, and 2, all cylinders should contribute equally to the engine torque. In this way, only the main harmonic orders would remain in the harmonic structure of the resultant torque, and when a misfire occurs, these harmonic orders change. The fault was simulated by setting the cylinder pressure so that an explosion did not occur. In this instance, when a misfire happens, the balance of the engine is destroyed, changing the corresponding harmonic and its multiples. In the experimental phase, three triaxial acceleration sensors were used to measure the vibration above the mounts, and three other uniaxial acceleration sensors were used to measure the vibration under the mounts. The result demonstrated similarity with the simulation, evincing the aptitude of the method to identify the misfire by analyzing the main harmonic orders of the generalized force.

Several authors approach the problem by having as the main analyzed characteristic of the engine vibration caused by the misfire. A locomotive engine vibroacoustic signals underwent a nonlinear analysis to detect misfire [Boguś et al. 2003]. The data was collected using a piezoelectric sensor, and the misfire was simulated by disconnecting one of the locomotive engine's cylinders. Three rotational velocities were used in the experiments with three measurement phases: normal conditions (all cylinders working), simulated misfire with cylinder 1 disconnected, and simulated misfire with cylinder 4 disconnected; the sensor was always in cylinder 1. The signal was measured in three channels, representing the parallel to main locomotive axis direction, the horizontal-transversal direction, and the vertical-transversal to main locomotive axis direction, and was processed with spectral analysis and nonlinear analysis, based on deterministic chaos theory, different from the previous studies that used Fourier analysis. The Lyapunov exponents were used to measure the system's trajectory to determine in which phase space it would converge and diverge, parameters that, when applied, can differ from one system state from another. The results show that, generally, in the scenario with all the cylinders working normally, the exponents are bigger than in the scenarios with a disconnected cylinder, which confirms the presence and the correct identification of chaotic components and their use to diagnose the misfire. Just as an addendum, in the cases that the tendency was not applied, the main problem was the influence of noises and the difficulty of eliminating them without damaging important signal components.

A 6-cylinder in-line 4-stroke diesel engine was mounted on a testing bench for

monitoring 7 vibration positions on the engine block and cylinder head with a piezoelectric accelerometer [Ftoutou e Chouchane 2008]. The fuel was injected directly into the combustion chambers in the order: 1-5-3-6-2-4, and a misfire was caused by disconnecting the fuel supply of cylinder 1, simulating a blockage in the injector hole. The statistical features Peak to Peak value (PP), Root Mean Square value (RMS), Crest factor (CTF), and Kurtosis factor (KF) were selected to analyze how misfire affects the vibration signal directly. In addition, the Distributions Overlapping was calculated to extract the maximum information of the averaged features variation and their ability to detect and isolate the simulated fault. The results were analyzed by calculating the averaged features PP, RMS, CTF, and KF at 700 and 1300 rpm. In the first one, the PP features better detected and isolated misfires at positions 5 and 6. RMS could isolate the fault in positions 1, 2, 3, 4, and 6; features CTF and KF failed in this scenario. In the second case, it was noticed that the variation of the features increased, and the isolation of the misfire was possible in positions 1,2, 3, and 4, with all features except C. Otherwise, in positions 5 and 6, only PP and RMS features could isolate the fault.

Discrete wavelet transformation was applied to decompose the signal of a piezoelectric accelerometer positioned on the top of the cylinders into low and high-frequency [Aihua et al. 2008]. The engine used in the experiments was a diesel 4-cylinders in line 4 stroke (sequences 1, 3, 4, and 2 of combustion). The misfire occurred in three different forms: single cylinder fault, double cylinders faulting continuously, and double cylinders faulting alternately, all simulated interrupting the diesel flow. The misfire signal is difficult to distinguish direct from a normal signal, but the experiments showed that this segregation is facilitated after 3-level decomposition with a db3 wavelet. Also, it was possible to identify the difference between the fault signal and the normal one in high-frequency. The outcome indicates that the combustion causes high-frequency vibration on the top of the cylinder. However, the wave crest fades from the high-frequency signal when a misfire happens. Therefore, the failure can be effectively detected by comparing the wave crest of the high-frequency signal and the low-frequency signal.

A statistical method, Z-freq, was formulated to detect misfires in a 4-cylinders in line 4-strokes hybrid electric vehicle (HEV), exploring the application of standard deviation and KF [Ngatiman e Nuawi 2018]. This method provides a two-dimensional graphical representation of the measured signal frequency distribution, calculating the distance of each data point from the signal centroid. In the first phase, the study focused on collection data under normal and faulty conditions from four piezo-film sensors located at each engine cylinder wall when the data was filtered and decomposed using Fast Fourier Transform (FFT) into two main categories: affix (low frequency) and anmex (high frequency), categories which can show the patterns of the studied conditions. In the second phase, it performed a signal analysis with the measured data to prove the method's effectiveness, calculating the Z-freq coefficient and a 2D representation graph

correlating the categories for a better understanding. Features like RMS, peak, and PP measurements of acceleration, velocity, and displacement were inputs to the monitoring machine. In conclusion, it was observed that the value of the Z-freq coefficient dropped by half when a misfire occurred, showing that the vibration behavior is altered, being greatly reduced in these situations.

Using an integrated approach to observe four accelerometers, adopting frequency analysis as a mechanism to extract features, using Fourier Transform to convert a signal from a time domain to a frequency domain and extract information like frequency and amplitude from the original signal, misfire was observed in a 4-cylinders in line 4-strokes engine [Jafarian et al. 2018]. Only the unique features related to the fault were selected to be applied in the classifier, so it could find patterns to predict three different faults: the slight misfire, the severe misfire, and abnormal valve clearance. The k-fold cross-validation method was used to train three classification techniques: Artificial Neural Networks (ANN), Support Vector Machines (SVM), and K-Nearest Neighbor (KNN). A fourth-order low-pass filter was used to eliminate high-frequency noises, cutting the frequency at 100 Hz, getting 16 features extracted from the sensors and 4 eigenvalue processed features from the multi-channel signal. Accuracy, sensitivity, and specificity were used as performance metrics in six scenarios of binary classification for fault diagnosis. The classification results showed that the SVM had better performance in all cases of test with a 98% accuracy. In addition, when analyzing only one cylinder each time, the accuracy was 100% in all three techniques, confirming the suitability of vibration signals for fault detection.

Mechanical rotating systems have a base frequency proportion to the engine speed and can cause harmonic vibrations. Misfire was introduced into a 2-cylinder 4-strokes motorcycle engine by interrupting the signal to the ignition coil and taking samples from a broadband piezoelectric vibration sensor (a knock sensor) at a 0.1º crank angle resolution [Rath et al. 2019]. In this way, the vibration signal was acquired without a discrete high-pass filter, unlike it would happen if the knocking noise were analyzed because the lower frequency components are useful to misfire detection. Knock noise introduces vibrations in the engine operation, while misfire can influence the amplitude of existing vibration, and it is impractical to map these influences to a physical model in an ECU, although a signal model focused on the knock sensor signal can be used to monitor changes in engine behavior or parameters, like the lack of increase in the cylinder pressure at the engine's power stroke, which was considered the ground truth to define whether a misfire event happened or not. An auto-regressive model was constructed to reduce the number of parameters to describe the knock sensor signal. A linear combination of the previous values was expressed as polynomial filter coefficients with the signal's properties estimated from the measured knock sensor signal by solving the Yule-Walker equations and a stochastic noise term. As a result,

the analyses of these coefficients, on average, allow the distinction between a normal engine operation and a misfire event, but it can not be done with a simple threshold in the coefficient because it has a high variance, not avoiding wrong misfire detection.

Artificial intelligence and its tools have been increasingly used in detecting misfires, applied in various ignition engines, and exploring variables already mentioned and many others. Wavelet packet transform (WPT) was used to make a multi-resolution analysis, all resolutions with the same frequency bandwidth, breaking up detail and approximation versions [Wu e Liu 2009]. WPT generalizes the wavelet transform, and it also has a time-frequency function. The sound signal measured was broken into four resolutions, producing sixteen sub-spaces to be computed. To decompose the signal, db4, db8, and db20 were applied. After the data reprocessing of WPT, Shannon entropy was used to extract features from the signal. The low frequencies were discarded, given its large quantity of noise. A neural network algorithm was used to classify the fault, receiving the feature vector resulting from Shannon entropy as input. In the same way as the previous work, a back-propagation algorithm with three layers was developed, but due to its uncertain convergence, a general regression neural network was also constructed. The main difference is that the general regression is not similar to the regression analysis that needs to suppose an exact function. It only needs the method of probability density function to be presented. The experiments were done in a 6-cylinder, 4-stroke gasoline engine, a microphone was used to measure the sound, and the engine's revolution was captured by an optical encoder close to the crankshaft. Six events were tested to estimate the fault diagnosis: an engine without any fault, air leakage of the intake manifold, camshaft sensor fault, electronic control thermal sensor fault, one cylinder misfiring, and two cylinders misfiring. The WPT proved to provide a good resolution when the signal changed unexpectedly. Both back-propagation and general regression algorithms were tested in the experiments, and all the recognition rates were over 95%, showing that the WPT is useful in fault classification.

A decision tree algorithm identified a misfire in a 4-cylinder 4-stroke gasoline engine [Devasenapati, Sugumaran e Ramachandran 2010]. The misfire was simulated by cutting off the high-voltage electrical supply from the spark plug of one or more cylinders. An accelerometer mounted on the center of the engine head was used to capture vibration signals, and a DACTRON FFT analyzer converted the signal from analog to digital and fed computer software to process and extract features. At the 1500 rpm rated speed of the engine, five types of conditions were measured: normal condition and with all the cylinder misfiring, one per time. Initially, statistical features were selected: standard deviation (a measure of the energy content of the vibration signal), standard error (the standard deviation of the sampling distribution, revealing how much sampling fluctuation the statistic has), sample variance (variance of the signal points), KF (the flatness or the spikiness of the signal), Skewness (the degree of

asymmetry of a distribution around its mean), the sum of all signal points, range, minimum value and maximum value of the signal. Not all of these features were obligatorily occupied; a decision tree was used to select the most relevant. The C4.5 Algorithm was the decision tree used. It played two different roles in the method. First, it was used to select the features and, afterward, to classify them. Before it could be used, the tree went through a construction phase. The training sample is partitioned recursively until all records in a partition have the same class and a pruning phase, removing the least reliable branches for better classification performance. In general, features well in discriminating stay in the decision tree. Using the optimized parameters in the tree, the classification performance of the tested data set at 1500 rpm was 94.14%. However, up to a maximum speed of 2000 under extreme conditions, the misfire classification has an accuracy of 68%. These results are particular to this application, and its generalization to other applications is impossible.

An artificial neural network (ANN) was implemented to detect the misfire of a turbocharged diesel engine on a dynamometer test bench [Liu et al. 2013]. Engine running parameters at normal and misfire conditions were measured to compose a data set that could be used as data inputs to train the ANN. Most of the information was collected directly from the engine sensors, and some with the input of other sensors. An ANN is a mathematical model that functions like a human brain to learn and generalize data. The network was used given its affinity for handling data with non-linearity and its capacity to work with multi inputs and output data. A backpropagation algorithm was used to detect the misfire, and it was composed of three layers: the input, hidden, and output layers. The network works in two ways, the input layer receives the data vector and feeds the hidden layer to the outputs; in the other direction, the calculated error from the comparison of the output value and the expected value is propagated backward so that the network parameters can be adjusted so in the next iteration the error becomes smaller, repeating until an acceptable error is found. The study's main objective was to select the most relevant parameter to build the training vector, a misfire fault topology was built to support this idea, facilitating the understanding of the chain reaction caused by the misfire and allowing the definition of variables that can be considered in experiments. Initially, six variables (engine speed, intake temperature, intake pressure, exhaust temperature, coolant temperature, and fuel consumption) were selected for the tests, and randomly 80% of the data measure were used to train the network, while the rest were used to verify its correct operation. The result shows that the ANN could detect all normal condition engines, but some misfire detection was wrong. It was noticeable that the input vectors were not good enough, so, analyzing the fault again, the in-cycle speed variation was added because of its difference during normal and misfire conditions. The ANN has trained again with this new vector, and the results improved as expected. The detection of misfire and normal events were

all correct, without any misdetection, showing that the neural network can be used for this purpose, also indicating that the choice of the training vectors may be even more important than the method used or the structure of the network.

Two acquisition systems were built to evaluate two different methods of misfire detection, one using vibration signals and another using acoustic signals [Firmino et al. 2021]. The experiments were developed with a 4-cylinder 4-strokes gasoline/ethanol engine. To detect misfires, the vibration and the sound emission were measured to test both methodologies. For the acoustic analysis, a sound acquisition device, made with an Arduino Due board and a MAX4466 microphone, was placed 30 cm from the engine block to record the sound emission from the engine. The vibration acquisition system, made with an Arduino UNO and an MPU6050 accelerometer, was placed in the middle of the engine block. The misfire was simulated by disconnecting the spark plug from each cylinder, causing five failure scenarios plus one of the normal conditions. It is important to emphasize that both acquisitions were performed simultaneously so that they could have the same conditions. Since the problem was a non-linearly separable pattern, an ANN was chosen because it could be constructed with more than one layer. A multilayer perceptron was developed with three layers: an input layer (in which was performed a linear transformation), a hidden layer (when a hyperbolic tangent function was applied, limiting the range of the data between -1 and 1), and an output layer (with a softmax function). FFT was used for both vibration and acoustic analysis, transforming the time-domain signal into a frequency-domain signal and extracting features to the ANN. The vibration data had 8 features (4 dominant frequencies and their amplitudes), and the acoustic data had 20 features (10 frequencies and their amplitudes). In addition, the signal energy was calculated and considered as another feature for both. One ANN for each analysis was built and trained, and some tests to validate the model were done using different data. Both methodologies were efficient for fault detection, vibration with an accuracy of 100% and acoustic with 87.5%.

A new method called deep twin convolutional neural network with multi-domain input (DTCNNMI) was created for misfire detection in a diesel engine [Qin et al. 2021]. The framework was composed of many layers; the most important ones were: convolution, batch normalization, pooling, flatten, concatenated, and fully connected layers. This structure combined automatically extracted time-domain, time-frequency-domain information, and hand-craft time-domain statistical features. Following are some of the features: rectified mean value, mean value, peak value, RMS value, K, PP value, clearance factor (CF), shape factor, and margin factor, among others. Based on the premise that the working state of an engine is reflected in the vibration signal and also that the misfire event should affect this vibration directly, the vibration signal of cylinders heads under different working conditions (single misfire of all cylinders, combinations of multiple misfire and normal condition) were recorded to feed the input layers, for analysis

and to create 4 data sets (A, B, and C with single misfire and D with multiple misfires) for testing. The model was tested in all data sets, A at 1300 rpm, B at 1800 rpm, C at 2200, and D also at 1800, and achieved an accuracy of 79.922%, 85.333%, 82.196%, and 94.275%, respectively. The DTCNNMI proved to be adaptive to different engine conditions, managing to extract the necessary characteristics for misfire detection, with good results in all scenarios.

A real-time abnormality detector developed with a convolutional neural network (CNN) was proposed [Shahid, Ko e Kwon 2022]. A magnetic pickup sensor was used to measure the angular speed of the engine and to represent the sensor signal better; it was transformed into a crank angle degree (CAD) signal that translates the behavior of the combustion strokes. The CNN was designed to detect irregularities in this CAD signal in each cycle of the internal combustion diesel engine. For this purpose, the CAD signal was filtered to remove some noise components. The CNN required a simple array as input and low computational power. A constructive approach was used to define the number of kernels (filters) in convolution and fully connected layers to optimize the network's generalization. Other parameters were defined according to the training results. The CNN has five layers: a convolution layer (convolves the input data and extracts the features from it), an activation layer (generates the output features, determining which neurons from the network should be activated), and a max pooling layer (reduces the complexity and avoid over-fitting) for feature extraction and also a flatten layer (transform the feature matrix into a vector to fed the next layer) and a fully connected layer (get the output out of the network) for classification. In addition, a softmax function is used to convert the final output to a probability for each class of the classification options. The method accurately detected and classified multiple irregularities in real-time. To evaluate the CNN, metrics like accuracy, Matthews Correlation Coefficient (MCC), and cross-entropy loss were used. The experiments were run in a 4-stroke marine diesel engine at a speed of 720 rpm under different loads, and three classes were considered: normal state, change in engine load, and fault condition (misfire). From almost 500 samples divided into these classes, the trained CNN was able to predict all samples from classes 1 and 2, while only one sample from class 3 was labeled as class 2, with an accuracy, MCC, and cross-entropy loss of around 0.998, 0.997 and 0.019, respectively.

3.1.1 Misfire Detection Summary and Comparison

The analyzed literature is diverse, and monitoring and detecting misfires is a topic that has advanced over the last decades. The latest studies focused on applying technologies and computational power for a better understanding of it. Table 3.1 summarizes the above studies most related to this work. We can note several scenarios regarding this failure; various engine components and external factors such as environ-

ment and drivability can cause its occurrence. Different types of analysis were observed (mechanical, vibration signals, and machine learning analyses).

Table 3.1 – Summary of misfire detection techniques

Source	Main Features	Used Techniques	Evaluation Metrics
[Shahid, Ko e Kwon 2022]	CAD signal	A five layers 1D CNN	Accuracy (%) 99.7
[Qin et al. 2021]	Time-domain statistical features (rectified mean value, mean value, peak value, RMS, KF, P, CF, shape factor, and margin factor) and wavelet packet energy features	Deep twin convolutional neural networks with multi-domain input	Accuracy (%) 79.922-94.275
[Firmino et al. 2021]	4 dominant frequencies and their amplitudes from vibration signal and 10 frequencies and their amplitudes from the acoustic data, plus the signal energy from both	A three-layer back propagation neural network	Accuracy (%) vibration with 100 and acoustic with 87.5
[Liu et al. 2013]	Engine speed, intake temperature, intake pressure, exhaust temperature, coolant temperature, fuel consumption, and in-cycle speed variation	A three-layer back propagation neural network	Not applicable
[Devasenapati, Sugumaran e Ramachandran 2010]	Statistical features (standard deviation, standard error, sample variance, K, Skewness, sum of all signal points, range, minimum value, and maximum value of the signal)	Decision tree (C4.5 Algorithm)	Accuracy (%) 94.14 at normal speed and 68 over 2000 rpm and under extreme conditions
[Wu e Liu 2009]	Shannon entropy features extracted from sound emission signal	Wavelet packet transform and artificial neural network	Not applicable
[Rath et al. 2019]	Autoregressive coefficients and mean knock sensor signal power	Autoregressive Coefficients Analysis	Not applicable
[Jafarian et al. 2018]	8 frequencies and 8 amplitudes	FFT as a feature extraction methodology	Accuracy (%) 97.18-98.40, Sensitivity(%) 95.57-97.44 and Specificity (%) 99.06-99.47
[Ngatiman e Nuawi 2018]	Amplitude, frequency, phase, energy variation, statistical features (PP, RMS, CTF, and K) from the signal and the Z-freq coefficients	Z-freq analysis	Not applicable
[Aihua et al. 2008]	High-frequency and low-frequency signals and their wave crest signal from a cylinder-head vibration	Wavelet analysis	Not applicable
[Ftoutou e Chouchane 2008]	Statistical features (PP, RMS, CTF, and K) from the vibration signal	Vibration analysis in the time domain	Not applicable
[Boguś et al. 2003]	Vibroacoustic exhaust locomotive engine	Nonlinear analysis with Lyapunov exponents	Not applicable
[Xu et al. 2019]	Gas explosion force, reciprocating inertial force, rotating inertia force parameters, frequency, amplitude, and phase of the acceleration signals	Generalized force at the gravity center	Not applicable
[Hu, Li e Zhao 2011]	Abnormal fluctuation signal	Multivariate statistical Analysis	90% partial misfire location
[Tinaut et al. 2007]	Change in kinetic energy at the compression and at the expansion stroke	Energy model	Not applicable
[Montani e Speciale 2006]	Crankshaft speed fluctuation	Wavelet-based analysis	94% single misfire recognition
[Kiencke 1999]	Torque fluctuation and pressure torque	Kalman filter	Not applicable
[Author, 2023]	Basic engine components, air Control, fuel and richness control, ignition advance, engine torque, alternator, and battery management	ML algorithms (XGBoost, Gradient Boosting, K-Means, K-Neighbors, Logistic Regression, and Support Vector Classification)	Precision (%), Recall (%), and F1-Score (%), best results: 92.40, 96.16, and 94.24, respectively

Source: Author (2023).

Compared to the models found in the literature, the machine learning models developed in this work offer a simpler computational approach, using classifiers instead of complex neural networks or sound/vibration analysis. A notable advantage of these

models is their consistent accuracy, reduced variability, and greater resilience to environmental changes such as vehicle speed or engine speed. Notably, the developed models were exclusively trained using authentic, real-world data from a dedicated test vehicle, avoiding reliance on publicly available datasets or simulations, as most related works do. Although the performance metrics align with the models presented in the literature, the models developed in this document are stable in different environments, resulting in more reliable accuracy and F1 score outputs with computational efficiency, requiring less computational resources. One aspect that draws attention in this work is that the best-performing model uses the XGBoost algorithm, which is less prevalent in the existing literature, and a solid application for it has been discovered. Another point of comparison with the literature is that the training process of the models in this work involved a meticulous resource selection procedure covering thousands of variables, different from what we found in other works that use a limited number of variables. Therefore, we found a strong relationship between some of them, which further minimizes the variability of the results when subjected to substantial input changes. Furthermore, unlike the literature, the developed models suggest that fault detection can be effectively achieved through classification algorithms, eliminating the exclusive reliance on anomaly analysis, the need for engine sound/noise data, or more complex algorithms. In summary, the models proposed in the work use variables commonly present in the ECUs of most vehicles on the market from the most diverse subsystems involved. In this way, we managed to reduce the costs of sensors and components related to fault detection, reducing the dependence on specific sensors, such as vibration sensors, using algorithms that are easily interpretable when compared to neural networks and using data acquired in real-time and directly from the ECU for analysis.

3.2 MACHINE LEARNING CLASSIFIERS FOR DRIVING BEHAVIOR AND FUEL CONSUMPTION CLASSIFICATION

Driver behavior is one of the biggest factors determining vehicle fuel consumption; therefore, driver profile analysis can make a difference in the car's economy when coupled with accurate feedback. The difference in fuel consumption reaches up to 30%, depending on the driver [Liimatainen 2011]. The works discussed below explored different techniques for analyzing the driver's profile and classification.

The classification of driver styles and behaviors can be made from vehicle-obtained data. Sometimes, it is desirable to make the classifier decision based on a short time window. A trained algorithm able to classify new observed drivers in different driving styles, inferring the style and comparing it to existing classes determined by previous analysis and clustering was suggested [Zheng et al. 2022]. Divided into two phases: a training one, where previous data is pre-processed and used to determine clusters or classes, and an inference one, where the real-time processed vehicle data

fits in one of the previously determined classes. After processing the data with filtering and dimension reduction, the k-means clustering algorithm was chosen to perform in the training phase. Data taken from the NGSIM, an open data set of cars and drivers on United States roadways, was used in this study for the algorithm test. Vehicle positions, velocity, and acceleration variables were selected for clustering. Using the data, three different clusters taken from the Calinski–Harabasz Score were founded and classified by the researchers as conservative, aggressive, and experienced drivers. The inference of which driver belongs to each class was made by a softmax function of vehicle data to cluster center distances, comparing the ratio belonging to some class probability and the sum of all belonging to each class probability. Based on k-folds cross-validation metrics, where some parts of data were used as a test set, and the remaining as training, accuracies between 70-90% were found relative to each k-size chosen, based on the f1-score, precision, and recall metrics.

The driver's habits mainly influence safe driving behaviors but also may be identified and classified by the results of reactions observed on the vehicle. [Lattanzi e Freschi 2021] proposes in their study an objective way of labeling driving habits as safe or unsafe, based on the relationship between vehicle's accelerations and velocity, and develop learning tools for classifying the driving behavior through only in-vehicle sensors data. For the experiments, using data from an open dataset, selected variables usually available in most car OBD-II interfaces, such as vehicle speed, engine speed, engine load throttle position, and less common available variables, such as steering wheel angle and brake pedal pressure. The classifying tools were developed using the Support Vector Machine and simple feedforward neural network techniques, which were tested using different combinations of used variables, taking into account the OBD available variables alone and later combined with the non-standard variables. Applying a 5x2 cross-validation, the most prominent results were shown by combining all of the explored variables, and no significant differences were found in comparing the different trainers. The obtained results show an accuracy of around 90% in both techniques.

Vehicle consumption and pollutant emissions are related to the driving profile. With this premise [Peppes et al. 2021] created an interface able to gather car sensor data and use it in Machine and Deep Learning algorithms, classifying the driver behavior as pro or non-eco-friendly, comparing the results and algorithms performances. It made an interface where the obtained data is sent to an OBD-II decoder, which transmits by Bluetooth communication to a smartphone. The smartphone then sends the collected data to a cloud platform, processing and analyzing it. The cloud platform has two main components: a streaming module and a big-data analysis and management module. The streaming module manages the data flow and communication between the cars and the data consumers, using the Apache Kafka platform to assist. The hybrid big data management and analysis module hosts both a relational and a non-relational

database and an analytic engine. After the data preprocessing through the PySpark module, an unsupervised clustering ML algorithm is applied to the current data set, with the view to adopt a data-labeled approach, based on the rpm and speed OBD values. The selection of these two attributes from the data available was performed because the vehicles deployed in this study are equipped with fueled-powered internal combustion engines (either gasoline or diesel), as it is widely known the fuel consumption of such engines is highly reliant on speed and rpm. The K-means algorithm was selected for clustering and classifying data. Three traditional machine learning algorithms were implemented and compared, i.e., logistic regression, Support Vector Machine (SVM) and Random Forest (RF) as well as two deep learning methods, and more specifically the MLP and the RNN algorithms. The aforementioned algorithms were benchmarked using the following metrics: loss, validation loss, accuracy, validation accuracy, F1 score, the area under the ROC curve, precision, recall, and execution time. It was found that all explored algorithms show similar results in the used metrics, getting near 100% accuracy, being the more explicit difference noted in the execution time, whereas the Deep Learning algorithms show far slowest times.

In-vehicle data recorders (IVDRs) installed on 155 military vehicles of 8 different types were used in the analysis over a period of one year and with over 350 drivers, exploring the influence of IVDR-based feedback on driving behavior in three experimental phases [Toledo e Shiftan 2016]. The first stage (no feedback) lasted 16 weeks, and drivers did not receive any information or guidance about the device or any feedback. This step served to identify the drivers' driving characteristics, which provided a baseline for comparison. The second stage (feedback limited to the worst drivers) was active for 18 weeks, where only the least cautious drivers received initial guidance on the functioning of the IVDR and received feedback on their performance. The third stage (full feedback) lasted 16 weeks, during which feedback was provided to all drivers. Thirteen driving-related events with different severity levels were measured: extreme and intermediate acceleration, extreme, intermediate, and moderate braking, left and right turns, and speeding events. The two thresholds that differentiate moderate, intermediate, and extreme severity levels were determined based on the intensity of the measurements. Data was collected to analyze safety performance and fuel consumption. With these events, a scoring system was developed to evaluate each driver based on a risk index developed according to the frequency and severity of the events and on periodic individual driving reports. The score had three levels: green for good drivers, yellow for intermediate drivers, and red for unsafe drivers (demanding attention). As a result, the relationship between fuel consumption and event rate was estimated. Fuel consumption was measured using the total monthly duration of trips taken by each vehicle as recorded by the device. A linear regression model incorporates vehicle type and event rate as explanatory variables. Events are assigned different weights accord-

ing to severity level, and various combinations of event types were tested to examine which events are the best fuel consumption indicators. The impact of event rates on fuel consumption significantly differed between small and large vehicles, but there were no significant differences between vehicles of the same size. Different vehicle models have their engine, mass, and design differences taken into account. The results of the estimates showed that the ratio between the coefficients of extreme and intermediate events is above four. Overall event rates dropped from 4.12 events per hour to 3.30 from Stage One to Stage Two, a considerable reduction of 20%, while for Stage Three, it dropped to 3.01, another 7%. The results suggest that IVDR events can be used as risk and fuel consumption indicators. The differences between the first and second stages were explored with paired statistical tests based on 314 participating pilots, demonstrating a significant decrease in the event rate. However, a similar test between the second and third stages showed no significant difference. Regarding fuel consumption, a reduction between 2-6.5% was observed from the First to the Second phase and between 3-10% from the First to the Third phase, with greater reductions obtained for large vehicles. The percentages showed that the work is valid and that monitoring and feedback help and encourages drivers to improve their driving attitudes, directly influencing fuel consumption and driving safety.

In the same way, a fair measure of driver performance was developed to use the information in an eco-driving incentive system [Liimatainen 2011]. Driver performance measurement equipment was installed on 12 buses, six of which are two-axle buses with a capacity for 81 passengers and six two-axle buses, three with a capacity of 94 and three with a capacity for 98 people. The article tested three hypotheses: first, that driver-independent factors have a considerable effect on the fuel consumption of buses; second, that the effect of these factors can be isolated to allow a fair comparison of drivers based on fuel consumption; and finally, that fuel consumption can be used as a performance indicator in an eco-driving incentive system for drivers if driver-independent factors are isolated. The information was collected by a data logger connected to the vehicle's CAN bus. Data were recorded by the data logger asynchronously with a recording every 1-2 s, collecting data such as vehicle identification number, fuel consumption, braking incidents, engine speed, vehicle speed, distance traveled, and time of operation. These raw data were collected for each run. Firstly, buses were mainly used on two routes, and approximately 26,000 km/month were traveled on both. In the second phase, the two routes were divided into six directions. With this subdivision, the differences in fuel consumption became greater. For some directions, consumption is up to 25% higher than for others. The differences are due to different road geometries (mountains/plains), road types (main roads/small streets), and bus types (two axles/tandem axle), confirming that the first hypothesis is true. In the third phase, the data were divided into time groups according to the beginning of

each route. Fuel consumption varies considerably during the hours, with a difference of up to 20 L/100 km between the less busy early morning hours and the afternoon rush hour. Differences in the number of passengers, stops, traffic lights, and other vehicles on the road cause variation. During large-scale tests, routes were assigned to 82 directions, ten-time groups, and two types of buses, forming 1640 comparison groups from November 2007 to May 2009. of 200 km per month were used to increase the reliability of the analysis, a total of almost 74 drivers per month. Two correlations of interest were highlighted. The first is between the reference consumption of each driver and the percentage of savings for the month under analysis. The correlation was weak (-0.15), indicating no considerable relationship between these variables. This means that the analysis method isolated the driver's effects on fuel consumption from the effects of external factors, confirming the second hypothesis. The second correlation was between a month's savings percentage and the mileage-weighted average of the previous 3 months. Pearson's correlation was high (0.79), indicating that an individual driver's savings percentages were fairly continuous at the same level. Average monthly consumption has decreased each year by 1.4-4.6%, suggesting that just awareness that fuel consumption is monitored can make drivers drive more economically. After analyzing the data for a large-scale test period, it can be stated that differences in driver savings percentages are based on driver behavior and not on external factors, suggesting that the third hypothesis is also true.

[Yang et al. 2018] developed an adaptive driving-style-oriented equivalent consumption minimization strategy for HEV. HEV usually has available management systems responsible for optimizing energy consumption by switching between fuel and battery. In most cases, this change is made by referencing an equivalent factor of consumption, which is a constant in most of these vehicles. Knowing that driving style influences energy consumption for each source type, the authors suggest adapting this equivalent factor based on the driver's behaviors. They first propose a statistical pattern recognition based on kernel density estimation and entropy theory to classify the driver's style from the instantaneous way of driving. It tests the probability that each driving style belongs to some level of aggressiveness based on prior and posterior probabilities calculated after training the parameters with previous data. For their studies, an experiment was done where drivers with different styles were invited to test a simulator in a virtual scenario, being only oriented to follow a specific car shown in the simulation, programmed to perform a usual Chinese bus track. Vehicle speed, throttle position, and vehicle acceleration data were chosen for the training and tests dataset, recognized by literature as usual parameters for classifying driving styles. Then, the obtained data in the experiments were analyzed to identify the factors and relations between driving styles and fuel consumption. The analysis showed that an aggressive driving style is more related to higher fuel and battery spending. The authors interpreted

this as caused by higher demand for hybrid driving mode use and switching from driving charging mode, which leads to high battery consumption, less use of the electric motor, and more use of fuel to compensate and charge. Yet, higher demand for acceleration and deceleration leads to more switches of engine ON/OFF state, requiring more fuel use. Moderate drivers, on the other hand, have more low power requirements and lead to optimal use of the driving charging mode of the vehicles. Considering these factors, the researchers proposed an adaptive Equivalent Factor method based on the actual style classification of the driver as more aggressive or moderate. Basically, the designed adaptive management system sets a higher Factor for a more aggressive style, which represents a higher cost of the electric motor energy use, and a lower factor, representing a low cost for electric energy. This leads the engine and electric motor to operate together optimally and more efficiently. Compared with the simulation patterns, benchmark tests made with a functioning test rig show that for moderate driver styles, the adaptive strategy reduces more than 10% of the fuel consumption for a low battery cost. For aggressive styles, the improvement is slower due to the higher energy demand but still succeeds in a better fuel economy performance and charging mode balance.

[Yang et al. 2022] suggests estimating the influence of penetration rate and position of aggressive drivers in the fuel consumption and emission means of each traffic setting. In the study, the authors collected data from an open dataset to classify driving styles as aggressive or moderate, using the K-means unsupervised learning method. The classification was made based on car-following modeling, where the chosen parameters are concerning the forward vehicle: relative distance, time-to-collision, time headway, and a safety margin coefficient. With the driving style classifications, an Long Short-Term Memory (LSTM) algorithm was used as Recurrent Neural Network to predict the velocities and positions of each vehicle based on the driving style, using the dataset parameters as the training set. These predictions were destined to perform simulations of different traffic settings in a single-lane scenario, where a leading vehicle was set with fixed settings, and the remaining were permuted in different arrangements. To simplify, the selected arrangements were the ones with aggressive-style vehicles always in adjacent positions, changing their penetration rate and positions related to the remaining moderate drivers. The resulting Fuel Consumption and Emissions rate was taken as the average calculated in the 50 simulated experiments, based on a so-called VT-Micro model of individual vehicle consumption and emission rate calculation. The simulated data analysis found that the leading vehicle state is an important factor in determining the correlation of aggressive vehicle positions and penetrations with Fuel Consumption and Emissions. When the leading vehicle has a positive Cumulative Speed Change, there is a strong positive correlation between the Fuel Consumption and Emission, and the effect is reversed with a negative Cumulative Speed Change,

with a strong negative correlation between the variables. Compared with a fixed benchmark, the penetration rate of aggressive drivers can result in up to 3% absolute changes in the emission, and the position results in up to 1.5% absolute change in the emissions.

[Ping et al. 2019] proposes two machine learning methods to classify and predict fuel consumption based on driver behavior and dynamic traffic data, using a spectral clustering algorithm and an LSTM recurrent neural network. In an experiment with fixed vehicle type, route, and weather conditions, 202 drivers were selected to drive with no additional task beyond tracing the specified route. Data was collected from 30 passenger cars, each with a gasoline engine and a six-speed automatic transmission. Different parameters were extracted from the vehicles and compared with the fuel consumption through the Pearson correlation coefficient to select the most relevant and influential ones for the clustering. In the end, data of speed, positive and negative accelerations showed the strongest correlations with a Pearson coefficient greater than 0.7; speed and acceleration variances show intermediate, with around 0.5 correlation coefficient, which have been selected for clustering. Data of gas pedal position, brake pedal position, and steering angle showed the lowest correlation with fuel consumption and were discarded for posterior analysis. The total fuel consumption calculation was used to integrate the instantaneous fuel consumption data obtained from the ECU, getting results with less than 6% estimated difference from real consumption. After treatments and processing to standardize dataset sizes, the obtained data were used for the unsupervised data feature extraction. The selected method was a Parallel Spectral Clustering Algorithm based on clustering data as a graph partitioning problem, where each data point (in this case, each driver per course) is an edge, and the connections between each point are weighted based on the distance, using parallel processing to improve the performance. Three clusters were found, with a 79.31% clustering accuracy, one of the drivers with low average speed and acceleration/power demand, a second with drivers showing low average speed but higher acceleration rates, and a last one with high speed and acceleration. These clusters, in the mentioned order, showed growing fuel consumption averages. As the clustering algorithm depends on long-term time windows data, an LSTM process is proposed to predict fuel consumption from short-time driving behavior observations. In addition to the driving behavior data obtained from vehicles, data gathered from visual sources with help from deep learning processes were used, providing information on on-road traffic factors, such as in-lane vehicles and pedestrian positions, and road structures, such as curves and intersections. Different algorithms and node quantities were compared with cross-validation training and testing with the used dataset. The LSTM with 150 nodes showed the best overall performances, getting an average of 81% prediction accuracy. The results showed that the suggested algorithms might have interesting results in predicting fuel consumption based on drivers' behavior and the surrounding information.

3.2.1 ML Classification Summary and Comparison

Table 3.2 summarizes the strategies used and the variables monitored in related works. In the present work, clustering and classification algorithms were developed and compared, and the results point to a correct division between the measured driving data, separating economical and non-economical drivers with accuracy, precision, and recall around 100%, varying between the experiments done. In general, we developed a work that combined the best practices noted above, correcting flaws such as data inconsistency, less accurate results, communication with the cloud, and real-time execution.

Table 3.2 – Summary of related works that use classification techniques.

Source	Main Features	Used Techniques	Evaluation Metrics
[Lattanzi e Freschi 2021]	Vehicle speed, engine speed, engine load, throttle position, steering wheel angle, and brake pedal pressure	Support vector machine and feedforward neural network	Accuracy(%): 90
[Peppes et al. 2021]	Altitude, bearing, speed, RPM, intake air temperature, engine temperature, throttle position, fuel, and engine runtime	Logistic regression, support vector machine, random forest, multilayer perceptron, and long short-term memory (recurrent neural network)	Accuracy(%): 98.2, 100, 100, 99.8, and 100 (respectively)
[Liimatainen 2011]	Fuel consumption, braking, RPM, speed, distance, and time operating	Correlation analysis and hypothesis testing	Not applicable
[Toledo e Shiftan 2016]	Braking, lateral acceleration, speeding, safety-related events, and fuel consumption	Feedback based on In-Vehicle Data Recorders	Reduction of 8% in safety incidents, and 3–10% in fuel consumption with feedback, a larger reduction obtained for large vehicles
[Yang et al. 2022]	Position, speed, acceleration, time-to-collision, time headway and safety margin	K-means (classifying), LSTM (prediction/simulation), and correlation analysis (fuel consumption/traffic settings)	Pearson Correlation Coefficient (Positive/Negative lead speed change): fuel consumption rate: 0.997/-0.997, fuel emission rate: 0.999/-0.965, and fuel emission of aggressive drivers: -0.99/0.9932
[Zheng et al. 2022]	Vehicle positions, velocity, and acceleration	Short-term observations algorithm	Based on K-folds cross-validation metrics, accuracies between 70-90% were found
[Yang et al. 2018]	Speed, throttle position, and acceleration	Density Estimation and Entropy Theory (probability estimation) and Rate Comparison	Fuel consumption Reduction (%): aggressive driving style: 1,8 - 4 and moderate driving style: 2,5 - 12
[Ping et al. 2019]	Speed, acceleration, brake pedal position, throttle position, and steering	Spectral Clustering Algorithm (clustering) and RNN - LSTM (predicting)	Clustering accuracy rate (%): 79.31 and Prediction Accuracy (%): 83,6
[Author, 2023]	Speed, acceleration, engine speed, engine temperature, engine air load, torque, throttle valve position, accelerator pedal position, battery voltage, clutch pedal position	K-means (clustering), Logistic Regression, and XGBoost (classifiers)	Accuracy (%), precision (%), and recall (%): 93.55, 100, and 91.70 in clustering; 100, 100, and 100 in regression; and 100, 100, and 100 in boosting.

Source: Author (2023).

Compared to the models described in the literature, the ML models for driver profile classification developed in this work demonstrate notable distinctions. While the selected algorithms for classification are commonly used in the field, the key differences lie in the variable selection process and the real-time application using the onboard ECU, ensuring data integrity. One significant advantage of my models is their slightly su-

perior performance, with some models achieving 100% accuracy, precision, and recall. Moreover, the main advantage is the real-time application, allowing classification while the vehicle is in motion. Unlike some literature approaches that rely on synthetic data or publicly available datasets, our models were trained exclusively with data acquired by the author, directly from the automotive ECU. Additionally, the variables used were carefully selected through an exhaustive FS process, encompassing thousands of variables from various engine subsystems. In terms of computational efficiency, we developed algorithms that are computationally less demanding compared to certain models found in the literature, such as neural networks or algorithms that analyze data over time. In addition to providing real-time analysis, execution of models on a cloud server. A noteworthy finding is that the majority of variables identified through the FS process, strongly related to driver behavior, are common across a wide range of vehicles. This eliminates the need for additional sensors. Additionally, the ability to classify driver behavior in real-time provides instant feedback to the driver, enabling them to improve their driving habits. In contrast, a significant portion of the literature primarily focuses on offline analysis and provides feedback or classification after the driving session.

3.3 ML REGRESSORS FOR FUEL CONSUMPTION PREDICTION

To predict a specific value in the form of classification or regression, Support Vector Machine (SVM) algorithm can be used. [Hamed, Khafagy e Badry 2021] developed two methods applying SVM, one based on Revolution Per Minute-Throttle Position Sensor (RPM_TPS) and another based on Vehicle Speed-Mass Air Flow (VS_MAF), both based on a legacy dataset containing Onboard Diagnostic (OBD) data with 18 variables. The proposed model consists of four phases: data pre-processing, feature weighting, FS, and SVM prediction model. The performance of the proposed model is evaluated using the R^2 metric, where the VS_MAF had an R^2 of 0.97, and RPM_TPS had 0.96.

[Liu e Jin 2023] proposed a model using support vector regression (SVR) to predict fuel consumption by adopting a steady-state estimation and transient correction, using a dataset originated from the D3 database of the Advanced Powertrain Research Facility at Argonne National Laboratory [ANL 2022]. They constructed a model divided into two modules, first to reflect the transient state of the vehicle, using Vehicle Speed and Vehicle Acceleration, and second to reflect the impact of vehicle inclination and load on fuel consumption, using Engine RPM and Engine Torque. SVR was used to predict fuel consumption, and the modeling data was analyzed using the Gaussian Mixture Model (GMM) [Reynolds et al. 2009] to classify the driving conditions accurately. The results show the model's Root Mean Squared Error (RMSE) of 0.2137, the Mean Absolute Percentage Error (MAPE) of 13.6896%, and the Mean Absolute Error (MAE) of 0.1526.

Predicting vehicular fuel consumption by driving cycles can be helpful in economy strategies, especially for optimizing the Energy Management Systems of HEVs. Based on that premise, [Rios-Torres, Liu e Khattak 2019] investigates how much-predicted consumption can change between conventional vehicles and HEV in standard and personalized driving cycles, determined considering driver and vehicle information. Furthermore, the study seeks to evaluate how much these predictions can help to minimize HEV consumption by adjusting their energy management systems. For these purposes, 12 personalized driving cycles in certain routes were generated using a tool developed in previous work called Case-Based System for Driving Cycle Design. This tool uses driver demographic information and their driving style category, determined by a score based on the proportion of high acceleration changes and the route and vehicle data. The generated driving cycles have been separated into two categories of route, highway and urban, and three categories of driving style, divided into calm, normal, and volatile. Two standard driving cycles were selected for comparison, the Federal Urban Driving Schedule (FUDS) and the Federal Highway Driving Schedule (FHDS), both provided by the U.S. Environmental Protection Agency (EPA). Given each driving cycle, the fuel consumption prediction was made through physical and polynomial models for both conventional and hybrid vehicles, which accounts for data such as vehicle speed, acceleration, engine speed, torque, and battery power. The authors adjust the equivalent consumption minimization strategy to optimize HEV consumption, responsible for an optimal balance between battery and fuel usage. This is done by optimizing the called “equivalent factor”, a parameter that establishes how much the battery electrical energy converts to fuel-provided energy, by a model that considers the driving cycle patterns. Results show that in conventional combustion vehicles, only the calm driver personalized cycles showed consumption lower than the standards. In comparison, all personalized HEV cycles had lower consumption rates on the urban ones, besides an overall better performance than the conventional vehicles. However, the FHDS cycle still showed better performances than all three HEV driving styles personalized cycles. Prediction analysis on the optimized HEV shows that adjustments based on driving cycles can save up to 12% of fuel usage, with higher savings for urban scenarios.

Considering the potential benefits of estimating fuel consumption in transportation jobs for economic and environmental purposes, [Katreddi e Thiruvengadam 2021] compares different machine learning regression models in the task of consumption prediction using experimental data collected in heavy-duty trucks. They installed portable emissions monitoring system sensors in two trucks of the same model, used in several trips by drivers with distinct driving styles, driving through various routes and conditions. The sensors collected data with over 100 registered features by a 1 Hz frequency, totalizing over 600 thousand data records. After a primary selection based on previ-

ous studies, the remaining variables included trip number, engine speed, trip distance, vehicle speed, fuel temperature, fuel rate, accelerator pedal position, actual torque, power, and engine load. Total fuel consumption was determined by summing up the instantaneous recorded rates. Still, they performed a correlation analysis between these remaining features to avoid correlated independent variables on prediction and reduce the complexity of the training models. Finally, after applying an RFE FS method, engine load, vehicle speed, and engine speed were used for the model training. A Feedforward Artificial Neural Network (ANN), a Multiple Linear Regression, and a Random Forest (RF) algorithms were the selected models implemented and tested for comparison. They used a backpropagation optimization for tuning neuron parameters of ANN, using cross-validation with 70% of data for training and remaining for validation beyond a setting of 100 trees for the Random Forest model. Measuring the instantaneous consumption prediction in a test set data, ANN was capable of an R^2 score of 0.78. Concerning cumulative fuel consumption, ANN showed slightly better results than the other models, with a 0.78 R^2 and an RMSE of 0.0025 L, while Multiple Linear Regression showed 0.73 R^2 and 0.0029 RMSE, and RF had a 0.72 R^2 and RMSE of 0.0030 L. With another approach, [Yao et al. 2020] proposed three fuel consumption prediction models based on driving behavior data collected by smartphones and OBD terminals. The developed models were RF, SVM, and Back Propagation (BP) Neural Network. The data used in the article was collected from taxis and mobile phone terminals with a sampling interval of 1 second, 75% of the collected data was randomly selected as training samples, and the remaining data were used as test samples to evaluate the accuracy of the developed models. To evaluate the accuracy and efficiency of the models, they used RMSE, mean absolute percentage error (K), coefficient of determination (R^2), and model running time. The RF model obtained an RMSE of 0.783, K of 0.069, R^2 of 0.635, and a running time of 0.140 seconds. The SVM model obtained an RMSE of 0.888, K of .073, R^2 of 0.519, and a running time of 0.933 seconds. The BP Neural Network model obtained an RMSE of 0.872, K of 0.075, R^2 of 0.547, and a running time of 0.724 seconds. Additionally, the article highlights that using smartphones to collect data can reduce costs associated with installing OBD devices, which does not have much impact because using OBD in vehicles is mandatory. Compared to our work, [Katreddi e Thiruvengadam 2021] made a limited selection of features while we applied several different algorithms to find the common variables most closely related to fuel consumption. [Yao et al. 2020], however, used ODB data, which has a reduced sampling frequency when compared to the acquisition directly from the ECU, in addition to using data from the cell phone, which creates a dependence on one more extra component to those found in the vehicle.

[Ziółkowski et al. 2021] focused on the performance of MLP models in predicting fuel consumption using 12 variables (7 quantitative and 5 qualitative) as input

parameters. The methodology involves collecting data on the technical parameters of passenger cars fabricated between 2010 and 2020, creating a database for training the ANN, and setting input and output variables based on a literature review. The data used was obtained from international tests conducted by automotive factories. The work reports a high rate of training, testing, and validation for the MLP model with an accuracy of 0.93-0.95, RMSE values of 0.40–1.15, MSE of 0.2249-0.4010, and MAPE of 5%–11.5%. While the work reports high accuracy for the MLP model in predicting fuel consumption, it does not directly compare MLP and other ML models. Another work applied an MLP model for predicting real-world fuel consumption rates of light-duty vehicles [Li et al. 2019]. The model considers input parameters like external environmental factors, vehicle company manipulations, and drivers' habits. To train and test the MLP model, the authors used a dataset of real-world fuel consumption rates for light-duty vehicles, splitting it into training and testing sets by 70% and 30%, respectively. They also performed k-fold cross-validation. Overall, they tried to demonstrate that the article provides a valuable contribution to the field of ML to predict real-world fuel consumption rates, but it does not show any comparison or numerical evaluation that guarantees this compared to other works found in the literature.

[Kanarachos, Mathew e Fitzpatrick 2019] develop an estimation model of instantaneous fuel consumption in vehicles using data measured by smartphone sensors. The proposed model consists of a Deep Neural Network, using an approach called soft sensors, that returns estimations of measure through modeling based on other sensor's measures. The fuel consumption data were obtained from the OBD via the vehicle's CAN-bus. To construct the estimation model, the authors compared two existent types of Deep Neural Networks: Long Short-Term Memory (LSTM) and Nonlinear Autoregressive with Exogenous Inputs (NARX) networks. The second showed smaller errors and was chosen for the final model test, where they performed cross-validation and compared several training algorithms to test the optimization methods that better fit the data, including different gradient-based, population-based, and large-scale population-based methods. In general, all of them showed good results, with average errors below 1 km/l, but the contrast-based Fruit Fly Optimization presented better scores, with median squared errors down to 0.35 km/l, with variance between predicted and measured data showed scores up to 96%. Finally, they tested the model with different sets of available features to find the set that best predicts consumption. Although the complete set showed far better scores, which may indicate a very strong relationship between the model and the dataset, not necessarily a relationship between the variables and fuel consumption.

3.3.1 ML Regressors summary and Comparison

Table 3.3 summarizes the methods and variables used and the results obtained in related works. In our work, regression algorithms were developed and compared, and the results point to a correct prediction of fuel consumption, with metrics MSE $(l/h)^2$, R^2, and MAE (l/h) of 0.28, 0.99, and 0.23, respectively, varying between the models and experiments carried out. Overall, our work merged the best practices pointed out above, improving the FS, using data exclusively from the vehicle's ECU, and avoiding data inconsistency, so that the results have been improved or maintained, but communicating with the cloud, running in real-time and without the need to add more devices.

Compared to the models found in the literature, the fuel consumption prediction models developed in this work demonstrate significant differences. Firstly, we use algorithms that are easier to describe and require less computational cost than specific literary works that rely on neural networks. One key advantage of this work is establishing transparent relationships between automotive engine variables and fuel consumption. It explicitly identifies the variables that have a stronger or weaker influence on instant fuel consumption prediction when comparing different datasets for training and validation of the models. Compared with the literature, the models show superior results in some cases, particularly in the coefficient of determination (R^2).

In contrast, in other cases, it achieves comparable performance with the advantage of algorithm simplicity. It was also observed that some works do not bring many evaluation metrics, unlike this one. Moreover, in this project, the models were exclusively trained and validated using real-world data acquired by the author from an automotive ECU, a departure from a substantial portion of the literature that relies on publicly available datasets. This distinction ensures that the models are grounded in the intricacies of real-world fuel consumption patterns. In terms of computational efficiency, compared to neural network algorithms, my models require less computational power and a smaller volume of training data, in addition to enabling real-time instant fuel consumption prediction provided by the cloud server. One notable observation is that the XGBoost algorithm, despite being less prevalent in similar applications, proves to be highly effective for both classifications (as seen earlier) and regression tasks (including fuel consumption prediction). The developed models can also predict instant fuel consumption without relying on historical data analysis, providing accurate predictions based solely on the data received during the vehicle's operation.

3.4 PARTIAL CONSIDERATIONS

This chapter presented an extensive review of relevant works in the application of machine learning in the automotive context, serving as a solid foundation for the development and validation of the study developed and presented in this document.

Table 3.3 – Summary of related works that use regression techniques.

Source	Main Features	Used Techniques	Evaluation Metrics
[Hamed, Khafagy e Badry 2021]	Vehicle's work period, latitude, longitude, altitude, barometric pressure, engine coolant temperature, fuel level, ambient, air temperature, engine rotation per minute, intake manifold pressure, amount of air entering the engine, air intake temperature, speed, ECU signaling response according to the chances of oxygen levels, throttle position, timing advance, air/fuel ratio	Support vector machine	The vehicle Speed-Mass Air Flow equation and Revolution Per Minute-Throttle Position Sensor equation had an R^2 of 0.97 and 0.96, respectively.
[Liu e Jin 2023]	Vehicle speed, vehicle acceleration, engine speed, and engine torque	Support vector regression	MAPE less than 14%, MAE less than 0.16 cc/s, R^2 0.97, and RMSE 0.1164 cc/s
[Rios-Torres, Liu e Khattak 2019]	Vehicle mass, aerodynamic drag coefficient, vehicle frontal area, rolling resistance coefficient, tire radius, engine-rated power, torque, continuous power output, max torque output, max speed, battery-rated voltage, and battery capacity	Used physical models for estimating fuel consumption	Adjusts based on driving cycles can save up to 12% of fuel usage, with higher savings for urban scenarios
[Katreddi e Thiruvengadam 2021]	Engine load, vehicle speed, and engine speed	Feedforward artificial neural network, multiple linear regression, and a random forest	Artificial Neural Network, Multiple Linear Regression, and Random Forest: R^2 0.78 and RMSE of 0.0025 L, R-square 0.73 and RMSE 0.0029, and R-square 0.72 and RMSE 0.0030 L, respectively.
[Yao et al. 2020]	Vehicle speed, engine load, coolant temperature, fuel consumption rate, and GPS location	Random Forest, Support Vector Machine, and Back Propagation Neural Network	Random Forest, Support Vector Machine, and Back Propagation Neural Network: RMSE of 0.783, K of 0.069, R^2 of 0.635, and a running time of 0.140 seconds; RMSE of 0.888, K of 0.073, R^2 of 0.519, and a running time of 0.933 seconds; RMSE of 0.872, K of 0.075, R^2 of 0.547, and a running time of 0.724 seconds; respectively
[Ziółkowski et al. 2021]	Cubic capacity, the number of cylinders, the number of valves, maximum power, maximum torque, compression rate, the kerb weight of the vehicle, type of engine, fuel injection, type of charge, gearbox, and drivetrain	Multilayer Perceptron	Accuracy of 0.93-0.95, RMSE of 0.40–1.15, MSE of 0.2249-0.4010, and MAPE of 5%–11.5%.
[Li et al. 2019]	The specific inputs used in the model are not explicitly listed but include external environmental factors, vehicle company manipulations, and drivers' driving habits	Multilayer perceptron	Not applicable; evaluation metrics were not used.
[Kanarachos, Mathew e Fitzpatrick 2019]	Time, GPS position (latitude, longitude, altitude), speed, acceleration (longitudinal, lateral, vertical), and the number of visible satellites	NARX Recurrent Neural Network	MSE of 0.43 kpl, correlation (predicted/measured) of 0.96
[Author, 2023]	Speed, acceleration, engine speed, engine temperature, engine air load, torque, throttle valve position, accelerator pedal position, battery voltage, clutch pedal position	XGBoost, Ridge Regression, and Support Vector Regression	MSE $(l/h)^2$, R^2, and MAE (l/h): 0.37, 0.99, and 0.25 for XGBoost; 0.28, 0.99 and 0.23 for Ridge; and 0.38, 0.99, and 0.30 for Support Vector Regression, respectively.

Source: Author (2023).

Exploration of the literature that applies machine learning techniques to misfire detection, driver behavior classification, and fuel consumption regression provides valuable insights (e.g., variables, algorithm, methodologies) for our work, focusing on automotive ECU data analysis. Reviewing similar studies gives us insight into the methodologies, algorithms, and approaches used before, which can inform and guide our

research.

The positive aspects of related work are considered to improve our work. By identifying successful techniques, robust algorithms, and effective methodologies employed by previous studies, we incorporated and adapted these approaches to our specific goals to save time and effort by leveraging existing and proven knowledge.

In addition to the positive aspects, it is essential to consider and learn from the limitations and disadvantages identified (e.g., use of synthetic data, offline analyses, limited variables) in related works. By critically reviewing the past researchers' challenges, we can proactively address these issues in our work and develop strategies to overcome potential shortcomings. Understanding the limitations of previous approaches helps us create more effective solutions and design experiments to validate the performance and generalizability of our models.

Research gaps (e.g., identification of engine subsystems correlated with the analyses, changing in the set of variables) not adequately addressed or explored are identified by studying related works. These gaps present opportunities for contributing to the field by developing new techniques, proposing alternative methodologies, or applying machine learning algorithms in new ways. Understanding the current state of the art allows us to identify areas where this work can significantly impact and advance the existing knowledge base.

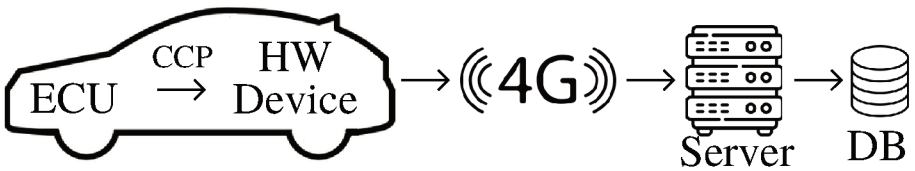
Related works provide a foundation (e.g., a background of the misfire and fuel consumption, influencing factors, and techniques) for validating and comparing the results obtained in the project with those of other researchers and, thus, evaluating the effectiveness and performance of our models compared to existing approaches. This validation process helps establish the robustness and reliability of the developments of this research, contributing to the credibility of the work in the application of machine learning in the automotive domain.

In conclusion, the insights gained from studying related work on misfire detection, driver behavior classification, and fuel consumption regression are invaluable in informing and guiding our research. By leveraging the strengths, addressing limitations, identifying research gaps, and validating our outcomes against existing work, we can contribute to creating new knowledge and advancing the field based on previous research.

4 MACHINE LEARNING APPLIED IN AUTOMOTIVE ECUS

In the following topics, details related to the development of the project and the experimental evaluations are discussed, explaining how the experiments were constructed, the data acquisition carried out, and the standards used, with an overview of the platform. Figure 4.1 illustrates the communication from reading the ECU variables to their respective storage on the server. More details are explained in the next sections. Other students integrated the project, each one was responsible for a part of the work, and it was essential that everyone collaborated and advanced together so that the general objective was achieved. In the course of the text, some activities of other members are mentioned because they complemented the development of this dissertation.

Figure 4.1 – Main parts of the communication structure.



Source: Author (2023).

To collect real data for the analyses proposed in this work, we used a car provided by Renault do Brasil, a Sandero 2019, model 1.0, with a four-stage four-cylinder spark-ignition gasoline engine. A developed acquisition hardware was connected to the engine ECU [Bedretchuk et al. 2023]. Thus, with the car available, it was possible to collect a significant amount of real data, which was used to train and validate the ML models. We did not use simulation data or public data sets, one of the assumptions differentiating this work from related works.

The acquisition system developed in IASE allows us to select and organize the target variables of the ECU through a graphical user interface (GUI), creating a configuration file that we call "essay", which is loaded in the hardware developed. This hardware, responsible for collecting the desired data directly from the ECU, was embedded in the vehicle via the Controller Area Network (CAN) Calibration Protocol. On top of the CAN bus, the ECUs usually support the CAN Calibration Protocol (CCP) or the Universal Measurement and Calibration Protocol (XCP), depending on the ECU model and manufacturer, to perform the data acquisition. Thus, the essay to read the variables from an ECU must respect the bandwidth limits imposed by the protocols. For

example, when using the CCP protocol in our car, the engine ECU communicates with the board at sample rates limited to 4, 5, 10, and 100ms. Each acquisition essay can also read about 160 variables using available sample rates.

This device communicates via 4G with the LISHA cloud server so that the data is processed and/or applied directly to ML models or any other type of treatment and manipulation, in real-time, with a greater computing power available, and stored in a database for later use, including in new studies and projects.

In our server, the ML algorithms are called workflows and are implemented in Python 3.8. When the hardware device is configured to read a variable from the engine ECU, the variables are marked with the workflows that must be applied. When the server receives the acquired data, it identifies which algorithms must be applied for that specific data. Furthermore, the data is also saved in the database for further analyses and/or visualization in a dashboard.

4.1 HARDWARE VALIDATION

One of the points that differ this study from most related works previously presented (3) is that all data used in the experiments were exclusively taken from the ECU, with the sampling rate limited by its capacity, which is lower than that of the sensors. A communication architecture involving software and hardware was structured and validated for this.

The IASE project's initial part was to build a device capable of communicating with the ECU through protocols such as the CAN Calibration Protocol and the Universal Measurement and Calibration Protocol. Throughout the evolution process, the work presented here has been developed in parallel with hardware advancements and upgrades. Every modification made to the hardware required rigorous testing using workflows (a definition that will be explained in the following topics). These tests covered several essential aspects: sampling frequency, data integrity, continuity, transmission time, latency, and data buffering.

In this way, this work not only fulfilled its purpose of data analysis but also played a crucial role in the industrial project involved, helping validate the hardware system integrated into the vehicle. This validation process ensured the accuracy of the hardware in terms of communicating with the automotive ECU and the cloud server, reliably delivering data with integrity and availability in real time. In addition to hardware validation, data acquisition experiments, carried out over a period of two years, played a vital role. They served to test specific features of the ECU and communication with the hardware, including data acquisition for different numbers of variables and time dimensions, further reinforcing its reliability and effectiveness.

Nevertheless, the validation process, combined with intelligent analysis, improves the project's value proposition, both from an engineering and market point of

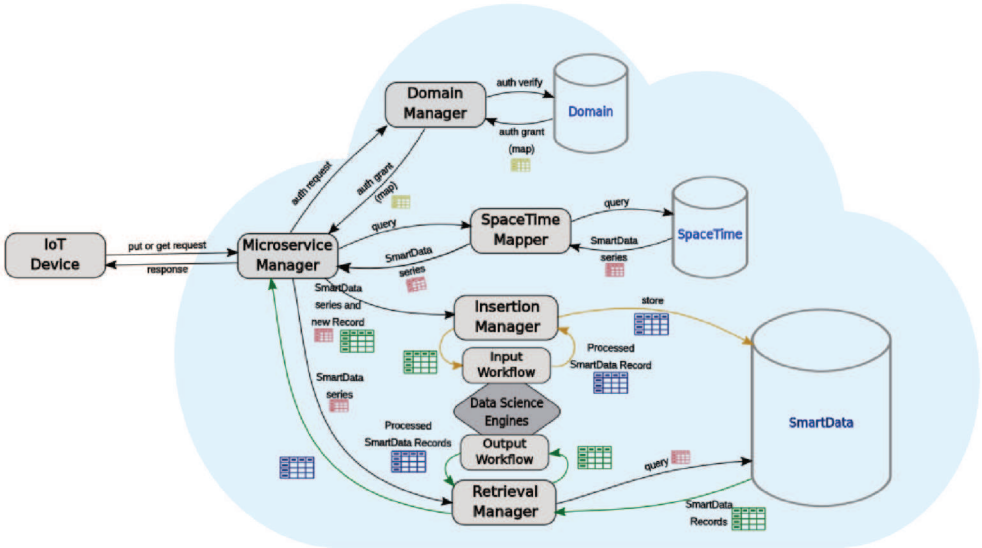
view, which aims to optimize cost efficiency in the production of vehicles, presented in the Journal Sensor [Bedretchuk et al. 2023]. By incorporating intelligent data analysis, made possible by this work, the vehicle gains new alternatives, such as detecting failures, improving fuel consumption, and streamlining engine calibration processes.

The validation process and intelligent analyzes performed with the hardware have far-reaching implications. Not only do they extend the advantages of the hardware, but they also increase the overall value proposition of the project. From an engineering standpoint, rigorous validation and smart analytics improve technological prowess. Furthermore, these advances can boost market appeal, positioning the project favorably for future commercialization if it interests Renault.

4.2 IOT PLATFORM AND WORKFLOWS

In an overview, the IoT Platform is organized as microservices (Figure 4.2) that provide secure storage and data processing for the SmartData series. The Microservice Manager is an IoT front-end, and the Domain Manager handles microservices requests, being responsible for authentication and mapping SmartData sets. The SpaceTime Mapper, on the other hand, maps regions of space and time to the SmartData that are stored or to be stored on the platform. The execution of data science algorithms on SmartData entering the platform are the responsibility of the Insertion Managers, and those leaving the platform are of the Retrieval Manager.

Figure 4.2 – IoT Platform Overview.



Source: LISHA (2023).

SmartData is characterized by a version, a unit, and source coordinates that

show where and when SmartData was produced, created, captured, sampled, etc. Stored on the Platform, it is a data point in a SmartData time series. The SmartData stored and processed by the platform has the following JSON representation:

SmartData 4.1 – SmartData structure.

```
{
    "version" : unsigned char
    "unit" : unsigned long
    "value" : double
    "uncertainty" : unsigned long
    "x" : long
    "y" : long
    "z" : long
    "t" : unsigned long long
    "dev" : unsigned long
    "signature": string
}
```

And have the following structure:

- **version**: the SmartData version:

  "1.1": version 1, Stationary (.1), representing data from a device that is not moving;

  "1.2": version 1, Mobile (.2), representing data from a device that is moving;

- **unit**: the type of the SmartData (see the SmartData documentation and typical units);

- **value**: the data value (e.g., the temperature measured by a thermometer);

- **uncertainty**: a measure of uncertainty, usually transducer-dependent, expressing Accuracy, Precision, Resolution, or a combination thereof;

- **x, y, z**: the absolute coordinates of the location where the data originated;

- **t**: the instant the data originated (in UNIX epoch microseconds).

- **dev**: a disambiguation identifier for multiple transducers of the same Unit and space-time coordinates (e.g., 3-axis accelerometer), "0" otherwise (i.e., if a single transducer is present);

- **signature**: a cryptographic identifier for mobile devices producing SmartData (only for version 1.2 / mobile).

SmartData series are classified based on the mode of operation: time-triggered or event-triggered. The former must define a period, while the others are considered event-triggered. The start of a series can be specified by time (by giving t0), by event, or manually (by not giving t0, which is then assumed to be the current time). Thus, the start of a time-driven series can be an event, and similarly, event-driven series can start at a certain time. The end of a series can be specified by time (by giving tf), by event, manually (with the finish method, which makes tf equal to the current time), or in terms of event count (by giving count). Events are internal (stored on the platform) or external SmartData, arithmetic, and logical operators. The SmartData Series stored and processed by the platform has the following JSON representation:

SmartData 4.2 – SmartData Series

```
"Series" : Object {
    "version" : unsigned char
    "unit" : unsigned long
    "x" : long
    "y" : long
    "z" : long
    "r" : unsigned long
    "t0" : unsigned long long
    "tf" : unsigned long long
    "type" : char[3]
    "period" : unsigned long
    "count" : unsigned long
    "event" : string
    "accuracy" : unsigned long
    "workflow" : unsigned long
}
```

- **version**: the version of the SmartData in the series (a series does not contain mixed versions SmartData);
- **unit**: the type of the SmartData in the series (see the SmartData documentation and typical units);
- **x, y, z**: the absolute coordinates of the center of the sphere containing the data points in the series (from a SmartData Interest);
- **r**: the radius of the sphere containing the data points in the series (initially from a SmartData Interest; is automatically adjusted with data point insertion);
- **t0**: (optional) a timestamp representing the time in which the series begins, in UNIX epoch microseconds;

- **tf**: (optional) a timestamp representing the time in which the series ends, in UNIX epoch microseconds;
- **type**: (optional) 'TTH' specifies high-frequency data (KHz sampling) with a fixed sampling rate. Some storage optimizations are applied.
- **period**: (optional) only defined for time-triggered series representing the period of data points (usually from a SmartData Interest, but also method create);
- **count**: (optional) specifies the number of data points to be captured before closing the series (tf is captured when counting data points are collected);
- **event**: (optional) a SmartData expression designating an event that marks the beginning of the series (tf is derived from the time the expression becomes/became true, representing the occurrence of "event");
- **workflow**: (optional) specify server-side algorithms to be applied on the series:

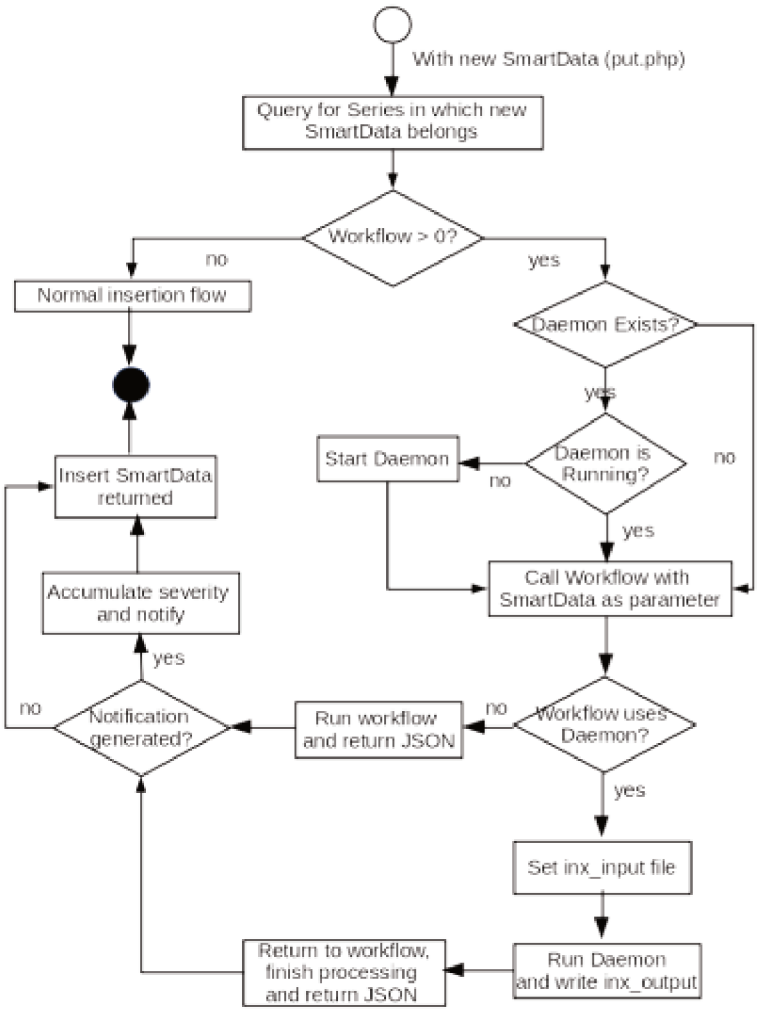
  input workflows: are executed during insert operations.

  output workflows: are executed with query operations.

The SmartData can be directed to a specific Workflow for data processing before its definitive insertion into the platform or its manipulation afterward. Workflows are used by the IoT platform to execute server-side algorithms related to the received series, which can be defined as input or output. The SmartData concept adds metadata to the read data to make it semantically complete, with a spatial location and reliability. In this way, it is possible to identify different devices in different locations, in this case, vehicles, without adjusting the settings of each one. Also, being a common data format, there is no need to adapt for each ECU so that the same algorithms can be used in all cars.

An input workflow can be specified during the creation of the series, and its execution occurs during insertions (PUT method) of SmartData in this Series, being applied to each SmartData individually and persisting through daemons. It can be used to pre-process data, execute algorithms, fix data points, generate notifications, and interact with other series. Daemons are sub-processes that receive data from the workflow, do the processing, and return it to the workflow or insert the processed data into a new series, preserving the original data. Figure 4.3 illustrates the flow with a diagram.

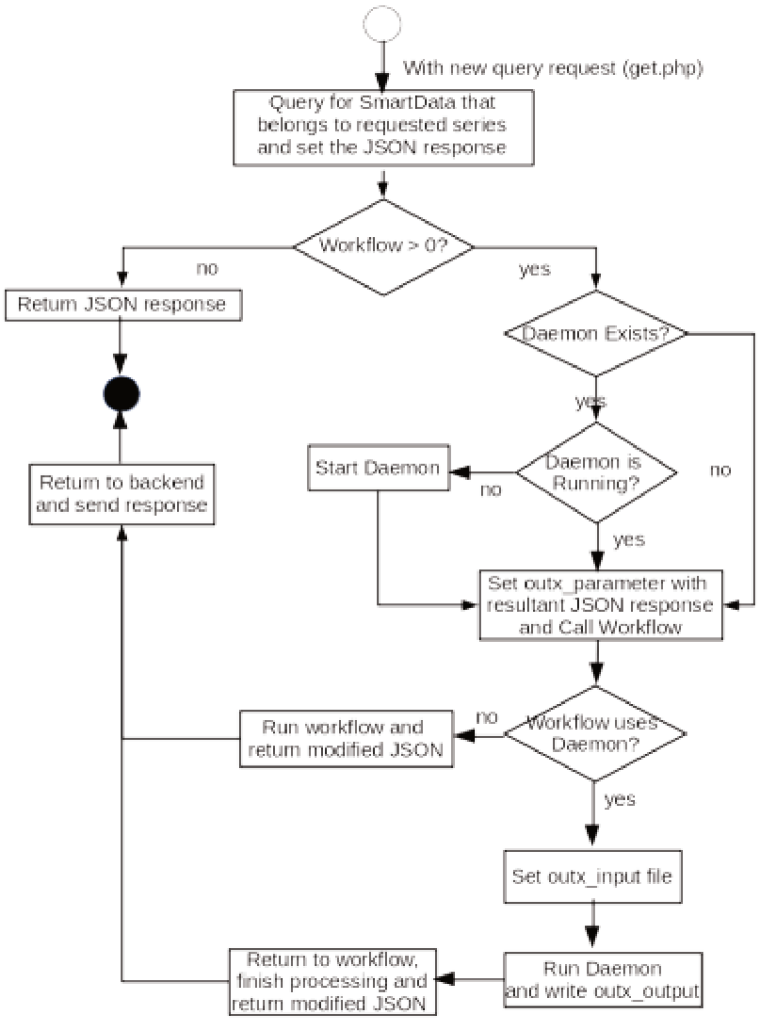
Figure 4.3 – Input Workflow Diagram.



Source: LISHA (2023).

An output workflow can be specified during a query request (GET method), and its execution is applied at the end of a query process to consider all returned SmartData records. Figure 4.4 illustrates the flow with a diagram; it can be used to post-process the data, to perform aggregations or transformations.

Figure 4.4 – Output Workflow Diagram.



Source: LISHA (2023).

Workflows are used because they can be executed in real-time, applying corrections to measurements and running machine learning algorithms for misfire detection, for example, notifying at runtime if faults are detected and providing real-time data visualization.

4.3 DATA ACQUISITION AND EXPERIMENTS

Over a two-year period, more than fifty experiments were conducted to build the datasets. Each type of analysis presented in this work had its peculiarities that required a different focus, but in general, one of the main objectives of creating the datasets was to obtain a large number of different variables. This allowed an FS process to identify the variables that were most relevant to the analysis in question (misfire detection or fuel consumption analysis). By incorporating a wide range of variables, the construction of

the dataset facilitated the identification of key relationships for these analyses, enabling the training and testing of various ML models. For this, a data acquisition methodology was followed and used for both analyses.

To start the data acquisition process, the hardware embedded in the vehicle must be loaded with the essay file. Which indicates the location of the signal in the ECU, including the definition of Smartdata (Unit, Dev, X, Y, Z, Signature, Type, Period, Workflow) and the minimum and maximum threshold values from the signals (pre-set to check if the measurement is as expected). The data from the essay are sent to the server. The data acquisition used in this work was time-triggered. During this phase, data is buffered and prepared for serial upload via 4G to the IoT platform using the SmartData format.

With the device embedded in the vehicle, numerous essays can be loaded onto it. This allows a wide range of tests with different objectives to be performed. Each of these essays can take into account different approaches for the analysis, which is excellent for comparing and evaluating the developed algorithms.

In the first experiments carried out, while the project hardware was being developed, most of the ECU data acquisition was done using the Integrated Calibration and Application Tool (INCA) from ETAS, which is used by Renault. This software allowed misfire, for example, to be forced into the engine by decalibrating system variables related to it. Subsequently, the tool created by IASE was used throughout the process, loading the essays in the vehicles, searching for the selected variables, and making the link between the ECU data and the server in the cloud. At that moment, the understanding of the misfire and other analyzes were already bigger, and with that, we were able to force the car through driving to generate failures or, in a way, manipulate fuel consumption. Remembering that the occurrence of misfires in the experiments ranged from none to more than one hundred, because of its unpredictability, it is impossible to control when the failure occurs and in what amount; we can only make the test environment more conducive for it to occur.

Unlike most of the related works, which carry out simulations of failures on test benches, the measurements in the project occurred at the car in operation, driven by project members, on the limited test track within the Ágora Industrial Park in Joinville, Brazil, where the Federal University of Santa Catarina is located. Next, it is discussed in detail how the acquisition and selection of features for each analysis were made.

4.4 FEATURE SELECTION AND ESSAYS

Considering the limited number of variables per essay, FS is important to reduce the set of variables that represents the phenomenon under analysis [Francis et al. 2022, Silva, Gracioli e Araujo 2022]. Also, it avoids overfeeding of ML models, removes harmful, redundant, or noisy variables, generates better generalization, reduces the

computational cost, and improves the performance [Zebari et al. 2020, Kiktova et al. 2014].

Strategies were chosen to create subsets of the available attributes, evaluating individual variables according to predefined criteria or directly sub-setting them. Then, a statistical evaluation was made to verify which of them was more qualified, this process usually has a stop condition, and the best result is the one that prevails until further evaluation [Kumar e Minz 2014]. As a result, among all subsets or variables tested, some outperformed the full set and were used instead.

The selection made for each analysis is treated separately below, but in general, the FS methods applied were:

- **SelectPercentile**: selects resources with a higher scoring percentage specified as parameters, 10%, 15% and 20% were used;
- **SelectKBest**: selects the features with the highest score according to the function chosen as a parameter. The Pearson correlation coefficient was used to find the best attributes of a dataset;
- **Sequential Feature Selector**: is a greedy procedure that finds the best feature to add to the subset of features that starts empty. First, finds the feature that maximizes a cross-validation score when an estimator is trained with a single feature. Once the first feature is selected, the procedure is repeated, adding a new feature to the subset and stopping when the desired number of selected features is reached;
- **Recursive Feature Elimination**: Select features recursively considering smaller and smaller sets. Initially, it is trained on the complete set where the importance of each feature is obtained concerning a specific attribute. Subsequently, the less important features are removed from the set, a procedure that is repeated until the desired number of features to be selected is reached. A version of this method in a cross-validation loop to find the optimal number of features was also used, called **Recursive Feature Elimination with Cross-Validation**.
- **SelectFromModel**: is a meta transformer used with any estimator that assigns importance to each resource about a specific attribute. The random forest regressor was used as an estimator with a square error criterion. Resources are removed if their importance is below the given threshold parameter. Also, the process ends when a desired number of features to be selected is reached.

4.4.1 Misfire Data Collection

To obtain different data sets with and without failure for training the algorithms, the experiments in the vehicle took place on different days and more than once a day,

with 1 to 3 experiments of 8 to 15 minutes being generated. The practices were limited to that time due to overload in the engine; the test car showed signs of degradation when being forced for longer periods, precisely because the failure can harm the engine and its functioning, as seen before (2).

Several experiments were conducted to understand the misfire in different scenarios (engine in fault and normal conditions, car in stationary and moving state, high and low gear, and different speeds). The experiments were performed with variables that describe basic car components and systems, such as engine status and others related to misfire, initially taken from the bibliography of causes (2.3.1) and related works (3). Numerous machine-learning algorithms were applied to analyze each set of variables, such as those described in the background (2.5). The algorithms training was done with the data collected in the experiments; for the supervised training, the misfire counter variable of the ECU was used; for the other cases, it was disregarded. Varying according to the algorithm, the training parameters were defined during its execution, observing metrics such as cross-entropy and squared error loss, among others, and observing the convergence of the loss. A general explanation of the parameters of the algorithms that obtained good performances and were considered for the results of the work are presented in subsection 4.5

Subsequently, to confirm the understanding of the causes of the failure and to improve the performance of the misfire detection algorithms, the FS algorithms developed were applied in the construction of new data sets. Having more than twenty thousand variables available in the ECU, it is necessary to analyze their importance and choose the essential ones for misfire detection. It is noteworthy that the fault counter generated by the ECU was used as a reference parameter to confirm whether the data read was a fault or not and also used to label the data so that the FS algorithms could be trained with the variables that symbolize failure over time and also for supervised machine learning training. Thus, after selecting the variables and forcing the car to fail, it was possible to obtain large data sets to analyze the failure identification algorithms.

Among the most selected features, which were used in fault detection, the following systems and their respective variables stand out (4.1):

4.4.2 Fuel Consumption Data Collection

We applied FS for driver profile classification and fuel regression in this analysis. In the classification context, the target variable was the consumption level label, and the predictors used for evaluation were the Logistic Regression and XGBoost Classifier models. For the regression problem, the target variable was the actual fuel consumption, and the evaluated models were Ridge, SVR, and XGBoost regressor. Table 4.3 shows the number of features selected at each Stage.

For this selection, the variables of the engine subsystems related to fuel con-

Table 4.1 – Misfire FS

Engine systems	**Variables**
General	Engine coolant temperature, engine speed, vehicle speed, current gear engaged, dead center counter;
Air Control	Engine air load, atmospheric pressure, intake manifold pressure, intake air temperature, throttle valve position, intake manifold temperature;
Fuel And Richness Control	Final alcohol adaptive, injection time - cylinder 1 - 1st injection, injection time - cylinder 2 - 1st injection, injection time - cylinder 3 - 1st injection, injection time - cylinder 4 - 1st injection, air/Fuel Ratio, richness regulation status, richness adaptation factor, catalyst Diagnosis Criteria (OSC Level), catalyst exhaust gas upstream oxygen sensor voltage, catalyst exhaust gas downstream oxygen sensor voltage
Ignition Advance	Intake camshaft phaser position, ignition advance
Engine Torque	Engine torque without gearbox request, estimated effective engine torque, effective engine torque target requested by real (pedal) and virtual (ACC/CC/SL) drivers, engine torque losses, final indicated torque raw, final indicated torque target.
Alternator and Battery Management	Alternator load, battery voltage, alternator power, battery state of charge, filtered alternator rotor current.

Source: Author (2023).

sumption were initially separated. Subsequently, the algorithms SelectPercentile, SelectKBest, SequentialFeatureSelector, SelectFromModel, RFE, and RFECV, available in the python library Scikit-Learn [Pedregosa et al. 2011], was used to separate the variables with a greater relationship with the problem, being selected a total of 46 unique variables (listed in Table 4.2), joining the result of each algorithm applied in all Stages. These algorithms are designed to operate with supervised learning methods, ranking and selecting features by statistical measures between them and the target, as in the case of SelectPercentile and SelectKBest, or evaluating the predictor's performances using different combinations of features, such as SequentialFeatureSelector, SelectFromModel, RFE, and RFECV.

The amount of obtained data varies according to the variables selected in the essay (in the GUI) and the time spent driving the vehicle. All data acquired while driving the vehicle is manipulated in real-time by the workflows, resulting in feedback to the stakeholders at runtime or is available for further offline analyses and reports (through the database).

The essays varied in set and quantity of variables, intending to explore several subsets of the engine to make an adequate and comprehensive selection of features. The tests took place in different environmental conditions, running from September 2022 to March 2023, taking periods of heavy rain, high and low temperatures, and dry and wet weather. With this differentiation, we expected to observe changes in the profile of consumption variables, even in the same route. The limiting variation is precisely the route because our test vehicle is not licensed to run on public roads. In this way, all tests occurred within the Ágora Industrial Park in Joinville.

Table 4.2 – Fuel consumption - FS

Engine Subsystem	**Variable Description**
Basic Component	Filtered engine speed
	Engine coolant temperature
	Engine speed
	Vehicle speed
	Total vehicle distance
	Accelerator pedal position
	Brake pedal - switches consolidation state
	Acceleration or deceleration vehicle state
	Relative throttle position in percent of the sensor supply voltage
	Value of applied offset correction
	Clutch pedal - minimum travel switch state - wire
	Instantaneous engine speed (tooth to tooth engine speed)
Air Control	Ambient air temperature
	Engine air load 1, 2, 3, and 4 TDC earlier
	Engine air load using predicted pressure
	Setpoint of the mass air flow pumped into cylinder
	Intake air temperature
	Throttle valve position setpoint
	Throttle valve position
	Intake manifold pressure
	Mass of air trapped in the cylinder for next ignition
Fuel And Richness Control	Final alcohol adaptive
	Injected fuel mass for ADAC [fuel consumption]
	Gas consumption for the mux
	Sum of fuel mass injected on the cylinder
	Injection time - cylinder 1, 2, 3, and 4 - 1st injection
	Catalyst warm-up is confirmed as necessary
	C factor
	Injected fuel mass setpoint disregarding cylinder
	Average injected fuel mass disregarding cylinder number or injection phase
	Effective time injection
	Offset applied on richness closed loop
	Richness close loop injection time factor applied
Ignition Advance	Maximum ignition advance
Engine Torque	Torque efficiency corresponding to minimum static ignition advance
	Estimated indicate engine torque
	Effective engine torque requested by real and virtual drivers
	Effective engine torque setpoint requested by real and virtual drivers
	Effective engine torque setpoint requested by real and virtual drivers
	Estimated effective engine torque
	Final torque target after modulation action
	Torque efficiency for idle speed controller
	Torque correction calculated
	Final torque setpoint after modulation action
Alternator and Battery Management	Battery voltage
	Alternator power

Source: Author (2023).

The first experiments were conducted in September, October, and November 2022 and had 30 acquisition variables, initially taken from the related works, summarized in Tables 3.2 and 3.3, for comparison. We refer to these datasets as "Stage 1". The main objective of the Stage 1 data sets was to understand the relationship between

Table 4.3 – Description of the acquisition datasets.

Stage	**Total / Selected features**	**Duration of experiments (min)**	**Driver's behavior**
Stage 1	30 / 13	13 - 27	Varied between aggressive and normal driving in the final half of the experiment
Stage 2	152 / 31	18 - 40	Varied between aggressive and normal driving throughout the experiment
Stage 3	141 / 28	10 - 15	Ordinary driver steering, not forced

Source: Author (2023).

the variables with fuel consumption. To do so, we forced a division in the experiments to make it evident in the data, where a part drove the car normally and then more aggressively, thus with higher fuel consumption. In later experiments, in January and February 2023, we monitored more variables to apply FS and improve training inputs. The experiments were done by forcing the vehicle randomly during the route; this alternation was not so explicit in the data. These datasets had more than 150 variables measured, and we refer to them as "Stage 2". Finally, we ran more experiments in February 2023 with a dataset that grouped the selected variables by the FS algorithms applied to the previous stages. In this phase, the vehicle was driven normally, as if it was an ordinary driver, so that we could validate the created ML models. We refer to it as "Stage 3".

Table 4.3 compiles information about each stage. The focus of the analysis was short-term driving, representing the day-to-day of a common driver without traffic, with an execution time of up to 40 minutes.

4.5 ALGORITHM DEVELOPMENT AND PARAMETERS

In this section, the general choice of algorithms that were selected for the evaluation is explained, and that will be presented in the results section (5), along with an example of the parameters used. A more detailed explanation of each tested algorithm can be found in the background section (2.5)

During the project, several machine learning algorithms with a diverse range of algorithm types, in an attempt to explore their effectiveness in solving fuel consumption and misfire detection problems, were implemented and evaluated. The tested algorithms included decision trees, different types of clustering, classifiers, regressors, and various types of neural networks (perceptron, MLP, autoencoder, NARX, etc). The de-

cision to test all these algorithms was based on their prominence in the field, where we reviewed the models present in the related works and literature (3), as well as understanding their suitability to the nature of the dataset and the design goals.

Throughout the test phase, the performance of each algorithm, based on criteria such as accuracy, precision, recall, f1-score, and R^2, was monitored. It became evident that certain algorithms outperformed others regarding predictive accuracy and generalizability. Consequently, most of the algorithms were eliminated from the analysis because they could not perform or had results far below the others.

Overall, the classification algorithms proved to be better for detecting misfires and classifying the driving profile. The XGBoost algorithm, in particular, excelled in terms of performance, serving very well for both analyses, including its adaptation to a regressor, which also had the best performance in the task of instantaneous prediction of fuel consumption.

For the implementations, we used Python version 3.8.10, the Sci-kit Learn library version 1.2.1 (for the models where it is mentioned), and the XGBoost version 1.7.0. All algorithms are available online at <https://github.com/canalrafael/Fuel-Consumption-Analysis.git>. The key parameters utilized in the algorithms are outlined in the table below (Table 4.4).

Table 4.4 – Algorithms parameterization.

Algorithm	**Parameter**	**Analysis**
Gradient Boosting	default	Misfire Detection
K-Means	default	
KNN	k=1	
Logistic Regression	max_iter=100	
SVC	default	
XGBoost	default xgboost.XGBClassifier()	
XGBoost	default xgboost.XGBClassifier()	Driver Profile Classification
Logistic Regression	max_iter=100	
K-Means	K=2	
XGBoost	default xgboost.XGBRegressor()	Fuel Consumption Prediction
Ridge Regression	default	
SVR	LinearSVR() with epsilon=1.0, and max_iter=10000	

Source: Author (2023).

4.5.1 Workflow example

The following code snippets exemplify parts of the workflow code. Initially, the workflow (Alg. 4.3) starts and prepared the structure to read and treat the SmartData, which validates the reading made by the server to be applied to machine learning models. Next, the workflow daemon (Alg. 4.4) receives the data points and starts its routine

of consuming the data, predicting the data point and sending the prediction (to a file or to the server). Each of these routines executes its part of the code asynchronously.

Algorithm 4.3 – Workflow.

```
1   if __name__ == "__main__":
2       # ++++++++++++++++++ DO NOT CHANGE THIS LINE
         ++++++++++++++++++
3       try:
4           smartdata = datamodel.read_smartdata(sys.argv[1],
             PATH_DIR + "/" + FILE_NAME)
5           # ++++++++++++++++++ DO NOT CHANGE THIS LINE
             ++++++++++++++++++
6           #try:
7           df = datamodel.smartdata_to_df(smartdata)
8           df = datamodel.add_label_col(df, labels_map)
9           df = datamodel.add_uuid_col(df)
10          handle_label_not_found(df)
11
12          for uuid in df["uuid"].dropna().unique():
13              content_df = df[df["uuid"] == uuid]
14              first_row = content_df.iloc[[content_df["t"].
                 idxmin()]].squeeze()
15              t = first_row["t"]
16              entry = f"{QUEUE_DIR}/{t}_{uuid}"
17              contents = content_df.to_json(None, orient="
                 records", lines=True)
18              try:
19                  queue.add_entry(entry, contents)
20              except Exception as err:
21                  unit = first_row["unit"]
22                  dev = first_row["dev"]
23                  x = first_row["x"]
24                  y = first_row["y"]
25                  z = first_row["z"]
26                  write_log(
27                      f"(Unit,Dev) {str(err)} for: ({unit},{dev
                         }) at {t} with uuid {uuid} at ({x}, {y},
                         {z}).\n"
28                  )
29      except Exception as err:
30          write_log(f"{err}\n")
31
```

```
32          # ++++++++++++++++++ DO NOT CHANGE THIS LINE
             ++++++++++++++++++
33          print(json.dumps(smartdata))  # Send smartdata back to
             PHP
34          # ++++++++++++++++++ DO NOT CHANGE THIS LINE
             ++++++++++++++++++
```

Algorithm 4.4 – Daemon.

```
1   async def main():
2   logging.debug("DAEMON STARTED")
3   done, pending = await asyncio.wait(
4       [
5           # Main functions for
6           asyncio.create_task(consume_data_queue_routine()),
7           asyncio.create_task(predict_df_row()),
8           asyncio.create_task(send_prediction_routine()),
9       ],
10      timeout=DAEMON_LIFETIME,
11  )
12  logging.debug("BEFORE GLOBAL_FLAGS TO TRUE")
13  GLOBAL_FLAGS["EXIT"] = True
14  logging.debug("AFTER GLOBAL_FLAGS TO TRUE")
15  try:
16      await asyncio.wait_for(asyncio.gather(*pending), 30)
17  except:
18      return
```

The results of the analysis performed by the workflow can be observed through a dashboard of a web tool such as Grafana, directly on the server or through local applications that communicate with the server (saving or reading data).

4.6 PARTIAL CONSIDERATIONS

This section summarizes the essential findings and contributions related to the communication structure, data acquisition methodology, dataset construction, feature selection, data validation, developed algorithms, and parameterization aspects discussed in the chapter.

This study's extensive experimentation and analysis thoroughly tested and validated the communication between the developed hardware and ECU and server. This validation ensured the integrity and availability of data, allowing for real-time analysis in the cloud.

The ability to monitor misfire classifications, driving profile analyses, and fuel consumption predictions in real time has provided stakeholders with valuable insights. The FS process was crucial in obtaining meaningful results by identifying variables strongly related to the analyzed aspects. This process demonstrated that despite different focuses, common variables significantly influence the engine's overall functioning and subsystems.

The attribute selection and system integrity validation lend credibility to the applicability of this work in the industry, particularly for Renault. The outcomes highlight the possibility of performing intelligent analyses comparable to or even outperforming the ECU's analyses, leveraging commonly available vehicle data without needing costly production sensors such as vibration or sound sensors. Furthermore, the speed of analysis and real-time monitoring facilitate prompt adjustments by engineers, expediting the engine calibration process and addressing concerns such as gas emissions and fuel consumption.

The developed algorithms provided valuable insights into the data nature and specific analysis objectives, highlighting their compatibility with supervised machine learning algorithms. However, in the case of misfire analysis, the data imbalance posed challenges for anomaly-detection neural network algorithms, resulting in accuracy levels below 60%. As a result, these algorithms were excluded from the subsequent chapter (5). All algorithms can be applied in workflows; for that, it is necessary that the models are loaded in the prediction routine and that the data entry (input) is adjusted for it.

In the following chapter, the evaluation of the developed models is treated with the explanation of the main results compared to the analyzed literature. The values obtained by the classification algorithms make their applicability to the project objectives clear, not requiring complex algorithms that are more difficult to understand and explain. The regression algorithms also showed relevant results, presenting a prediction of instantaneous consumption with precision for real-time use. Thus, temporal algorithms were not used; the real-time analysis was performed without the need for monitoring over time to bring consistent results.

5 EVALUATION

In the following subsections, we present the results obtained in the misfire detection and fuel consumption analyses. Additionally, we make comparisons with related works that are close to ours (described in Section 3).

5.1 MISFIRE CLASSIFICATION RESULTS

To evaluate the model, all the experiment's data were gathered into a single set and balanced it. This is due to the fact that the number of samples with misfires is much smaller than the number of healthy samples since the occurrence of a misfire cannot be easily controlled, and it occurs sporadically.

Subsequently, we divided the data into training and testing sets, shuffling the data and dividing them into 70% for training and 30% for testing, then applied the ML models. This technique allows us to verify the generalization of the models and compare the different results of the trained models. To check the overall performance of the models, we calculated their precision, recall, and F1-score on the test set so that we have metrics to compare with related works as [Devasenapati, Sugumaran e Ramachandran 2010, Firmino et al. 2021].

Figure 5.1 illustrates the best results obtained with an FS method for each model trained on the same dataset. The Y axis, which contains the percentage (%) of hits, starts at 80%, given the difference between the results, and the X axis contains the ML model name. The blue columns are the precision of the ML model, the orange is the recall, and the green is the F1-score.

Figure 5.2 illustrates the ROC curve's plot. The higher the curve, i.e., the greater the area under the curve, the better the model can do a binary classification in terms of having a high true positive rate against a low false positive rate. In this case, XGBClassifier is clearly the better model.

In general, the XGBoost Classifier proved to be the best model for identifying misfires, not only in the best-case scenario as illustrated in Figure 5.1, but also in the average case of different features sets obtained through the FS methods, as demonstrated in Table 5.1, presenting the average precision, recall, and F1-score obtained by the models. The algorithm had a precision of up to 92.40%, a recall of 96.16%, and an F1-score of 94.24%. On average, it achieves a precision of 87.55%, recall of 92.25%, and an F1-score of 89.79%, with a standard deviation of 7.38%, 4.30%, and 5.90%, respectively. Therefore, XGBoost is the algorithm that, on average, achieved the highest F1 score, which is the metric chosen to rank the algorithms. However, it is worth noting that the standard deviation for XGBoost is larger than that of other algorithms, such as Gradient Boosting, indicating that it is less consistent compared to these other algorithms.

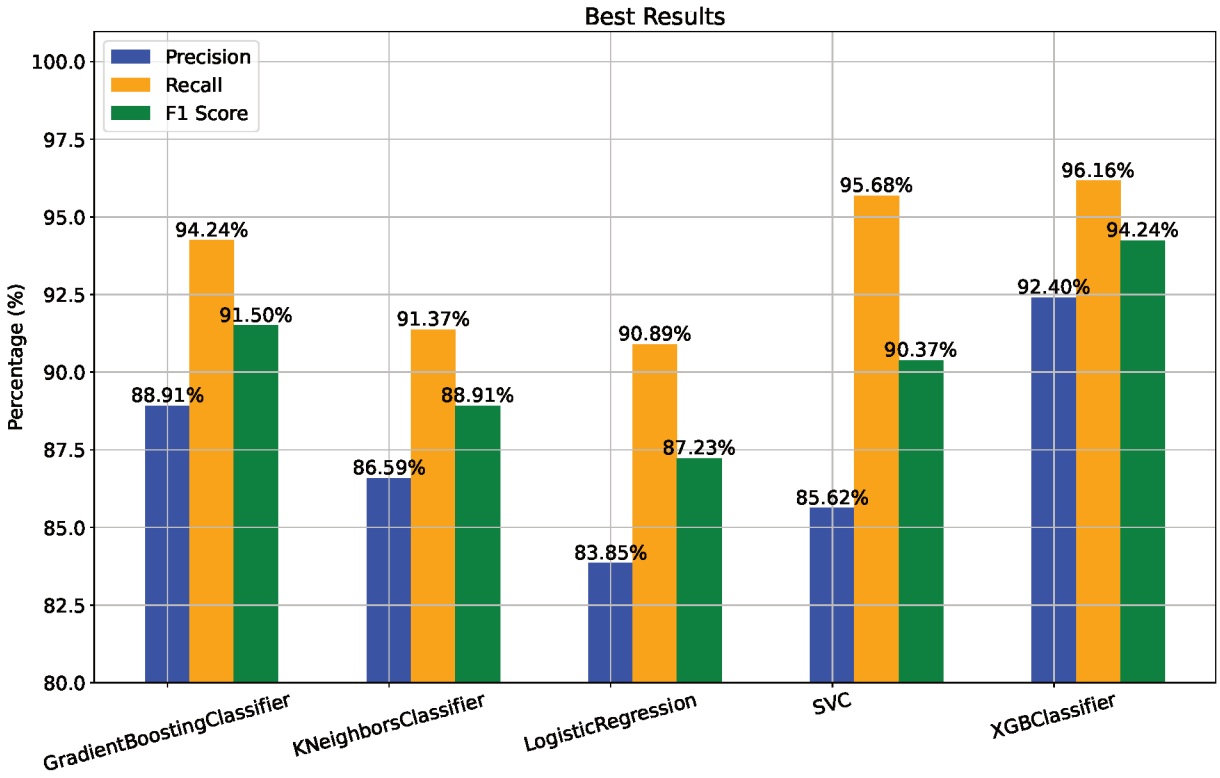


Figure 5.1 – Best result for each trained model using F1 score (green) as parameter.

Another way of illustrating the results is shown in Figure 5.2. A ROC curve is a graphical representation of the performance of a binary classification model, illustrating the trade-off between the true positive rate (TPR) and false positive rate (FPR) for different classifications. The higher the curve, indicating a greater area under the curve (AUC), the better the model's ability to discriminate between the two classes. A higher AUC suggests the model has a higher TPR while maintaining a lower FPR. In this case, in addition to the information provided by Table 5.1, it is possible to observe that the AUC also indicates the XGBClassifier model as the best.

For the Gradient Boosting algorithm, Table 5.1 shows the precision ranges from 72.51% to 88.91%, the recall ranges from 87.29% to 95.68%, and the F1-score ranges from 79.22% to 91.50%. The average precision is 84.48%, recall is 92.73%, and F1-score is 88.37%. The standard deviations for precision, recall, and F1 score are 6.05%, 3.05%, and 4.62%, respectively. As can be observed by the lower deviations, it has consistently good results, despite not reaching as high scores as XGBoost.

The K-Neighbors algorithm exhibits a precision range of 85.62%-86.59%, a recall range of 91.37%-91.85%, and an F1-score range of 88.40%-88.91%. The average precision is 86.03%, recall is 91.53%, and F1 score is 88.69%. The algorithm demonstrates low standard deviations, with 0.50% for precision, 0.28% for recall, and 0.26% for the F1-score. This algorithm has a good average precision, recall, and f1-score

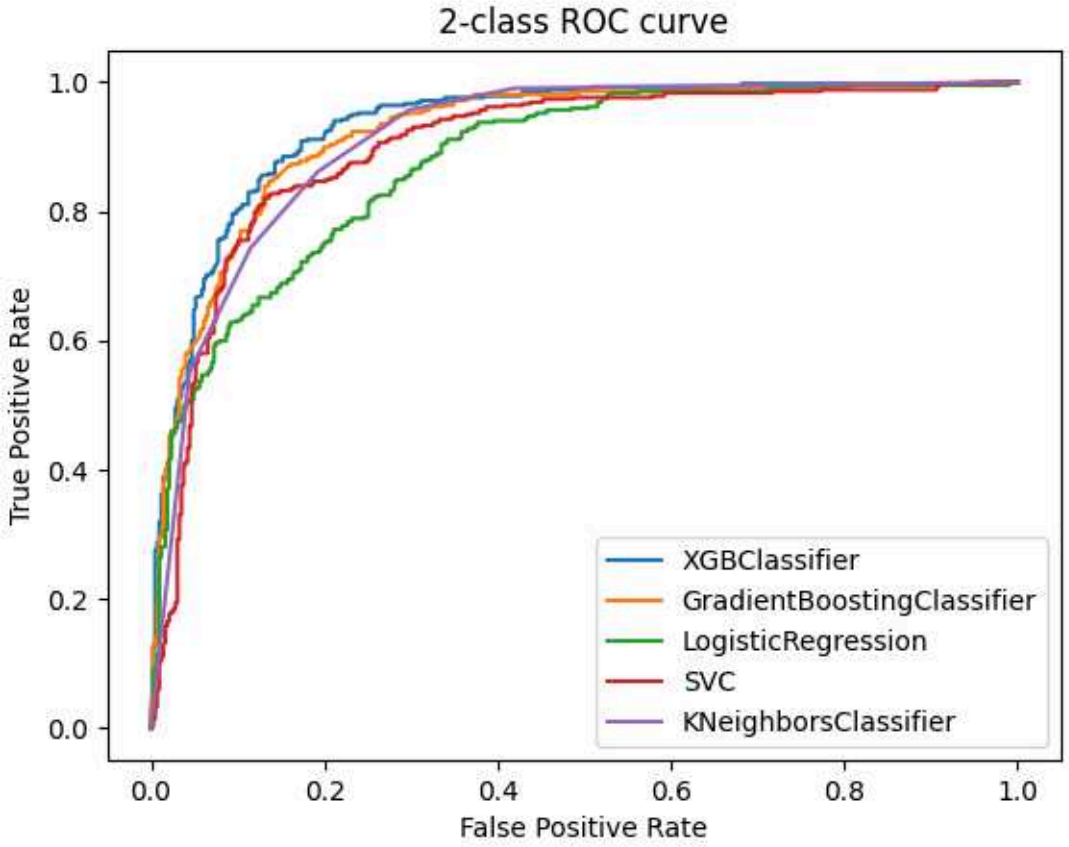


Figure 5.2 – ROC Curve.

results and the most consistent results because it has a smaller standard deviation.

Continuing the analysis of Table 5.1, Logistic Regression demonstrates precision of up to 83.85%, recall of 92.33%, and an F1 score of 87.23%. On average, it achieves a precision of 79.95%, recall of 84.89%, and an F1 score of 82.21%, with a standard deviation of 3.06%, 6.79%, and 3.59%, respectively. SVC presents a precision of up to 85.62%, a recall of 95.68%, and an F1 score of 90.37%. The average precision is 82.17%, recall is 93.05%, and F1 score is 87.27%, with standard deviations of 3.06%, 2.29%, and 2.71%, respectively.

Finally, the K-Means clustering has inferior results than the other models as it is possible to see in Table 5.1: the algorithm demonstrates precision ranging from 6.90% to 35.95%, recall ranging from 6.06% to 60.00%, and an F1 score ranging from 9.97% to 35.95%. On average, it achieves a precision of 23.64%, a recall of 34.00%, and an F1 score of 19.43%. The standard deviations for precision, recall, and the F1 score are 15.02%, 27.02%, and 14.36%, respectively. Because of this discrepancy, it is not represented in Figure 5.1.

While accuracy is commonly used in various studies such as [Shahid, Ko e Kwon 2022, Qin et al. 2021, Jafarian et al. 2018, Firmino et al. 2021], it is not the most suitable

Table 5.1 – Results evaluation metrics

Algorithm	Information	Precision	Recall	F1 Score
XGBoost	Range	73.03%-92.40%	84.41%-96.16%	78.31%-94.24%
	Average	87.55%	92.25%	89.79%
	Std dev.	7.38%	4.30%	5.90%
Gradient Boosting	Range	72.51%-88.91%	87.29%-95.68%	79.22%-91.50%
	Average	84.48%	92.73%	88.37%
	Std dev.	6.05%	3.05%	4.62%
K-Means	Range	6.90%-35.95%	6.06%-60.00%	9.97%-35.95%
	Average	23.64%	34.00%	19.43%
	Std dev.	15.02%	27.02%	14.36%
KNN	Range	85.62%-86.59%	91.37%-91.85%	88.40%-88.91%
	Average	86.03%	91.53%	88.69%
	Std dev.	0.50%	0.28%	0.26%
Logistic Regression	Range	76.60%-83.85%	76.02%-92.33%	77.04%-87.23%
	Average	79.95%	84.89%	82.21%
	Std dev.	3.06%	6.79%	3.59%
SVC	Range	79.79%-85.62%	91.61%-95.68%	85.40%-90.37%
	Average	82.17%	93.05%	87.27%
	Std dev.	3.06%	2.29%	2.71%

Source: Author (2023).

metric for classification tasks similar to ours. So, we adopted precision, recall, and F1-score as they provide a more comprehensive assessment in scenarios where the class distribution is imbalanced, or the cost of misclassification varies across classes, also complemented with the ROC curve. By employing these metrics, we were able to capture the nuanced performance of our classification model more accurately.

Compared to related works like [Qin et al. 2021] and [Jafarian et al. 2018], our work stands out due to its innovative approach of performing real-time analysis using data directly from the Electronic Control Unit (ECU). This methodology can be applied to other vehicles with the same variables. Moreover, even getting similar results, our work utilizes simpler and more easily understandable algorithms [Dreiseitl e Ohno-Machado 2002], which are executed in the cloud.

5.2 FUEL CONSUMPTION

This Section presents the results of classifying the driver's driving profile and predicting instantaneous fuel consumption separately and in detail.

5.2.1 Driving Profile Classification Results

To evaluate the classifiers, we performed cross-validation, partitioning the datasets into training and testing sets. The chosen partitioning method took one experiment of a "Stage" as a test set and the others for training, similar to [Ping et al. 2019]. This technique allows us to verify the generalization of the models, in addition to applying in results the metrics of accuracy, precision, and recall, as used and described by [Peppes et al. 2021]. In this way, we obtained the overall performance of the models and how many correct classifications were made, visualizing the number of false positives and negatives.

Figure 5.3 shows a graph of a test experiment with the fuel consumption variable (blue) highlighted by the High Consumption class (red) made by XGBoost. The points not marked with color represent the economic consumption class. In this example, the model identifies whether the consumption level was high or low compared to the vehicle's original consumption variable.

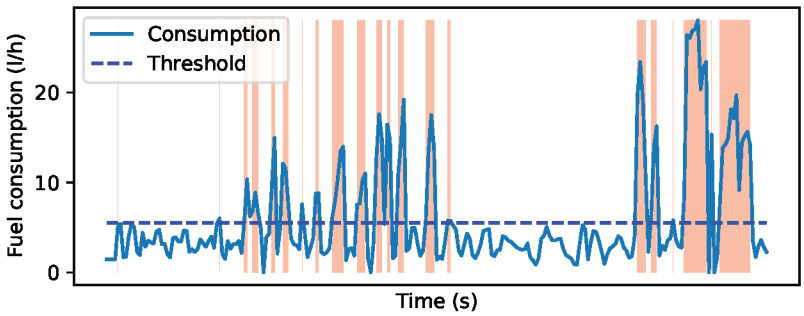


Figure 5.3 – Fuel consumption (in blue) over time, with the classification of high consumption periods (in red).

As explained in Section 4.3, we divided the experiments into Stages, in which we applied changes in the selected variables due to the knowledge acquired (i.e., domain-specific) and the FS process. Table 5.2 summarizes the results progressively (stage after stage), emphasizing the benefits of a better understanding of the problem and the implications of FS algorithms.

Table 5.2 also shows the training and testing results obtained by the explored classification and clustering methods, through the different experimental stages. The shown metrics do not bring all detailed evaluations, but a summary of gathered results from different feature sets, as well as experiments train and test divisions, showing the range, average, and standard deviation of the obtained scores for different scenarios.

When comparing the experiment stages, Stage 1 shows lower performances, ranging from 80% to 95% of accuracy, while the second and third stages show better results, varying from 81% up to 99% of accuracy. Furthermore, the precision of the latest experiments is far higher, reaching up to 100% scoring, yet a smaller recall is

Table 5.2 – Classification results evaluation metrics.

	Algo.	Info.	Accuracy	Precision	Recall
Stage 1	XGBoost	Range	80.60%-94.80%	76.30%-97.40%	72.30%-99.40%
		Average	88.09%	85.43%	89.98%
		Std dev.	5.47%	6.26%	10.22%
	Logistic Regre.	Range	20.39%-92.72%	29.38%-100.00%	0.61%-100.00%
		Average	75.63%	78.86%	77.34%
		Std dev.	22.63%	20.23%	28.23%
	K-Means	Range	26.29%-79.31%	29.27%-100.00%	46.60%-53.40%
		Average	66.63%	75.30%	52.13%
		Std dev.	18.16%	21.77%	2.48%
Stage 2	XGBoost	Range	81.21%-99.59%	92.82%-100.00%	40.54%-98.53%
		Average	95.42%	96.86%	88.07%
		Std dev.	6.41%	2.77%	21.27%
	Logistic Regre.	Range	70.64%-97.93%	53.52%-100.00%	35.14%-95.68%
		Average	88.94%	86.17%	78.44%
		Std dev.	9.33%	15.71%	22.71%
	K-Means	Range	6.45%-93.55%	7.94%-98.07%	19.76%-91.70%
		Average	60.41%	52.32%	61.92%
		Std dev.	31.36%	30.63%	24.53%
Stage 3	XGBoost	Range	95.00%-100.00%	93.33%-100.00%	93.33%-100.00%
		Average	97.30%	96.82%	96.11%
		Std dev.	1.00%	3.18%	3.29%
	Logistic Regre.	Range	85.00%-100.00%	76.47%-100.00%	80.00%-100.00%
		Average	92.67%	90.92%	90.56%
		Std dev.	10.13%	14.11%	11.85%
	K-Means	Range	6.83%-90.68%	6.85%-100.00%	8.33%-75.00%
		Average	45.65%	45.42%	39.72%
		Std dev.	34.05%	39.93%	23.83%

Source: Author (2023).

obtained. This shows an improvement in the updated variable set for classifying levels of consumption, but still with good results with the older set. Analyzing the distributions and value range of consumption records for some experiments, there were cases where a lower variety in the data, which combined with the wrong selection of features, led to a weaker performance. Overall, the obtained results showed that the classification methods had good precision in determining the consumption level from the right selected features, reaching up to 100% of scoring for all metrics in some of the train and test splits.

Regarding the classifier methods, in general, XGBoost shows a slightly better scoring than the Logistic Regression. This changed in some stages and for certain

features selected sets, but overall, XGBoost had more persistent performances, ranging between high averages and low standard deviations. Besides, it had better prevalent results, as highlighted in green in Table 5.2, showing greater average scores, and lower variations, showing persistence in them. In contrast, Logistic Regression shows higher scores in some cases, but also has lower averages and higher standard deviations on the results, reaching accuracies and recalls under 30% for some specific tests and feature sets, which did not occur with XGBoost, although had fair results in general.

The K-Means clustering presented inferior results in comparison with the classification algorithms, trained specifically for the given classes. It got lower accuracy, precision, and recall averages in all steps, ranging from 40% to 75%. The reason can be found in visualizing the standard deviation in Table 5.2, with values greater than the ones obtained in the other algorithms. K-Means proved to be more sensitive to the set of variables used in training, having precision values below 10%, bringing the average down. Even with this issue of having wider variations with the wrong resource sets that negatively affect its performance, in general, K-means was capable of getting a reasonable scoring, showing accuracy, precision, and recall up to 90% in some tests. This indicates that the specific threshold used for comparison was not perfect or that it can be adjusted for the identified clusters, reinforcing that the driving styles clusters given by K-Means may also capture a division in consumption levels, not only a binary division. It is also notable that no specific FS was applied for K-Means, using for some tests the same features selected for the classification algorithms and the related works common features, which also have contributed to lower and less persistent results. Besides, in our work, we performed the comparison of the clusters with real data of the instantaneous consumption levels, in contrast to [Peppes et al. 2021, Zheng et al. 2022], where simulated data or no fuel consumption were observed.

Compared to related works, we improved the overall results, having metrics with superior values, an extensive FS work that accurately identifies the driver's relationship with key variables of fuel consumption, and ensuring the integrity of the data used. One of the observed advances, when compared to similar models and analysis [Ping et al. 2019], is obtaining similar or superior metrics using lower-cost methods. In the referred work, the authors got an accuracy of 80% to 83% with more complex methods such as Neural Networks, using visual information in addition to the vehicle-gathered data, while in this work, by using only vehicle-observed features, we reach 100% accuracy in some scenarios through computationally less expensive models as Logistic Regression [Dreiseitl e Ohno-Machado 2002] and Gradient Boosting [Taghavi e Shoaran 2019], besides they are not a black-box. This would allow the execution of classification within the ECU and not on a cloud server, for instance.

Furthermore, in this work, we predicted a consumption level with high accuracy and precision using directly a level of measured consumption as a basis. Similar works

obtained equivalent or lower accuracy and precision, from 80% to 100%, using labels generated by clusters for classification, as in [Peppes et al. 2021, Zheng et al. 2022], while in this work, we managed to obtain the same or better performance, up to 90% and 100%, using direct reference consumption for training and testing classifiers.

5.2.2 Fuel Consumption Prediction Results

To evaluate the predictive models for fuel consumption, various metrics can be used, such as mean squared error (MSE), mean absolute error, root mean squared error, and coefficient of determination (R^2), among others [Ziółkowski et al. 2021]. For allowing a better comparison with related works and, because they complement each other in the analyses, we evaluated the following ones in each set of algorithms: MSE, R^2 score, and MAE.

Mean Squared Error (MSE) is a metric that shows an average of the square of errors between predicted and measured values, using the squared term that gives higher penalties for higher error values. For each set of measured values $y_1, \ldots, y_n$ and predicted values $\hat{y_1}, \ldots, \hat{y_n}$, the MSE is determined by

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{5.1}$$

Mean Absolute Error (MAE) is a metric that measures the average of the errors using the absolute error value, which stands for a slightly moderate average, without distinction from higher or lower penalties. From some set $y_1, \ldots, y_n$ of measured values and $\hat{y_1}, \ldots, \hat{y_n}$ predicted values, the MAE is obtained by:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|. \tag{5.2}$$

R^2 is a metric that evaluates how much of the data variance is explained by the predicted values given by the model. It has a maximum value of 1, where the score closer to 1 represents a better fit. Given $y_1, \ldots, y_n$ real values, $\overline{y}$ their mean value and $\hat{y_1}, \ldots, \hat{y_n}$ predicted values, R^2 is obtained by the following equation

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2} \tag{5.3}$$

To assess the regression algorithms, we executed a similar analysis with the same cross-validation procedure as explained in the previous subsection. For the first two Stages, we separated in each round one of its experiments for testing and the remaining for training. For the third Stage, we performed 3-fold cross-validation, where data was randomly split into 3 sets. Where we performed cross-validation between them, selecting one for testing and the remaining for training.

To illustrate, Figure 5.4 shows the predicted fuel consumption levels estimated by XGBoost Regressor (in orange) compared to the actual measured consumption (in

blue). The graph exemplifies the results given by these regressor models, which provide predictions of the actual values of fuel consumption levels in each instant.

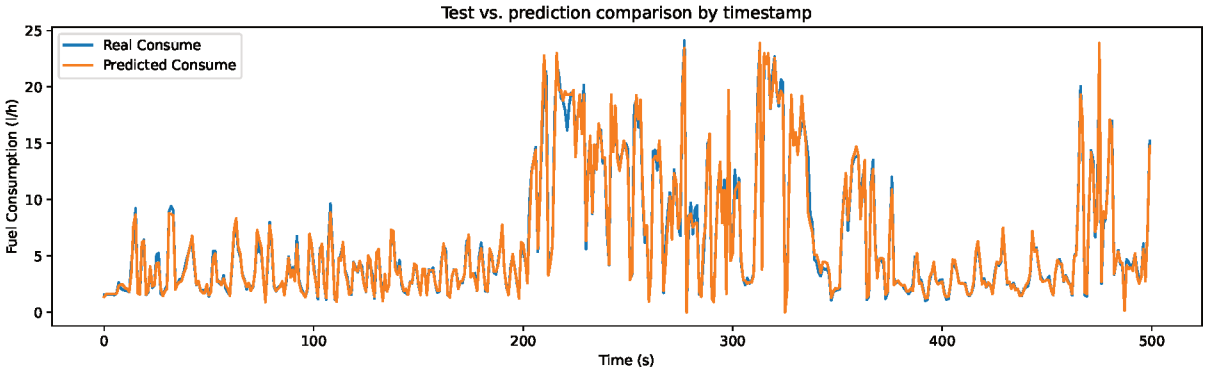


Figure 5.4 – Comparison between measured (blue) and predicted (orange) fuel consumption levels over an experiment, through XGBoost Regressor.

Similar to the classification evaluations, we summarize the results of regression models in Table 5.3, aggregating different train and test results by Stage of experiments and used algorithm.

Results in Table 5.3 show a broader range of obtained scores and metrics between models and Stages of experiments using different sets of experiments. Concerning the Stages, once more, we observe overall lower scores for the Stage 1 experiments, an improvement for Stage 2, and the best scores for Stage 3 experiments. In Stage 1 results, it can be seen, for example an average R^2 of 0.78 in XGBoost Regression and 0.62 with SVR, besides a higher MSE average of 4.49 and 6.31. The MSE is slightly higher than MAE, with 0.73 for XGBoost and 1.27 for SVR, indicating that these averages are due to high peak errors, with an overall good score in most cases. Still, we obtained high outlier errors, as with Ridge Regression, which occurred using specific features and experiment sets.

In Stage 2, a similar case occurs, with similar R^2 and error averages for XGBoost and SVR, besides the higher R^2 peaks of 0.97 for XGBoost, 0.95 for SVR, and 0.97 in Ridge, as can be seen in the range values of these metrics. Yet, in Stage 3, we could obtain some overall good results, with R^2 averages above 0.9 and MSE average scores below 1 (l/h), as highlighted in Table 5.3. This shows the improvement of the newly available features in the latest experiment Stages, shown by the higher and more consistent scores obtained in Stage 3.

Concerning the models, the XGBoost had the most consistent and good results in general for all stages and feature sets. As may be seen in Table 5.3, it kept similar results in Stages 1 and 2, being the better one in the first Stage, showing MAE and R^2 averages above 0.9 and below 0.9 respectively, with just slightly lower mean scores than SVR for Stage 2. Furthermore, in Stage 3, it has shown overall better performance

Table 5.3 – Regression results in evaluation metrics.

	Algo.	Info.	MSE ((l/h)^2)	R^2	MAE (l/h)
Stage 1	XGBoost	Range	0.89-11.73	0.54-0.91	0.55-1.02
		Average	4.49	0.78	0.73
		Std dev.	3.64	0.12	0.17
	Ridge Regres.	Range	2.58-4255.82	-434.42 -0.84	0.97-65.19
		Average	414.35	-40.58	9.55
		Std dev.	1219.00	124.99	18.66
	SVR	Range	3.36-16.66	0.35-0.81	0.96-2.09
		Average	6.31	0.62	1.27
		Std dev.	3.53	0.14	0.32
Stage 2	XGBoost	Range	0.93-16.12	0.49-0.97	0.46-1.74
		Average	7.41	0.77	1.01
		Std dev.	6.79	0.22	0.52
	Ridge Regres.	Range	0.94-164.08	-4.19-0.97	0.61-9.88
		Average	26.40	0.17	2.66
		Std dev.	45.52	1.44	2.53
	SVR	Range	1.63-15.41	0.51-0.95	0.79-2.25
		Average	7.01	0.78	1.39
		Std dev.	4.94	0.16	0.46
Stage 3	XGBoost	Range	0.37-0.68	0.96-0.99	0.25-0.42
		Average	0.49	0.98	0.32
		Std dev.	0.08	0.01	0.05
	Ridge Regres.	Range	0.28-1.80	0.75-0.99	0.23-1.35
		Average	0.91	0.91	0.66
		Std dev.	0.57	0.09	0.42
	SVR	Range	0.38-0.71	0.95-0.99	0.30-0.58
		Average	0.58	0.97	0.44
		Std dev.	0.10	0.01	0.08

Source: Author (2023).

in comparison to all models (R^2 scores above 0.96, average MSE and MAE respectively below 0.5 (l/h)² and 0.4 (l/h), among with consistent results shown by the slight deviation in the scores). This indicates the model's capability to get overall good predictions in most cases. In comparison, SVR was also capable of showing promising results, reaching up to 0.95 and 0.99 R^2 scores in Stages 2 and 3, and the average MSE and MAE not above 7 (l/h)^2 and 1.4 l/h respectively. Finally, the Ridge Regression model has shown by far the most lower performances, in some cases having far higher errors, as in Stages 1 and 2, with negative R^2 and extreme MSE and MAE values. This shows that, in general, yet having some good scores in some cases, as the highlighted MAE and MSE range scores in Stage 3, it may not generalize well for most cases.

Regarding the results of the NARX-NN, a model adapted from [Kanarachos, Mathew e Fitzpatrick 2019], we performed tests with different sets of features and

settings for the model, using the same proceeding and similar settings described by the authors with some adjustments for our data. However, the algorithm did not show good scores in comparison to the other explored algorithms, not being possible to replicate the results achieved in their application. Among all the validations, the higher R^2 score obtained was below 0.21, with most cases getting negative scores, besides the high MSE and MAE, showing that the model did not perform well with our data.

Compared to the related works, we achieved results similar to or better than other explored models. For example, in [Hamed, Khafagy e Badry 2021, Liu e Jin 2023], similar algorithms were explored for the regression problem getting an R^2 score of 0.96 and 0.97, while we could obtain up to 0.99 of R^2 in the observed tests with XGBoost and SVR. It is noticeable that in our case, the consumption data was provided by ECU sensors instead of using physical models or database-provided data. Furthermore, the presented results show advancements also in comparison to [Yao et al. 2020], which using even more complex Neural Network models, reached R^2 scores from 0.5 to 0.6, while using computationally less expensive models, we could reach scores between 0.8 and 0.99, besides of an MSE of 0.49 $(l/h)^2$, corresponding to an RMSE of 0.7 l/h in comparison to the 0.87 l/100km RMSE obtained in their work. Finally, although we applied similar tests and settings, as mentioned above, the model explored by [Kanarachos, Mathew e Fitzpatrick 2019] could not show promising results in our dataset. Still, with less expensive models as XGBoost and SVR, we were able to reach an average MSE of 0.49 $(l/h)^2$ and 0.58 $(l/h)^2$ respectively, or even down to 0.28 in Ridge Regression and 0.37 in XGBoost, compared to the 0.43 $(kpl)^2$ obtained by these authors with use of the Recurrent Neural Network.

6 CONCLUSION

This study effectively accomplished its primary objective of monitoring the combustion engine to detect misfires, analyze driving profiles, and predict fuel consumption through machine learning techniques. It utilized variables extracted directly from the electronic control unit (ECU). It employed a variety of machine learning models, aiming to enhance the efficiency of existing systems, prolong the lifespan of engine components, and furnish valuable insights to both engineers and vehicle operators.

The present work implemented six ML models to detect misfires, thereby establishing the effectiveness of these models in accurately classifying this particular fault with a precision superior of 90%. Also, two scenarios related to fuel consumption were validated. First, comparing three ML models to classify the driving profile concerning fuel consumption, and second, comparing three ML regression models to predict the fuel consumption, proving its validity on both subjects with results that indicate a correct categorization of the driver in economic or non-economic and an accurate prediction of consumption. In the classifiers, we could obtain values up to 100% accuracy, precision, and recall, with an average of around 97% in models like XGBoost and similar results with Logistic Regression. This shows an improvement over other works, where lesser scoring was obtained, despite using more complex and expensive models or not real collected data. Furthermore, in the regressors, our models were able to show R^2 scores up to 0.99, an MAE of 0.23 (l/h), besides an average MSE of 0.49 (l/h)^2, reaching down to 0.28 (l/h)^2 in some tests. Compared to similar works, these results also show advancements in reaching good scores with similar or even less expensive models, besides using real data collected for the training and tests.

The evaluation of the models, involving both training and testing phases, utilized authentic data collected from a vehicle's ECU, and the application of the workflows takes place on a cloud server. With real-time data processing during vehicle operation, engineers or other interested parties can monitor the results through local or web applications. Consequently, this research has practical applicability in several automotive sectors, including industry, engineering, and transport in general. Given the increasing emphasis on data acquisition and the multidisciplinary involved in the sector, ECUs and additional vehicle components are expected to generate even more data, so other opportunities for ML-based analysis will be created.

For future efforts, we intend to focus on complementing the analysis with different models so that more failures can be detected to decrease the risk of damage to the engine and increase passenger safety. Another possible analysis is related to the useful life of engine components, which can be monitored in the medium and long term to verify the wear caused by failures and how their monitoring can extend the life of the components. In addition, it would be interesting to verify the loss of accuracy of the

sensors and their acquisitions so that it is possible to adapt the models so that the analyzes carried out remain accurate and do not lose their integrity.

Due to the multidisciplinarity of the automotive industry and the demand for more data, we expect vehicle ECUs and other automotive components to generate more information, creating the opportunity for more complex data-based analyses with ML.

BIBLIOGRAPHY

ABE, S. *Support vector machines for pattern classification*. [S.l.]: Springer, 2005. v. 2.

ABOKYI, E. et al. Industrial growth and emissions of co2 in ghana: The role of financial development and fossil fuel consumption. *Energy Reports*, v. 5, p. 1339–1353, 2019. ISSN 2352-4847.

AGGARWAL, C. et al. *Data mining: the textbook*. [S.l.]: Springer, 2015. v. 1.

AGGARWAL, C. C. et al. Neural networks and deep learning. *Springer*, Springer, v. 10, p. 978–3, 2018.

AIHUA, J. et al. Detection ofenginemisfire bywaveletanalysisofcylinder-headvibrationsignals. *International Journal of Agricultural and Biological Engineering*, v. 1, n. 2, 2008. ISSN 1934-6344. Disponível em: <https://ijabe.org/index.php/ijabe/article/view/4/30>.

ALI, M. U. et al. Co2 emission, economic development, fossil fuel consumption and population density in india, pakistan and bangladesh: A panel investigation. *International Journal of Finance & Economics*, v. 27, n. 1, p. 18–31, 2022.

ANL, A. N. L. *Downloadable Dynamometer Database (D3) testing results for vehicles that run on gasoline or diesel*. 2022.

AZRIN, A. A. et al. An overview of the spark plug engine profile in a spark ignition engine. *IOP Conference Series: Materials Science and Engineering*, IOP Publishing, v. 1092, n. 1, p. 012030, mar 2021. Disponível em: <https://doi.org/10.1088/1757-899x/1092/1/012030>.

BEDRETCHUK, J. P. et al. Low-cost data acquisition system for automotive electronic control units. *Sensors*, v. 23, n. 4, 2023. ISSN 1424-8220.

BOARD, C. A. R. Technical status update and proposed revisions to malfunction and diagnostic system requirements applicable to 1994 and subsequent california passenger cars, light-duty trucks, and medium - duty vehicles – (obdii). *CARB Staff Report*, 1991.

BOGUŚ, P. et al. Nonlinear analysis of combustion engine vibroacoustic signals for misfire detection. SAE International, p. 8, 3 2003. ISSN 0148-7191. Disponível em: <https://www.sae.org/publications/technical-papers/content/2003-01-0354/>.

BOGUś, P.; MERKISZ, J. Misfire detection of locomotive diesel engine by non-linear analysis. *Mechanical Systems and Signal Processing*, v. 19, n. 4, p. 881–899, 2005. ISSN 0888-3270. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0888327004000883>.

BULUT, I. S.; ILHAN, H. Cloud based vehicle and traffic information sharing application architecture for industry 4.0 (iot). In: *2019 International Conference on Information and Telecommunication Technologies and Radio Electronics (UkrMiCo)*. [S.l.: s.n.], 2019. p. 1–7.

CANAL, R. et al. Misfire detection in combustion engines using machine learning techniques. In: IEEE. *XIII Brazilian Symposium on Computing Systems Engineering (SBESC)*. [S.l.], 2023.

CANAL, R.; RIFFEL, F. K.; GRACIOLI, G. Driving profile analysis using machine learning techniques and ecu data. In: *2023 IEEE 32nd International Symposium on Industrial Electronics (ISIE)*. [S.l.: s.n.], 2023. p. 1–6.

CANAL, R.; RIFFEL, F. K.; GRACIOLI, G. Machine learning for real-time fuel consumption prediction and driving profile classification based on ecu data. *Elsevier*, v. 232, 2023. ISSN 0957-4174.

CHEN, J.; Bond Randall, R. Improved automated diagnosis of misfire in internal combustion engines based on simulation models. *Mechanical Systems and Signal Processing*, v. 64-65, p. 58–83, 2015. ISSN 0888-3270. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0888327015001727>.

CHEN, J. et al. Automated misfire diagnosis in engines using torsional vibration and block rotation. *Journal of Physics: Conference Series*, IOP Publishing, v. 364, p. 012020, may 2012. Disponível em: <https://doi.org/10.1088/1742-6596/364/1/012020>.

CHEN, S. K. et al. Machine learning for misfire detection in a dynamic skip fire engine. *SAE International Journal of Engines*, SAE International, v. 11, n. 6, p. 965–976, 2018. ISSN 19463936, 19463944.

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

CHEN, Y. et al. Data-driven fuel consumption estimation: A multivariate adaptive regression spline approach. *Transportation Research Part C: Emerging Technologies*, v. 83, p. 134–145, 2017. ISSN 0968-090X.

CLAßEN, J. et al. Real driving emission calibration—review of current validation methods against the background of future emission legislation. *Applied Sciences*, v. 11, n. 12, 2021. ISSN 2076-3417.

DEVASENAPATI, S. B.; SUGUMARAN, V.; RAMACHANDRAN, K. Misfire identification in a four-stroke four-cylinder petrol engine using decision tree. *Expert Systems with Applications*, v. 37, n. 3, p. 2150–2160, 2010. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417409007301>.

DREISEITL, S.; OHNO-MACHADO, L. Logistic regression and artificial neural network classification models: a methodology review. *Journal of Biomedical Informatics*, v. 35, n. 5, p. 352–359, 2002. ISSN 1532-0464.

DUA, D.; GRAFF, C. *UCI Machine Learning Repository*. 2017. Disponível em: <http://archive.ics.uci.edu/ml>.

DZIUBINSKI, M. et al. Modelling characteristics of spark ignition engine injection system. *Advances in Science and Technology Research Journal*, v. 11, p. 103–117, 06 2017. Disponível em: <http://www.astrj.com/Modelling-characteristics-of-spark-ignition-engine-injection-system,70565,0,2.html>.

FIRMINO, J. L. et al. Misfire detection of an internal combustion engine based on vibration and acoustic analysis. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, v. 43, n. 336, 2021. ISSN 1806-3691. Disponível em: <https://doi.org/10.1007/s40430-021-03052-y>.

FRANCIS, L. T. et al. Data-driven anomaly detection of engine knock based on automotive ecu. In: *2022 XII Brazilian Symposium on Computing Systems Engineering (SBESC)*. [S.l.: s.n.], 2022. p. 1–8.

FTOUTOU, E.; CHOUCHANE, M. Detection of diesel engine misfire by vibration analysis. In: *Congrès Tunisien de Mécanique COTUM'08*. Hammamet: [s.n.], 2008.

GAO, J. et al. Fuel consumption and exhaust emissions of diesel vehicles in worldwide harmonized light vehicles test cycles and their sensitivities to eco-driving factors. *Energy Conversion and Management*, v. 196, p. 605–613, 2019. ISSN 0196-8904.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

HAFEEZ, A. et al. Machine learning based ecu detection for automotive security. In: IEEE. *2021 17th International Computer Engineering Conference (ICENCO)*. [S.l.], 2021. p. 73–81.

HAMED, M. A.; KHAFAGY, M. H.; BADRY, R. M. Fuel consumption prediction model using machine learning. *Int. Journal of Advanced Computer Science and Applications*, Science and Information (SAI) Organization Limited, v. 12, n. 11, 2021.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The elements of statistical learning: data mining, inference and prediction*. 2. ed. [S.l.]: Springer, 2009.

HAWKINS, M. et al. Cyber-physical production networks, internet of things-enabled sustainability, and smart factory performance in industry 4.0-based manufacturing systems. *Economics, Management, and Financial Markets*, Addleton Academic Publishers, v. 16, n. 2, p. 73–83, 2021.

HAYKIN, S.; NETWORK, N. A comprehensive foundation. *Neural networks*, v. 2, n. 2004, p. 41, 2004.

HMIDA, A. et al. Effects of misfire on the dynamic behavior of gasoline engine crankshafts. *Engineering Failure Analysis*, v. 121, p. 105149, 2021. ISSN 1350-6307. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1350630720316733>.

HOLDEN, E.; GILPIN, G.; BANISTER, D. Sustainable mobility at thirty. *Sustainability*, MDPI, v. 11, n. 7, p. 1965, 2019.

HU, C.; LI, A.; ZHAO, X. Multivariate statistical analysis strategy for multiple misfire detection in internal combustion engines. *Mechanical Systems and Signal Processing*, v. 25, n. 2, p. 694–703, 2011. ISSN 0888-3270. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0888327010002955>.

HUANG, Y.; MENG, S. Automobile insurance classification ratemaking based on telematics driving data. *Decision Support Systems*, v. 127, p. 113156, 2019. ISSN 0167-9236.

HUANG, Y. et al. Fuel consumption and emissions performance under real driving: Comparison between hybrid and conventional vehicles. *Science of The Total Environment*, v. 659, p. 275–282, 2019. ISSN 0048-9697.

HäNDEL, P. et al. Smartphone-based measurement systems for road vehicle traffic monitoring and usage-based insurance. *IEEE Systems Journal*, v. 8, n. 4, p. 1238–1248, 2014.

İNCI, M. et al. A review and research on fuel cell electric vehicles: Topologies, power electronic converters, energy management methods, technical challenges, marketing and future aspects. *Renewable and Sustainable Energy Reviews*, Elsevier, v. 137, p. 110648, 2021.

JAFARIAN, K. et al. Misfire and valve clearance faults detection in the combustion engines based on a multi-sensor vibration signal monitoring. *Measurement*, v. 128, p. 527–536, 2018. ISSN 0263-2241. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0263224118303439>.

KALOGIROU, S. A. Artificial intelligence for the modeling and control of combustion processes: a review. *Progress in Energy and Combustion Science*, v. 29, n. 6, p. 515–566, 2003. ISSN 0360-1285.

KANARACHOS, S.; MATHEW, J.; FITZPATRICK, M. E. Instantaneous vehicle fuel consumption estimation using smartphones and recurrent neural networks. *Expert Systems with Applications*, v. 120, p. 436–447, 2019. ISSN 0957-4174.

KATREDDI, S.; THIRUVENGADAM, A. Trip based modeling of fuel consumption in modern heavy-duty vehicles using artificial intelligence. *Energies*, v. 14, n. 24, 2021. ISSN 1996-1073.

KIENCKE, U. Engine misfire detection. *Control Engineering Practice*, v. 7, n. 2, p. 203–208, 1999. ISSN 0967-0661. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0967066198001506>.

KIKTOVA, E. et al. Comparison of feature selection algorithms for acoustic event detection system. In: *Proceedings ELMAR-2014*. [S.l.: s.n.], 2014. p. 1–4.

KLENK, M. et al. Misfire detection by evaluating crankshaft speed - a means to comply with obdii. *SAE Transactions*, SAE International, v. 102, p. 598–607, 1993. ISSN 0096736X, 25771531. Disponível em: <http://www.jstor.org/stable/44611400>.

KOMORSKA, I. Detection of the engine head gasket defects on the basis of vibration signal. *Silniki Spalinowe*, 2011. ISSN 0138-0346. Disponível em: <https://www.infona.pl/resource/bwmeta1.element.baztech-article-LOD6-0030-0017>.

KPMG. Industry leaders foresee dramatic changes: Where the opportunities may lie. *Global Automotive Executive Survey*, v. 22, 2021.

KRUSE, T.; KURZ, S.; LANG, T. Modern statistical modeling and evolutionary optimization methods for the broad use in ecu calibration. *IFAC Proceedings Volumes*, v. 43, n. 7, p. 739–743, 2010. ISSN 1474-6670. 6th IFAC Symposium on Advances in Automotive Control. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1474667015369184>.

KUMAR, V.; MINZ, S. Feature selection: a literature review. *SmartCR*, v. 4, n. 3, p. 211–229, 2014.

LATTANZI, E.; FRESCHI, V. Machine learning techniques to identify unsafe driving behavior by means of in-vehicle sensor data. *Expert Systems with Applications*, v. 176, p. 114818, 2021. ISSN 0957-4174.

LEI, Y. et al. Applications of machine learning to machine fault diagnosis: A review and roadmap. *Mechanical Systems and Signal Processing*, v. 138, p. 106587, abr. 2020.

LI, C. et al. Variability in real-world emissions and fuel consumption by diesel construction vehicles and policy implications. *Science of The Total Environment*, v. 786, p. 147256, 2021. ISSN 0048-9697.

LI, J. et al. Feature selection: A data perspective. *ACM computing surveys (CSUR)*, ACM New York, NY, USA, v. 50, n. 6, p. 1–45, 2017.

LI, Y. et al. Multilayer perceptron method to estimate real-world fuel consumption rate of light duty vehicles. *Ieee Access*, IEEE, v. 7, p. 63395–63402, 2019.

LIIMATAINEN, H. Utilization of fuel consumption data in an ecodriving incentive system for heavy-duty vehicle drivers. *IEEE Trans. on Intelligent Transportation Systems*, v. 12, n. 4, p. 1087–1095, 2011.

LILLY, I. R. *Diesel engine reference book*. Woburn: Butterworth-Heinemann, 1999. v. 2.

LIU, B. et al. Misfire detection of a turbocharged diesel engine by using artificial neural networks. *Applied Thermal Engineering*, v. 55, n. 1, p. 26–32, 2013. ISSN 1359-4311. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1359431113001452>.

LIU, X.; JIN, H. High-precision transient fuel consumption model based on support vector regression. *Fuel*, v. 338, p. 127368, 2023. ISSN 0016-2361.

LOIS, D. et al. Multivariate analysis of fuel consumption related to eco-driving: Interaction of driving patterns and external factors. *Transportation Research Part D: Transport and Environment*, v. 72, p. 232–242, 2019. ISSN 1361-9209.

LY, M. V.; MARTIN, S.; TRIVEDI, M. M. Driver classification and driving style recognition using inertial sensors. In: *2013 IEEE Intelligent Vehicles Symposium (IV)*. [S.l.: s.n.], 2013. p. 1040–1045.

MA, X.; SHAHBAKHTI, M.; CHIGAN, C. Deep learning based distributed meta-learning for fast and accurate online adaptive powertrain fuel consumption modeling. *IEEE Transactions on Vehicular Technology*, p. 1–14, 2023.

MCCORD, K. *Automotive Diagnostic Systems: Understanding OBD I and OBD II*. [S.l.]: CarTech Inc, 2011.

MERKISZ, J.; BOGUS, P.; GRZESZCZYK, R. Overview of engine misfire detection methods used in on board diagnostics. *Journal of Kones, Combustion Engines*, v. 8, n. 1-2, p. 326–341, 2001.

MIRI, I.; FOTOUHI, A.; EWIN, N. Electric vehicle energy consumption modelling and estimation—a case study. *International Journal of Energy Research*, Wiley Online Library, v. 45, n. 1, p. 501–520, 2021.

MONTANI, M.; SPECIALE, N. Multiple misfire identification by a wavelet-based analysis of crankshaft speed fluctuation. In: *2006 IEEE International Symposium on Signal Processing and Information Technology*. [S.l.: s.n.], 2006. p. 144–148.

MORADI, E.; MIRANDA-MORENO, L. Vehicular fuel consumption estimation using real-world measures through cascaded machine learning modeling. *Transportation Research Part D: Transport and Environment*, v. 88, p. 102576, 2020. ISSN 1361-9209.

MURATA, N. et al. On-line learning in changing environments with applications in supervised and unsupervised learning. *Neural Networks*, v. 15, n. 4, p. 743–760, 2002. ISSN 0893-6080. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0893608002000606>.

NAG, P.; GHANEKAR, U.; HARMALKAR, J. A novel multi-core approach for functional safety compliance of automotive electronic control unit according to iso 26262. In: IEEE. *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*. [S.l.], 2019. p. 1–5.

NAKAGAWA, S.; FUKUCHI, E.; NUMATA, A. A new diagnosis method for an air-fuel ratio cylinder imbalance. *SAE 2012 World Congress & Exhibition*, 2012. ISSN 0148-7191. Disponível em: <https://saemobilus.sae.org/content/2012-01-0718/>.

NGATIMAN, N. A.; NUAWI, M. Z. Hybrid vehicle engine misfire detection using piezo-film sensors and analysing with z-freq. *Journal of Mechanical Engineering*, Faculty of Mechanical Engineering Universiti Teknologi MARA (UiTM), v. 7, n. 1, p. 269–285, 2018. ISSN 1823-5514. Disponível em: <https://ir.uitm.edu.my/id/eprint/41749/>.

NGUYEN, T.-H.; CHEON, B. M.; JEON, J. W. Can fd performance analysis for ecu re-programming using the canoe. In: *The 18th IEEE International Symposium on Consumer Electronics (ISCE 2014)*. [S.l.: s.n.], 2014. p. 1–4.

PAVLOVIC, J. et al. Understanding the origins and variability of the fuel consumption gap: Lessons learned from laboratory tests and a real-driving campaign. *Environmental Sciences Europe*, Springer, v. 32, p. 1–16, 2020.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011.

PEPPES, N. et al. Driving behaviour analysis using machine and deep learning methods for continuous streams of vehicular data. *Sensors*, v. 21, n. 14, 2021. ISSN 1424-8220.

PING, P. et al. Impact of driver behavior on fuel consumption: Classification, evaluation and prediction using machine learning. *IEEE Access*, v. 7, p. 78515–78532, 2019.

QIN, C. et al. Dtcnnmi: A deep twin convolutional neural networks with multi-domain inputs for strongly noisy diesel engine misfire detection. *Measurement*, v. 180, p. 109548, 2021. ISSN 0263-2241. Disponível em: <https://www.sciencedirect.com/science/article/pii/S026322412100525X>.

RAHMAN, A.; SMITH, A. D. Predicting fuel consumption for commercial buildings with machine learning algorithms. *Energy and Buildings*, v. 152, p. 341–358, 2017. ISSN 0378-7788.

RAHNAMA, P.; ARAB, M.; REITZ, R. D. A time-saving methodology for optimizing a compression ignition engine to reduce fuel consumption through machine learning. *SAE International Journal of Engines*, SAE International, v. 13, n. 2, p. 267–288, 2020. ISSN 19463936, 19463944.

RATH, M. et al. Analysis of autoregressive coefficients of knock sensor signals for misfire detection in internal combustion engines. In: *2019 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. [S.l.: s.n.], 2019. p. 1–6.

REYNOLDS, D. A. et al. Gaussian mixture models. *Encyclopedia of biometrics*, Berlin, Springer, v. 741, n. 659-663, 2009.

RIOS-TORRES, J.; LIU, J.; KHATTAK, A. sc. *International Journal of Sustainable Transportation*, Taylor & Francis, v. 13, n. 2, p. 123–137, 2019.

RUSSELL, S. J. *Artificial intelligence a modern approach*. [S.l.]: Pearson Education, Inc., 2010.

SHAHID, S. M.; KO, S.; KWON, S. Real-time abnormality detection and classification in diesel engine operations with convolutional neural network. *Expert Systems with Applications*, v. 192, p. 116233, 2022. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417421015451>.

SHAHVERDY, M. et al. Driver behavior detection and classification using deep convolutional neural networks. *Expert Systems with Applications*, v. 149, p. 113240, 2020. ISSN 0957-4174.

SHARMA, A.; SUGUMARAN, V.; DEVASENAPATI, S. B. Misfire detection in an ic engine using vibration signal and decision tree algorithms. *Measurement*, v. 50, p. 370–380, 2014. ISSN 0263-2241. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0263224114000244>.

SILVA, M. E. R. da; GRACIOLI, G.; ARAUJO, G. M. de. Feature selection in machine learning for knocking noise detection. In: *2022 XII Brazilian Symposium on Computing Systems Engineering (SBESC)*. [S.l.: s.n.], 2022. p. 1–8.

SINGH, S.; POTALA, S.; MOHANTY, A. An improved method of detecting engine misfire by sound quality metrics of radiated sound. *Proceedings of the Institution of Mechanical Engineers Part D Journal of Automobile Engineering*, v. 233, 2018.

SLAVIN, D. et al. Empirical modeling of vehicle fuel economy based on historical data. In: *The 2013 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2013. p. 1–6.

SYTA, A.; CZARNIGOWSKI, J.; JAKLIńSKI, P. Detection of cylinder misfire in an aircraft engine using linear and non-linear signal analysis. *Measurement*, v. 174, p. 108982, 2021. ISSN 0263-2241. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0263224121000191>.

TAGHAVI, M.; SHOARAN, M. Hardware complexity analysis of deep neural networks and decision tree ensembles for real-time neural data classification. In: *2019 9th International IEEE/EMBS Conference on Neural Engineering (NER)*. [S.l.: s.n.], 2019. p. 407–410.

TINAUT, F. V. et al. Misfire and compression fault detection through the energy model. *Mechanical Systems and Signal Processing*, v. 21, n. 3, p. 1521–1535, 2007. ISSN 0888-3270. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0888327006001191>.

TING, L. L.; MAYER J. E., J. Piston Ring Lubrication and Cylinder Bore Wear Analysis, Part I—Theory. *Journal of Lubrication Technology*, v. 96, n. 3, p. 305–313, 07 1974. ISSN 0022-2305. Disponível em: <https://doi.org/10.1115/1.3451948>.

TOLEDO, G.; SHIFTAN, Y. Can feedback from in-vehicle data recorders improve driver behavior and reduce fuel consumption? *Transportation Research Part A: Policy and Practice*, v. 94, p. 194–204, 2016. ISSN 0965-8564.

TSAKALIDIS, A. et al. Catalyzing sustainable transport innovation through policy support and monitoring: The case of trimis and the european green deal. *Sustainability*, MDPI, v. 12, n. 8, p. 3171, 2020.

WANG, R. et al. Research on the fault monitoring method of marine diesel engines based on the manifold learning and isolation forest. *App. O. Res.*, v. 112, p. 102681, 2021. ISSN 0141-1187.

WANG, S. et al. Predicting ship fuel consumption based on lasso regression. *Transportation Research Part D: Transport and Environment*, v. 65, p. 817–824, 2018. ISSN 1361-9209.

WANG, X. et al. Misfire detection using crank speed and long short-term memory recurrent neural network. *Energies*, v. 15, n. 1, 2022. ISSN 1996-1073.

WEBER, M. et al. A hybrid anomaly detection system for electronic control units featuring replicator neural networks. In: ARAI, K.; KAPOOR, S.; BHATIA, R. (Ed.). *Advances in Information and Communication Networks*. Cham: Springer International Publishing, 2019. p. 43–62. ISBN 978-3-030-03405-4.

WEHMEYER, C.; NOÉ, F. Time-lagged autoencoders: Deep learning of slow collective variables for molecular kinetics. *The Journal of chemical physics*, AIP Publishing LLC, v. 148, n. 24, p. 241703, 2018.

WEI, H. et al. Gasoline engine exhaust gas recirculation – a review. *Applied Energy*, v. 99, p. 534–544, 2012. ISSN 0306-2619. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0306261912003595>.

WELLER, K. et al. Real world fuel consumption and emissions from ldvs and hdvs. *Frontiers in Mechanical Engineering*, Frontiers Media SA, v. 5, p. 45, 2019.

WU, J.-D.; LIU, C.-H. An expert system for fault diagnosis in internal combustion engines using wavelet packet transform and neural network. *Expert Systems with Applications*, v. 36, n. 3, Part 1, p. 4278–4286, 2009. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417408001814>.

XIE, X. et al. Fuel consumption prediction models based on machine learning and mathematical methods. *Journal of Marine Science and Engineering*, MDPI, v. 11, n. 4, p. 738, 2023.

XU, C. et al. Misfire detection based on generalized force identification at the engine centre of gravity. *IEEE Access*, v. 7, p. 165039–165047, 2019.

YANG, S. et al. Driving-style-oriented adaptive equivalent consumption minimization strategies for hevs. *IEEE Trans. on Vehicular Technology*, v. 67, n. 10, p. 9249–9261, 2018.

YANG, Y. et al. Influence of driving style on traffic flow fuel consumption and emissions based on the field data. *Physica A: Stat. Mechanics and its Apps.*, v. 599, p. 127520, 2022. ISSN 0378-4371.

YAO, Y. et al. Vehicle fuel consumption prediction method based on driving behavior data collected from smartphones. *Journal of Advanced Transportation*, Hindawi Limited, v. 2020, p. 1–11, 2020.

YAO, Y. et al. Modeling of individual vehicle safety and fuel consumption under comprehensive external conditions. *Transportation Research Part D: Transport and Environment*, v. 79, p. 102224, 2020. ISSN 1361-9209.

YE, J. Application of extension theory in misfire fault diagnosis of gasoline engines. *Expert Systems with Applications*, Elsevier, v. 36, n. 2, p. 1217–1221, 2009.

ZEBARI, R. et al. A comprehensive review of dimensionality reduction techniques for feature selection and feature extraction. *Journal of Applied Science and Technology Trends*, v. 1, n. 2, p. 56 – 70, May 2020.

ZHANG, Y.-T. et al. Develop of a fuel consumption model for hybrid vehicles. *Energy Conversion and Management*, Elsevier, v. 207, p. 112546, 2020.

ZHAO, Z.; LIU, H. Spectral feature selection for supervised and unsupervised learning. In: *Proceedings of the 24th International Conference on Machine Learning*. New York, NY, USA: Association for Computing Machinery, 2007. (ICML '07), p. 1151–1157. ISBN 9781595937933. Disponível em: <https://doi.org/10.1145/1273496.1273641>.

ZHENG, T. et al. Real-time combustion torque estimation and dynamic misfire fault diagnosis in gasoline engine. *Mechanical Systems and Signal Processing*, v. 126, p. 521–535, 2019. ISSN 0888-3270. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0888327019301396>.

ZHENG, X. et al. Real-world fuel consumption of light-duty passenger vehicles using on-board diagnostic (obd) systems. *Frontiers of Environmental Science & Engineering*, Springer, v. 14, p. 1–10, 2020.

ZHENG, X. et al. Real-time driving style classification based on short-term observations. *IET Communications*, v. 16, n. 12, p. 1393 – 1402, 2022.

ZHOU, M.; JIN, H.; WANG, W. A review of vehicle fuel consumption models to evaluate eco-driving and eco-routing. *Transportation Research Part D: Transport and Environment*, v. 49, p. 203–218, 2016. ISSN 1361-9209.

ZIÓŁKOWSKI, J. et al. Use of artificial neural networks to predict fuel consumption on the basis of technical parameters of vehicles. *Energies*, v. 14, n. 9, 2021. ISSN 1996-1073.

ÇAPRAZ, A. G. et al. Fuel consumption models applied to automobiles using real-time data: A comparison of statistical models. *Procedia Computer Science*, v. 83, p. 774–781, 2016. ISSN 1877-0509. The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016) / The 6th International Conference on Sustainable Energy Information Technology (SEIT-2016) / Affiliated Workshops.

APPENDIX A – MACHINE LEARNING ALGORITHM APPLICATION

A.0.1 Decision Tree

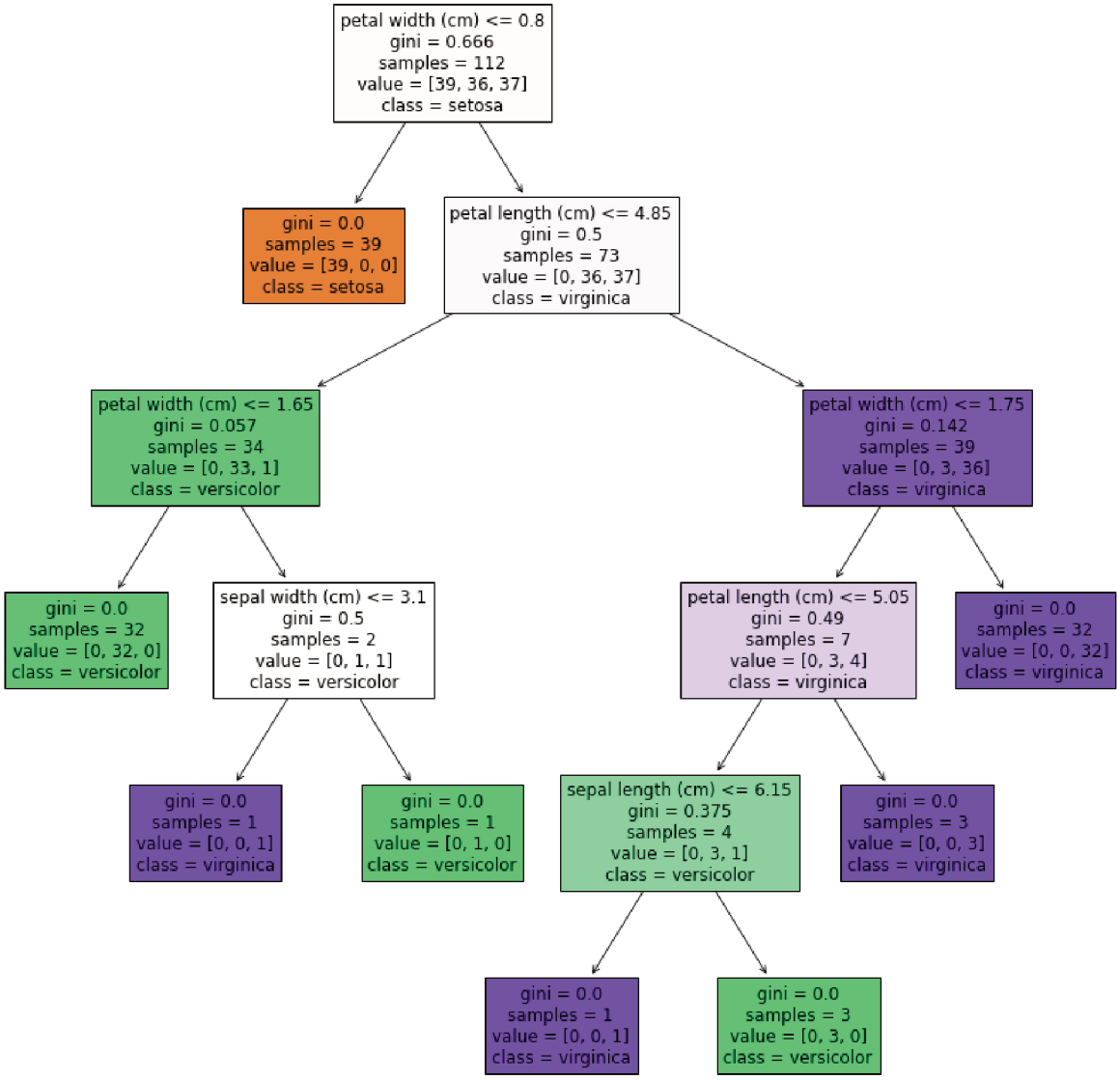
In this code example (Alg. A.1), the libraries matplotlib and SKLearn were used to create and train the decision tree and to plot an image to show the result of the algorithm, respectively. It starts by importing the libraries for its execution and loading the dataset. After the dataset is divided into 4: train, test, and their respective targets, the tree is trained and printed for visualization (Figure A.1). Finally, the created tree is tested and evaluated for its performance. The evaluation metrics (Figure A.2) estimate the algorithm's performance, checking if it has learned as expected or needs retraining or other adaptations.

Algorithm A.1 – Decision tree algorithm application.

```
1   #importing the required libraries
2   from sklearn.model_selection import train_test_split
3   from sklearn.metrics import classification_report
4   from sklearn.tree import plot_tree, DecisionTreeClassifier
5   from sklearn.datasets import load_iris
6   import matplotlib.pyplot as plt
7
8   #loading iris dataset imported with sklearn
9   dataset = load_iris()
10
11  #spliting the dataset into train and test, with their
     respective targets
12  input_train, input_test, output_train, output_test =
     train_test_split(dataset.data, dataset.target, test_size =
     0.25)
13
14  #training the tree
15  model = DecisionTreeClassifier().fit(input_train,
     output_train)
16
17  #ploting the tree
18  plt.figure(figsize=(15,15))
19  plot_tree(model, filled=True, class_names = dataset.
     target_names, feature_names = dataset.feature_names)
20  plt.show()
21
22  #testing and evaluating the model with a prediction output
23  output_prediction = model.predict(input_test)
```

```
24      print(classification_report(output_test, output_prediction,
         target_names = dataset.target_names ))
```

Figure A.1 – Decision tree created with A.1.



Source: Author (2023).

A.0.2 K-Means

In this code example (Alg. A.2), we split the iris dataset into training and test sets, then we perform k-means clustering on the training set and evaluate the clustering performance on the test set. The evaluation metric is calculated using silhouette score

Figure A.2 – Decision tree (A.1) metrics.

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	11
versicolor	1.00	0.93	0.96	14
virginica	0.93	1.00	0.96	13
accuracy			0.97	38
macro avg	0.98	0.98	0.98	38
weighted avg	0.98	0.97	0.97	38

Source: Author (2023).

from sklearn metrics. The code includes two plots: one for the training set and another for the test set. Each plot shows the data points colored based on their assigned clusters, and the cluster centroids are marked with red crosses. Finally, the code outputs the silhouette score as an evaluation metric for the clustering performance on the test set.

Algorithm A.2 – K-Means algorithm application.

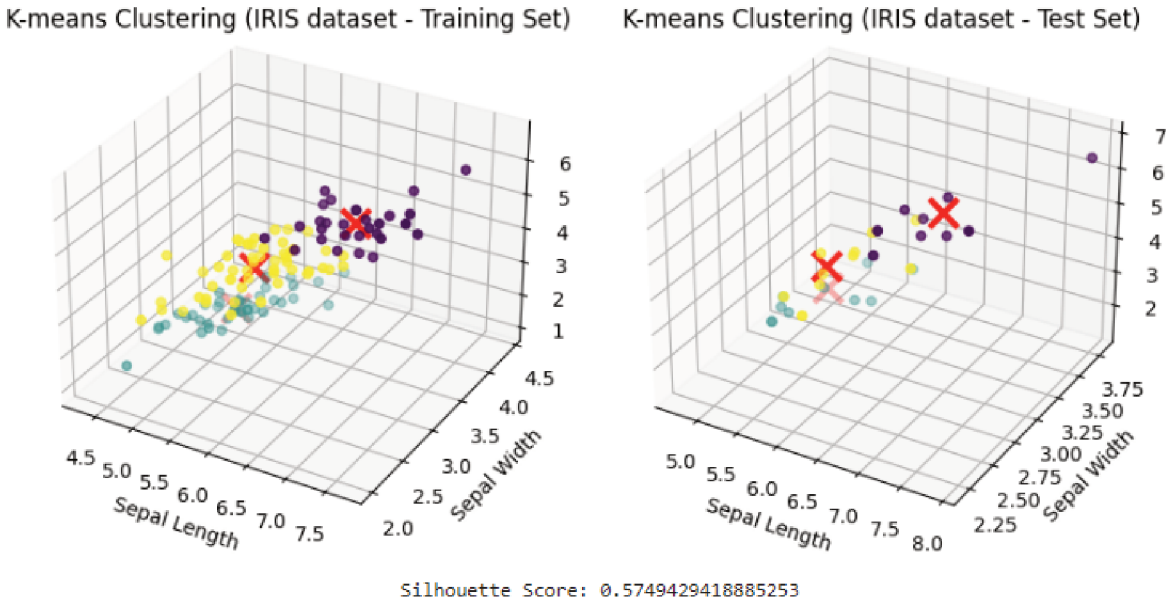
```
# Importing the required libraries
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import silhouette_score

# Loading iris dataset imported with sklearn
iris = load_iris()
# Consider all three features (sepal length, sepal width, and
  petal length)
X = iris.data
y = iris.target

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
 test_size=0.2, random_state=42)

# Perform k-means clustering on the training set
# Number of clusters
k = 3
```

```
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X_train)
    labels_train = kmeans.labels_
    centroids_train = kmeans.cluster_centers_

    # Evaluate the clustering performance on the test set
    labels_test = kmeans.predict(X_test)
    silhouette_avg = silhouette_score(X_test, labels_test)

    # Plot the training data points and cluster centroids
    fig = plt.figure()
    ax = Figureadd_subplot(111, projection='3d')
    ax.scatter(X_train[:, 0], X_train[:, 1], X_train[:, 2], c=
     labels_train)
    ax.scatter(centroids_train[:, 0], centroids_train[:, 1],
     centroids_train[:, 2], marker='x', s=200, linewidths=3, c='r'
     )
    ax.set_xlabel('Sepal Length')
    ax.set_ylabel('Sepal Width')
    ax.set_zlabel('Petal Length')
    ax.set_title('K-means Clustering (IRIS dataset - Training Set
     )')
    plt.show()

    # Plot the test data points and cluster centroids
    fig = plt.figure()
    ax = Figureadd_subplot(111, projection='3d')
    ax.scatter(X_test[:, 0], X_test[:, 1], X_test[:, 2], c=
     labels_test)
    ax.scatter(centroids_train[:, 0], centroids_train[:, 1],
     centroids_train[:, 2], marker='x', s=200, linewidths=3, c='r'
     )
    ax.set_xlabel('Sepal Length')
    ax.set_ylabel('Sepal Width')
    ax.set_zlabel('Petal Length')
    ax.set_title('K-means Clustering (IRIS dataset - Test Set)')
    plt.show()

    # Print the evaluation metric
    print(f"Silhouette Score: {silhouette_avg}")
```

By running this code, separate visualizations for the training and test sets are

obtained, and you can assess the clustering performance using the silhouette score (low score due to the fact that one of the classes is not linearly separable), as shown in Figure A.3.

Figure A.3 – K-Means (A.2) metrics.



Source: Author (2023).

A.0.3 Logistic Regression

In this code example (Alg. A.3), we load the iris dataset and split the data into training and test sets. Initializes a logistic regression model and fits it to the training data, and uses the trained model to make predictions on the test set. Calculates the accuracy score and prints it. Generates a classification report that includes precision, recall, F1-score, and support. Computes the confusion matrix and plots it using a heat map.

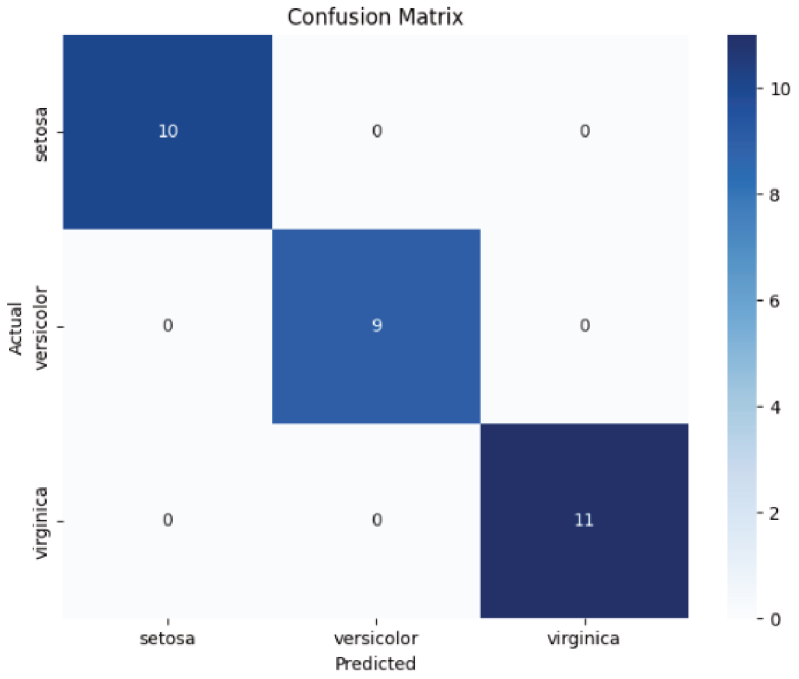
Algorithm A.3 – Logistic Regression algorithm application.

```
# Importing the required libraries
import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

```
9       from sklearn.metrics import accuracy_score,
         classification_report, confusion_matrix
10
11      # Load the iris dataset
12      iris = load_iris()
13      X = iris.data
14      y = iris.target
15      class_names = iris.target_names
16
17      # Split the data into training and test sets
18      X_train, X_test, y_train, y_test = train_test_split(X, y,
         test_size=0.2, random_state=42)
19
20      # Train a logistic regression model
21      model = LogisticRegression()
22      model.fit(X_train, y_train)
23
24      # Make predictions on the test set
25      y_pred = model.predict(X_test)
26
27      # Compute the accuracy score
28      accuracy = accuracy_score(y_test, y_pred)
29      print(f"Accuracy: {accuracy}")
30
31      # Generate a classification report
32      report = classification_report(y_test, y_pred, target_names=
         class_names)
33      print("Classification Report:")
34      print(report)
35
36      # Generate a confusion matrix
37      cm = confusion_matrix(y_test, y_pred)
38      df_cm = pd.DataFrame(cm, index=class_names, columns=
         class_names)
39
40      # Plot the confusion matrix
41      plt.figure(figsize=(8, 6))
42      plt.title("Confusion Matrix")
43      sns.heatmap(df_cm, annot=True, cmap="Blues", fmt="d")
44      plt.xlabel("Predicted")
45      plt.ylabel("Actual")
46      plt.show()
```

By running this code, you will see the confusion matrix and the evaluation metrics for the Logistic Regression classifier applied to the iris dataset (Figs. A.4 and A.5)

Figure A.4 – Logistic Regression confusion matrix (A.3).



Source: Author (2023).

Figure A.5 – Logistic Regression (A.6) metrics.

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Source: Author (2023).

A.0.4 K Neighbors

In this example code (Alg. A.4), the iris dataset is loaded from scikit-learn. The data is split into training and test sets. A KNeighborsClassifier model is trained on the training data, and predictions are made on the test set using the trained model.

The accuracy score is computed and printed and a classification report is generated, including precision, recall, F1-score, and support. A confusion matrix is computed and plotted using a heat map.

Algorithm A.4 – K Neighbors algorithm application.

```
# Importing the required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn. neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,
 classification_report, confusion_matrix

# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target
class_names = iris.target_names

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
 test_size=0.2, random_state=42)

# Train a KNeighborsClassifier model
model = KNeighborsClassifier()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Compute the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Generate a classification report
report = classification_report(y_test, y_pred, target_names=
 class_names)
print("Classification Report:")
print(report)
```

```
35
36      # Generate a confusion matrix
37      cm = confusion_matrix(y_test, y_pred)
38      df_cm = pd.DataFrame(cm, index=class_names, columns=
         class_names)
39
40      # Plot the confusion matrix
41      plt.figure(figsize=(8, 6))
42      plt.title("Confusion Matrix")
43      sns.heatmap(df_cm, annot=True, cmap="Blues", fmt="d")
44      plt.xlabel("Predicted")
45      plt.ylabel("Actual")
46      plt.show()
```

By running this code, the same results presented for Logistic Regression (Figs. A.4 and A.5) are obtained.

A.0.5 Support Vector Classification

The iris dataset is loaded and splits into training and testing sets in this example code (Alg. A.5). An SVC model is trained on the training data, and predictions are made on the test set using the trained model. The accuracy score is computed and printed. A classification report is generated, including precision, recall, F1-score, and support. A confusion matrix is computed and plotted using a heat map.

Algorithm A.5 – Support Vector Classification algorithm application.

```
1       # Importing the required libraries
2       import numpy as np
3       import pandas as pd
4       import matplotlib.pyplot as plt
5       import seaborn as sns
6       from sklearn.datasets import load_iris
7       from sklearn.svm import SVC
8       from sklearn.model_selection import train_test_split
9       from sklearn.metrics import accuracy_score,
         classification_report, confusion_matrix
10
11      # Load the iris dataset
12      iris = load_iris()
13      X = iris.data
14      y = iris.target
15      class_names = iris.target_names
16
```

```
    # Split the data into training and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
     test_size=0.2, random_state=42)

    # Train an SVC model
    model = SVC()
    model.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = model.predict(X_test)

    # Compute the accuracy score
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {accuracy}")

    # Generate a classification report
    report = classification_report(y_test, y_pred, target_names=
     class_names)
    print("Classification Report:")
    print(report)

    # Generate a confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    df_cm = pd.DataFrame(cm, index=class_names, columns=
     class_names)

    # Plot the confusion matrix
    plt.figure(figsize=(8, 6))
    plt.title("Confusion Matrix")
    sns.heatmap(df_cm, annot=True, cmap="Blues", fmt="d")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
```

By running this code, the same results presented for Logistic Regression (Figs. A.4 and A.5) are obtained.

A.0.6 Gradient Boosting

In Algorithm A.6, we first load the iris dataset, then split the dataset into training and test sets. Next, we create a Gradient Boosting classifier using GradientBoosting-Classifier from sklearn.ensemble, specifying the number of estimators and the learning rate as parameters. We train the classifier using the training and make predictions. To

evaluate the classifier's performance, we plot the confusion matrix that shows the true labels against the predicted labels, providing insights into the classifier's performance. Additionally, we can visualize the feature's importance using a bar plot, representing the relative importance of each feature in the classifier's decision-making process.

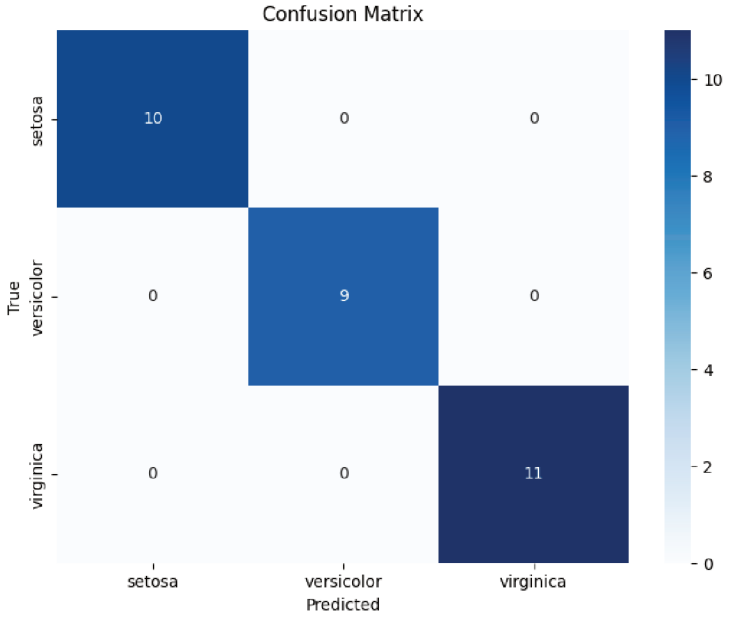
Algorithm A.6 – Gradient Boosting algorithm application.

```
# Importing the required libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
 test_size=0.2, random_state=42)

# Create and train the Gradient Boosting classifier
clf = GradientBoostingClassifier(n_estimators=100,
 learning_rate=0.1, random_state=42)
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
 xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
```

```
34      plt.show()
```

By running this code, the confusion matrix and the bar plot display the feature importance for the Gradient Boosting classifier applied to the iris dataset (Figure A.6).

Figure A.6 – Gradient Boosting (A.6) metrics.



Source: Author (2023).

Another way of implementing gradient boosting is XGBoost classifier A.7. In this example, we first load the iris dataset, then split the dataset into training and testing. We create an instance of the XGBClassifier class, which is trained on the training data using the fit method. Using the prediction method, we use the trained model to make predictions on the test set and calculate the model's accuracy by comparing the predicted labels with the true labels. The accuracy is the ratio of correct predictions to the total number of samples in the test set. We initialize an empty confusion matrix, then we iterate through each sample in the test set and increment the corresponding cell in the confusion matrix based on the true and predicted labels. Finally, we plot the confusion matrix and display it with appropriate labels.

Algorithm A.7 – XGBoost algorithm application.

```
1       # Importing the required libraries
2       import numpy as np
3       import matplotlib.pyplot as plt
4       from sklearn.datasets import load_iris
5       from sklearn.model_selection import train_test_split
6       import xgboost as xgb
```

```
7
8       # Load the iris dataset
9       iris = load_iris()
10      X = iris.data
11      y = iris.target
12
13      # Split the data into training and test sets
14      X_train, X_test, y_train, y_test = train_test_split(X, y,
         test_size=0.2, random_state=42)
15
16      # Create and train the XGBoost classifier
17      model = xgb.XGBClassifier()
18      model.fit(X_train, y_train)
19
20      # Make predictions on the test set
21      y_pred = model.predict(X_test)
22
23      # Calculate the accuracy
24      accuracy = np.mean(y_pred == y_test)
25      print(f"Accuracy: {accuracy}")
26
27      # Plot the confusion matrix
28      confusion_matrix = np.zeros((3, 3))
29      for i in range(len(y_test)):
30          confusion_matrix[y_test[i], int(y_pred[i])] += 1
31
32      plt.imshow(confusion_matrix, interpolation='nearest', cmap=
         plt.cm.Blues)
33      plt.title('Confusion Matrix')
34      plt.colorbar()
35      tick_marks = np.arange(len(iris.target_names))
36      plt.xticks(tick_marks, iris.target_names, rotation=45)
37      plt.yticks(tick_marks, iris.target_names)
38      plt.xlabel('Predicted')
39      plt.ylabel('True')
40      plt.show()
```

By running this code, a single plot that summarizes the results of the XGBoost classifier on the iris dataset is constructed. The plot represents the confusion matrix, providing insights into the classifier's performance. In this case, the results are the same presented before (Figure A.6.

A.0.7 Ridge Regression

In this code example (A.8, the iris dataset is loaded and split into training and test sets. A Ridge Regression model is trained on the training data, and predictions are made on the test set using the trained model. The mean squared error (MSE) and the coefficient of determination (R^2 score) are computed and printed. A density plot is generated to visualize the density of the actual and predicted values using kernel density estimation.

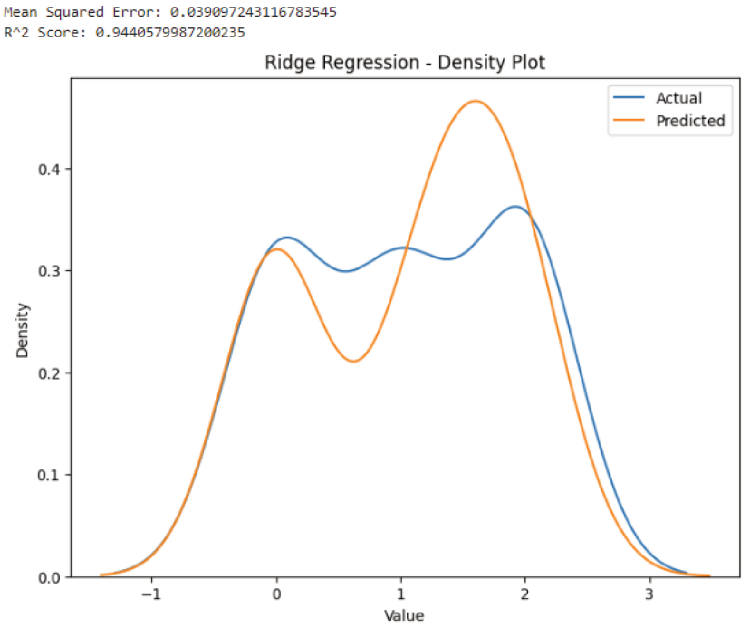
Algorithm A.8 – Ridge Regression algorithm application.

```
# Importing the required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
 test_size=0.2, random_state=42)

# Train a Ridge Regression model
model = Ridge()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Compute the mean squared error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Compute the coefficient of determination (R^2 score)
r2 = r2_score(y_test, y_pred)
```

```
32      print(f"R^2 Score: {r2}")
33
34
35      # Plot the density of predicted and actual values (density
         plot)
36      plt.figure(figsize=(8, 6))
37      sns.kdeplot(y_test, label='Actual')
38      sns.kdeplot(y_pred, label='Predicted')
39      plt.xlabel('Value')
40      plt.ylabel('Density')
41      plt.title('Ridge Regression - Density Plot')
42      plt.legend()
43      plt.show()
```

Running this code provides insights into the distribution of values and allows you to compare the shape and overlap of the two distributions applied to the iris dataset (Figure A.7)

Figure A.7 – Ridge Regression (A.8) metrics.



Source: Author (2023).

A.0.8 Support Vector Regression

In this code example (A.9, the iris dataset is loaded, and the data is split into training and test sets. An SVR model is trained on the training data, and predictions are made on the test set using the trained model. The mean squared error (MSE) and

the coefficient of determination (R^2 score) are computed and printed. A density plot is generated to visualize the density of the actual and predicted values using kernel density estimation

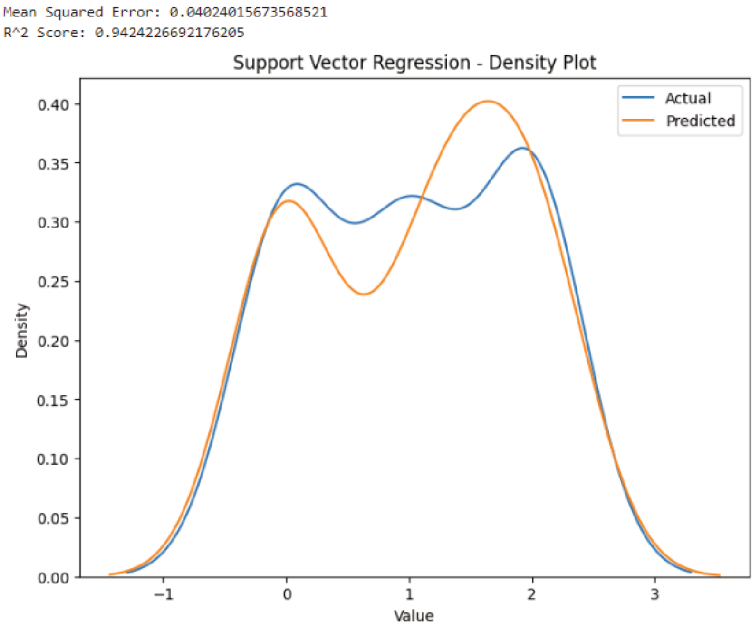
Algorithm A.9 – Support Vector Regression algorithm application.

```
# Importing the required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
 test_size=0.2, random_state=42)

# Train an SVR model
model = SVR()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Compute the mean squared error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# Compute the coefficient of determination (R^2 score)
r2 = r2_score(y_test, y_pred)
print(f"R^2 Score: {r2}")

# Plot the density of predicted and actual values (density
 plot)
plt.figure(figsize=(8, 6))
```

```
36      sns.kdeplot(y_test, label='Actual')
37      sns.kdeplot(y_pred, label='Predicted')
38      plt.xlabel('Value')
39      plt.ylabel('Density')
40      plt.title('Support Vector Regression - Density Plot')
41      plt.legend()
42      plt.show()
```

Running this code provides insights into the distribution of values and allows you to compare the shape and overlap of the two distributions applied to the iris dataset (Figure A.8)

Figure A.8 – Ridge Regression (A.9) metrics.



Source: Author (2023).

A.0.9 Gradient Boosting Regression

In this code example (Alg. A.10), the iris dataset is loaded, and the data is split into training and test sets. The training and test datasets are converted into DMatrix format, the input format used by XGBoost. The parameters for XGBoost regression are defined. In this example, the objective is set to 'reg:squarederror', indicating a regression task, and the evaluation metric is set to 'rmse' (root mean squared error). The XGBoost regression model is trained using the training data, and the defined parameters and predictions are made on the test set using the trained model. The mean squared error (MSE) and the coefficient of determination (R^2 score) are computed and printed. A

density plot is generated to visualize the density of the actual and predicted values using kernel density estimation

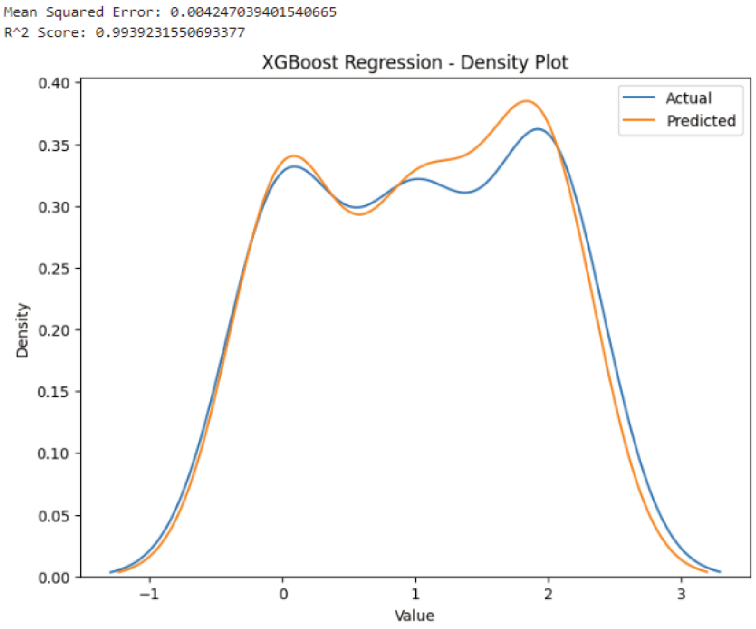
Algorithm A.10 – XGBoost Regressor algorithm application.

```
# Importing the required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import xgboost as xgb

# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
 test_size=0.2, random_state=42)

# Convert the dataset into DMatrix format
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# Define the parameters for XGBoost regression
params = {
    'objective': 'reg:squarederror',
    'eval_metric': 'rmse'
}

# Train the XGBoost regression model
model = xgb.train(params, dtrain)

# Make predictions on the test set
y_pred = model.predict(dtest)

# Compute the mean squared error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
38
39      # Compute the coefficient of determination (R^2 score)
40      r2 = r2_score(y_test, y_pred)
41      print(f"R^2 Score: {r2}")
42
43      # Plot the density of predicted and actual values using
         seaborn
44      plt.figure(figsize=(8, 6))
45      sns.kdeplot(y_test, label='Actual')
46      sns.kdeplot(y_pred, label='Predicted')
47      plt.xlabel('Value')
48      plt.ylabel('Density')
49      plt.title('XGBoost Regression - Density Plot')
50      plt.legend()
51      plt.show()
```

Running this code provides insights into the distribution of values and allows you to compare the shape and overlap of the two distributions applied to the iris dataset (Figure A.9)

Figure A.9 – XGBoost Regressor (A.10) metrics.



Source: Author (2023).

A.0.10 Perceptron

In this code example (Alg. A.11), the libraries numpy was used to convert the dataset into only two classes, and SKLearn was used to create and train the perceptron. It starts by importing the needed libraries. Afterward, the 'versicolor' and 'virginica' classes are merged into one class, leaving only two classes: Setosa or Other, so that the dataset could be used in this example. Then the dataset is split into 4: train, test, and their respective targets. At last, the perceptron was trained and evaluated to see how it performed. The evaluation metrics (Figure A.10) showed that the algorithm could classify all the data correctly.

Algorithm A.11 – Perceptron algorithm.

```
1   #importing the required libraries
2   import numpy as np
3   from sklearn.linear_model import Perceptron
4   from sklearn.model_selection import train_test_split
5   from sklearn.datasets import load_iris
6   from sklearn.metrics import classification_report
7
8   #loading iris dataset imported with sklearn
9   iris = load_iris()
10
11  #making the dataset linearly separable
12  targets = np.where(iris.target == 0, 1, 0)
13
14  #spliting the dataset into train and test, with their
     respective targets
15  input_train, input_test, output_train, output_test =
     train_test_split(iris.data, targets, test_size = 0.25)
16
17  #training the perceptron
18  model = Perceptron(max_iter=10, tol=0.001)
19  model.fit(input_train, output_train)
20
21  #testing e evaluating the model with a prediction output
22  output_prediction = model.predict(input_test)
23  print(classification_report(output_prediction, output_test))
```

A.0.11 MultiLayer Perceptron

In this code example (Alg. A.12), the SKLearn library was used to build and train the MLP. It starts by importing the necessary libraries and dividing the dataset into 4:

Figure A.10 – Perceptron (A.11) metrics.

	precision	recall	f1-score	support
Other	1.00	1.00	1.00	23
Setosa	1.00	1.00	1.00	15
accuracy			1.00	38
macro avg	1.00	1.00	1.00	38
weighted avg	1.00	1.00	1.00	38

Source: Author (2023).

train, test, and their targets. The network structure was defined with three hidden layers with more than 30, 20, and 10 neurons each, respectively, with 150 iterations and an activation function, a function responsible for processing the signal generated by the linear combination of inputs and the synapse weights, for generating the neuron output signal. Subsequently, the MLP was trained and evaluated, and the evaluation metrics (Figure A.11) showed that the algorithm could classify all data correctly.

Algorithm A.12 – Perceptron algorithm.

```
1   #importing of required libraries
2   from sklearn.neural_network import MLPClassifier
3   from sklearn.model_selection import train_test_split
4   from sklearn.datasets import load_iris
5   from sklearn.metrics import classification_report
6
7   #loading iris dataset imported with sklearn
8   iris = load_iris()
9
10  #spliting the dataset into training and test sets, and their
     respective targets
11  input_train, input_test, output_train, output_test =
     train_test_split(iris.data, iris.target, test_size=0.25)
12
13  #defining the structure of the network
14  model = MLPClassifier(hidden_layer_sizes=(30,20,10), max_iter
     =150, activation = 'relu')
15
16  #training the multilayer perceptron
17  model.fit(input_train, output_train)
18
19  #testing e evaluating the model with a prediction output
```

```
20      y_pred = model.predict(input_test)
21      print("Predicted:", y_pred,"\n Expected", output_test)
22      print("Accuracy of MLPClassifier : ", model.score(input_test,
          output_test))
```

Figure A.11 – MultiLayer Perceptron (A.12) metrics.

```
Predicted: [2 1 1 2 1 1 1 1 0 0 2 2 2 1 2 1 0 0 1 0 0 2 0 1 1 0 2 0 1 0 1 0 2 2 2 2 0 1]
 Expected: [2 1 1 2 1 1 1 1 0 0 2 2 2 1 2 1 0 0 1 0 0 2 0 1 1 0 2 0 1 0 1 0 2 2 2 2 0 1]
Accuracy of MLPClassifier :  1.0
```

Source: Author (2023).

A.0.12 Autoencoder

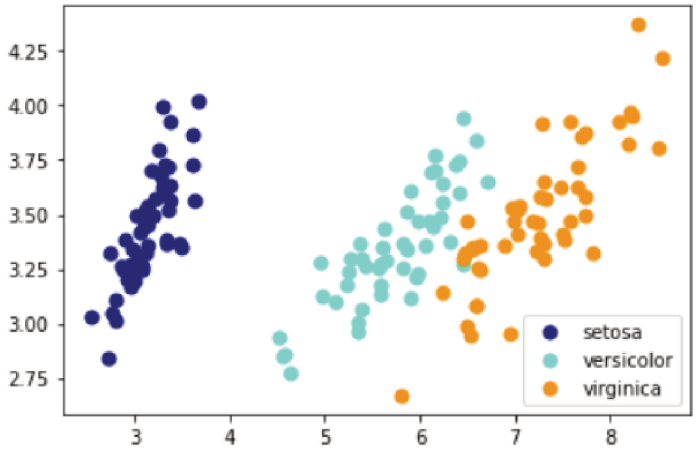
An application example is shown in the following code (Alg. A.13), where an AE is used to reconstruct the Iris dataset. The keras library was used to build the model and, with the TensorFlow library, train and evaluate the AE, importing the dataset with the SKLearn library. The network was structured by receiving input with 4 dimensions (4 attributes), encoding to 2 dimensions, and reconstructing back to 4 dimensions. The encoder and decoder were implemented with a linear activation function because of the nature of the dataset. It was trained with 1000 epochs with 75% of the dataset. The reconstruction made by the algorithm was plotted in another code. The result can be seen in Figure A.12 and can be compared with Figure 2.4, representing the original dataset. It is visible that the AE kept the setosa class separate from the others, with an acceptable reconstruction error (in terms of scale).

Algorithm A.13 – Autoencoder algorithm.

```
1       #importing of required libraries
2       import tensorflow
3       from tensorflow.keras.optimizers import Adam
4       import keras
5       from keras.models import Model, load_model
6       from keras.layers import Input, Dense
7       from sklearn import datasets
8       import matplotlib.pyplot as plt
9       %matplotlib inline
10
11      iris = datasets.load_iris()
12
```

```
13      #creating the AE, input with 4 dimensions (4 attributes),
         enconding to 2 dimensions, and reconstructing back to 4
         dimensions.
14      #Encoder e decoder with a linear activation function because
         of the nature of the dataset
15      input_dimension = iris.data.shape[1]
16      encoding_dimension = 2
17      input_iris = Input(shape=(input_dimension,))
18      encoded = Dense(encoding_dimension, activation='linear')(
         input_iris)
19      decoded = Dense(input_dimension, activation='linear')(encoded
         )
20      autoencoder = Model(input_iris, decoded)
21      autoencoder.compile(optimizer='adam', loss='mse')
22
23      #training the autoencoder
24      history = autoencoder.fit(iris.data, iris.data,
25                      epochs=1000,
26                      batch_size=32,
27                      shuffle=True,
28                      validation_split=0.25,
29                      verbose = 0)
30
31      #testing the encoded layer with the training input
32      encoder = Model(input_iris, encoded)
33      encoded_input = Input(shape=(encoding_dimension,))
34      decoder_layer = autoencoder.layers[-1]
35      decoder = Model(encoded_input, decoder_layer(encoded_input))
36      encoded_data = encoder.predict(iris.data)
```

Figure A.12 – Autoencoder (A.13) reconstruction.



Source: Author (2023).