Politecnico di Torino

Master Degree Thesis in Automotive Engineering

Data-Driven Approaches for the Design of Traction Electrical Motors

Supervisors:
Prof. Repetto Maurizio
Prof. Solimene Luigi
Prof. Ferrari Simone

Candidate:
Davoli Christian

Academic Year 2023/24

Abstract

In the last decades, the automotive industry has fostered the integration of electrical machines into vehicle's powertrain. The legislation regarding the environmental pollution is becoming more and more stringent, till to the point in which the classical internal combustion engines will not be compliant anymore. Consequently, Battery Electric Vehicles (BEVs), Hybrid Electric Vehicles (HEVs), and Plug-in Hybrid Electric Vehicles (PHEVs) are designed depending on the level of integration of the electrical machine in the powertrain.

This trend has increased the necessity to design effective electrical machines that are not only efficient but also cost-effective and powerful. Their design is therefore challenging and often requires extensive multi-physics considerations. For example, high Torque requires higher current, but this also leads to more complex cooling system. At the same time, electrical machines should be compact enough to fit inside the vehicles, ensuring excellent structural safety. The conflicting nature of these objectives adds complexity to the design process.

Due to this challenging environment, a multi-objective design approach is essential to achieve the best trade-off for specific applications. The main tool used is the Finite Element Analysis (FEA). By means of this, it is possible to evaluate electromagnetic, structural, and thermal performance of the design. However, even if this tool represents the state of art for multi-physics analysis, it is very computationally intensive. Therefore, it is limiting the exploring capabilities of the design space. In this context, Machine Learning can provide a large help by training regression models able to speed up the design process, allowing to use FEA only on the best promising configurations.

This thesis investigates the possibility of applying SVR and Neural Network to effectively predict torque and torque ripple, starting from an initial IPM (Internal Permanent Magnets) geometry defined by 8 input features. To train the models a dataset composed by 5000 samples have been created using FEA, in order to have the labels to link the geometry to the final prediction. The models are trained on a subset of the whole dataset to learn the statistics of the training set. Then this knowledge it is assessed on the test set, where the predictions can be compared with the real labels.

The final outcome is that both SVR and Neural Network can be effective tools in predicting the aforementioned outputs, providing, in most of the cases, a limited error range. Neural Network showed slightly superior performances with the main drawback of a longer training time compared to SVR.

Lastly, these models can be implemented in an optimization strategy, like differential evolution, to identify the non-dominated solutions belonging to the Pareto front. Given the slightly superior performances in terms of time, the SVR model has been implemented in the aforementioned differential evolution strategy. The current preliminary implementation needs more accurate validation but has already shown promising results in terms of speed of convergence, providing a good exploration of the design space.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and motivations

This first chapter provides an overview of the growing importance of electric mobility and highlights the key challenges in the design of electric machines. It also briefly describes the strategies that are adopted to address these problems. After this initial part, then the focus is moved toward the contribution of this thesis work, introducing the objectives and the impact that these can bring to the design of traction electrical motors.

1.1 Advancement in Electric Mobility

Electric Mobility, also known as e-Mobility, is one of the most crucial factors in the technological progress within the automotive industry. It encompasses the principles of using electric propulsion for different transportation means. This concept is representing one of the key pillar in addressing global sustainability challenges. In particular, sustainability is defined as the *"ability to meet the needs of the current generation without compromising the ability of future generations to meet their needs"* [1]. It is clear how sustainable development is thus involving social, economic, and environmental dimensions. In this scenario, e-Mobility plays a significant role as it gives the possibility to address the ecological challenges and promoting environmental sustainability by reducing pollution.

The growing awareness of environmental pollution is fostering the adoption of electric propulsion for a wide range of transportation means, like passengers' car, busses but also ships, ferries, and sea vessels. In the automotive field, and particularly with the passenger cars, in the last years many alternative propulsion solutions were introduced:

- Battery Electric Vehicles (BEVs): packed with batteries that store energy to power the electrical machines. They need external charging;
- Hybrid Electric Vehicles (HEVs): combine electrical machines with classical Internal Combustion Engine (ICE) to improve fuel consumption;
- Plug-in Hybrid Electric Vehicles (PHEVs): similar to HEVs but features larger battery that can be charged from an external source, allowing to extend the pure electric range.

The development of these alternatives is driven by the urgent need of controlling harmful emissions, such Noxious emissions (NO_x), mitigate Green House Gasses

(GHG) and to provide systems with an overall higher efficiency. These factors, along with rising pollution levels, contribute to climate change and affect directly human health:

- Noxious Emissions (NO_x): are responsible for contributing to air pollution and respiratory diseases. Their levels should be controlled especially at local level since they are directly in contact with human beings [2];
- Green House Gasses (GHG): particularly carbon dioxide (CO_2), are major contributors to climate change. Many times, the actions to effectively reduce carbon dioxide emissions are opposite to Noxious Emissions (NO_x);
- Energy efficiency: which reflects the necessity of optimizing resources usage, avoiding waste.

These high-level goals are not only ethical imperatives but are also becoming part of the continuously evolving vehicles homologation procedures. In particular, the European Union's stringent CO_2 emission targets aim to achieve carbon neutrality by 2050. While internal combustion engines have reached high levels of maturity, meeting these increasingly rigorous standards remains a significant challenge. Original equipment manufacturers (OEMs) are subjected to fleet average target, setting a policy of penalties and incentives for those who can respect them. Many typical approaches are investigated to be able to meet the regulations in short terms. Particularly, Fig. 1.1, reports the main pathway that has been currently followed.

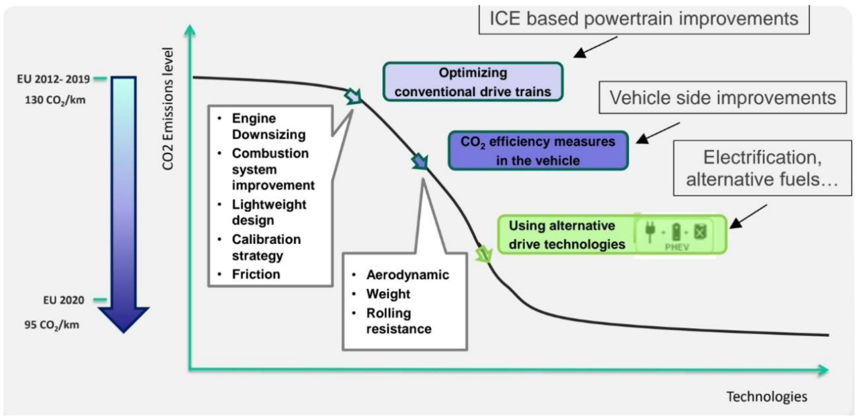

ICE based powertrain improvements
EU 2012- 2019
130 CO$_2$/km
EU 2020
95 CO$_2$/km
CO2 Emissions level
Optimizing conventional drive trains
Vehicle side improvements
• Engine Downsizing
• Combustion system improvement
• Lightweight design
• Calibration strategy
• Friction
CO$_2$ efficiency measures in the vehicle
Electrification, alternative fuels…
Using alternative drive technologies
PHEV
• Aerodynamic
• Weight
• Rolling resistance
Technologies

Figure 1.1: Example of typical approach to meet regulation targets [3]

The introduction of the electrical machines can effectively provide a solution to the different environmental problems, providing:

- zero CO_2 emissions at the local level (Tank to wheel - TWT);
- zero noxious emissions at the local level;
- very high energy efficiency;
- low acoustic emissions;
- energy recovery during braking and downhill to store energy inside the battery.

In addition to their environmental benefits, electric powertrains offer several advantages for the end users, such as:

- Acceleration, providing very high torque even at zero speed;
- Elasticity and smoothness, as they can cover the whole speed range without the necessity of having clutches and gearboxes;
- Higher energy efficiency;
- Regenerative braking during braking manoeuvres;
- Configurable performance options for drivers;
- Potentially lower maintenance cost, despite higher initial purchasing cost.

The rising interest in electric mobility is reflected in both consumer demand and the priorities of Original Equipment Manufacturers (OEMs), who view electric powertrains as a solution to stringent environmental regulations. This is evident from the global market trends, with nearly 14 million new electric cars registered in 2023, bringing the total number on the roads to 40 million, as illustrated in Fig. 1.2. In Europe, new electric car registrations reached nearly 3.2 million in 2023, increasing by almost 20% relative to 2022.

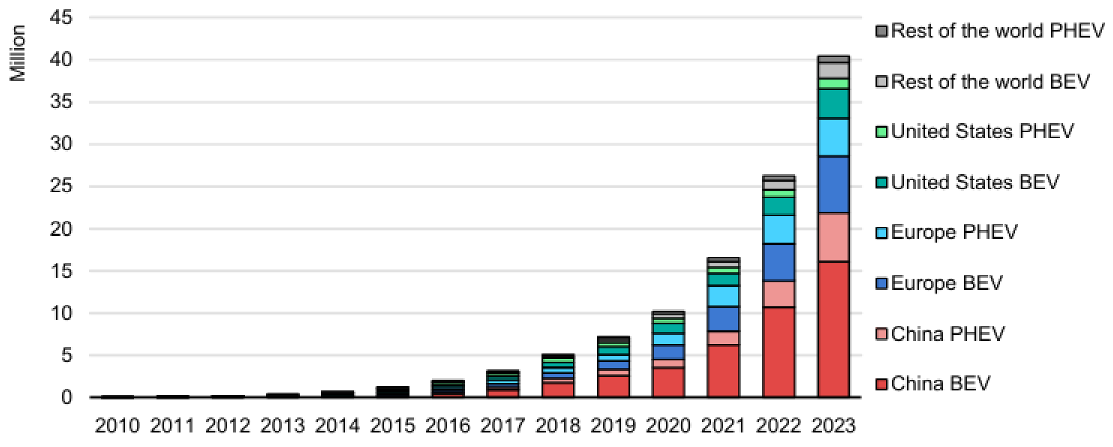


Figure 1.2: Global electric car stock [4]

1.2 Problem Statement

All these factors together are contributing to the necessity of developing electrical machines with higher performances and higher sustainability. The design and optimization of electrical machine is crucial and influences the rate of adoption of electrical solutions in the market. Electric machine's design is challenging as it is involving different physical domains such electromagnetic, structural, and thermal. All these aspects must be carefully evaluated at design stage and many times the final design requires a balance among all these factors. For examples, it is known that to achieve a larger torque, it is required a larger current, but this will lead also to higher losses and therefore to higher temperature. The design process can be defined as a multi-objective design problem. The optimal trade - off is limited by the computational capabilities of the design setup since all the configurations must be analysed with Finite Element Analysis (FEA).

1.2.1 Optimization Strategies

The design process of electrical machines for automotive applications is facing many constraints and challenges. As mentioned earlier, the primary goal is to maximize torque production while minimizing the cost, torque ripple, and weight of active parts. Additionally, the design must ensure structural integrity to withstand increasing rotor speeds, while maintaining thermal efficiency and reducing vibrations. The size of the machine is also critical, as it must fit within the space available in the vehicle. It is clear how this design must deal with the interaction of many physical domains that are very often leading to conflicting objectives. This is increasing the difficult of the design process, requiring experience by designers and high computational effort to explore the design space.
Given the complexity of these interdependent factors, multi-objective optimization algorithms (MOOA) have become popular tools for exploring the design space and identifying optimal configurations. However, evaluating each configuration through FEA is computationally expensive, often requiring numerous iterations before achieving satisfactory performance. This is translating in a process that requires many hours to find viable solutions.

1.3 Objectives of the thesis

This thesis intends to determine the extend in which ML can be used inside the design procedure of electrical machines and whether the trained models are able to give valuable insight when used for the optimization purpose identifying the best set possible. Specifically:

- Two Machine Learning models are trained and tested on a dataset obtained with FEA analysis. In particular, SVR and Neural network are compared to identify the best model both in terms of prediction accuracy but also in terms of training time;
- The SVR model is tested in a multi-objective optimization process, using Evolutionary Algorithms, like Differential Evolution. The goal is to assess the possibility of integrating Machine Learning models in the identification of the best configurations, aiming at maximizing torque while minimizing torque ripple. Again, a comparison with the Pareto Front obtained on the dataset is given, to determine the quality of the model.

These objectives are effectively belonging to Data - Driven Approaches, which is leveraging on the introduction of Machine Learning in the classical workflow.

1.3.1 Data - Driven Approaches

In the last year, Machine Learning has gained a lot of popularity, due to its learning and predicting complex behaviours capabilities. Many models are present in literature, each with its own strong point. During the development of the thesis, Support

Vector Machines (SVR) and Neural Network were adopted since they led to the best results.

In the context of design optimization, it is now clear how the design space exploration is limited by the computational resources that are needed by the FEA software to evaluate the main output given an input geometry. Data - Driven Approaches are of interest since they can be used to predict a certain output starting from a set of inputs, as reported in Fig. 1.3. The main advantage is that the time needed is much short compared to the classical FEA analysis. It is clear therefore, how with Machine Learning, it is possible to explore the design space faster and with less computational effort. However, it is important to understand that the complexity behind the electrical machine's design, involving multi-physical phenomena (i.e., electromagnetic, thermal, structural, or acoustic aspects), introduces significant non-linearity. Due to this, Machine Learning models do not achieve the same precision as FEA, thus associating a certain error with the predictions. However, a well tuned model can indeed give important insight about the analysed configuration, allowing to understand its performances. This can enhance the optimization process, as refinements with FEA can be applied only to promising configurations, resulting in considerable time savings.

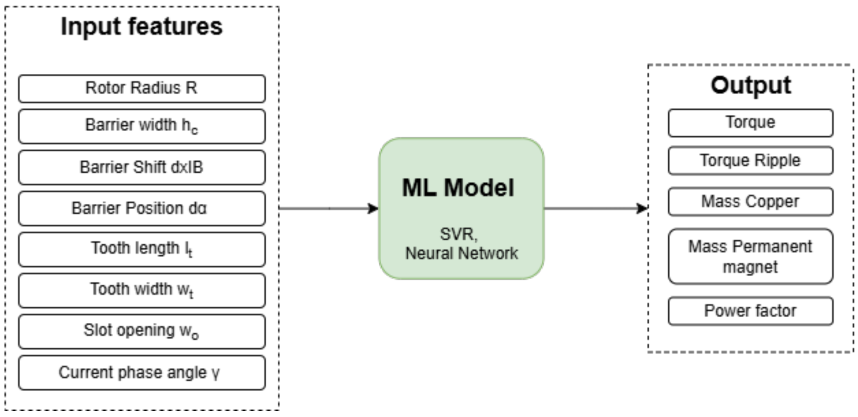


Figure 1.3: Data - Driven approach

1.4 Organization of the thesis

This part is dedicated to the description of the different chapters of this thesis work. In particular:

- Chapter 1 - *Introduction*: describes the necessity and the motivations behind this thesis work, providing a focus on sustainable development and on the major advantages that electric mobility can bring both to customers and to OEMs. It also highlights the key challenges in the design of electric motors and the role of multi-objective optimization strategies. Finally, it introduces the concept of Data-Driven approach, showing the thesis objectives.

- Chapter 2 - *eMotor*: examines the principal electrical machines adopted in the automotive field, with a focus on Internal Permanent Magnet motors (IPM).

Moreover, a brief overview on torque and torque ripple mechanism is offered. The last part is dedicated to the introduction of the case study (Honda Accord) with the description of the main structure specification;

- Chapter 3 - *SyR-e*: describes the framework and the procedure that has been followed to create the surrogate dataset relying on SyR-e and FEMM. In particular, the interaction among those two software allowed to evaluate 5000 different geometries, measuring the outputs of interest;

- Chapter 4 - *Machine Learning*: introduces the theory behind SVR and Neural Network models. After a first theoretic discussion, useful to provide an overview on the main hyperparameters to be optimized, the results obtained with the two models are reported. Finally a comparison among those two is examined;

- Chapter 5 - *Optimization framework*: after a brief overview about the different Evolutionary algorithms (EAs), the main focus is on Differential Evolution (DE). This strategy has been implemented in the thesis, to evaluate if the ML models, trained on the dataset, are able to offer valuable insight when dealing with optimization and design space exploration. To asses the quality of the results, the obtained *Pareto front* is compared with the one evaluated by SyR- e and FEMM. Specifically, Pymoo Performance Indicators are used as metrics for the comparison.

- Chapter 6 - *Conclusion*: provides a final overview of the results obtained during the development of the thesis, highlighting the main outcomes and identifying possible future works.

The overall research developed in this thesis follows the logical sequence summarized in Fig. 1.4.

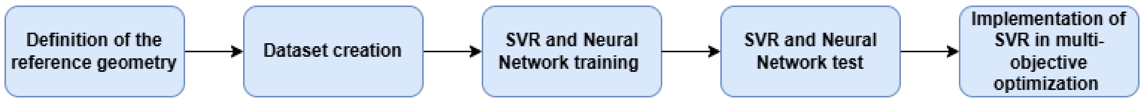

Definition of the reference geometry
Dataset creation
SVR and Neural Network training
SVR and Neural Network test
Implementation of SVR in multi-objective optimization

Figure 1.4: Logical Sequence

Chapter 2

Electric motors

The objective of this Chapter is to provide a brief overview on the principal typologies of electrical machines that are suitable for automotive applications. This is particular relevant as it would give the opportunity to introduce the main concept behind the torque production mechanisms, allowing therefore to understand what are the key geometry parameters that are influencing the performances of the machines. Lastly, the case study of this thesis is introduced.

2.1 Electric motors generalities

The automotive sector has seen in the last decade the introduction of electrical machines in the vehicles powertrain, providing valuable alternatives to the classical internal combustion engines (ICE). This shift is largely driven by the need to meet increasing torque demands, enhance efficiency, respect environmental targets, and improve the overall performance of vehicles. Researchers have devoted substantial effort to refining and optimizing various types of electric motors (eMotors) to address these challenges.

Key areas of focus include enhancing torque production capabilities, ensuring high efficiency, and maintaining reliability, all while designing motors that are compact enough to fit within the constrained spaces of modern vehicles.

Another significant consideration is cost, especially since many eMotors rely on permanent magnets, such as neodymium–iron–boron (NdFeB). These materials, while providing strong magnetic fields essential for efficient motor operation, are expensive and subject to supply chain vulnerabilities.

Both DC and AC solutions are present in literature [5]. Despite the easier control capabilities, the former are practically never use due to their lower efficiency, limited maximum rotor speed, brush wear and poor overload capabilities.

Fig. 2.1 offers a schematic overview about the different types of electrical machines that are used for traction application. The principal solution for automotive application is represented by AC machines. There are two big families:

- Asynchronous: they need a certain slip between the rotor and stator field to induce current in the rotor, which generates the required torque;
- Synchronous: the rotor and the stator's magnetic field are rotating at the synchronous speed ($\omega_0 = \frac{2*\pi*f}{pp}$ where f is the frequency of the stator voltage and current while pp is the number of poles pairs).

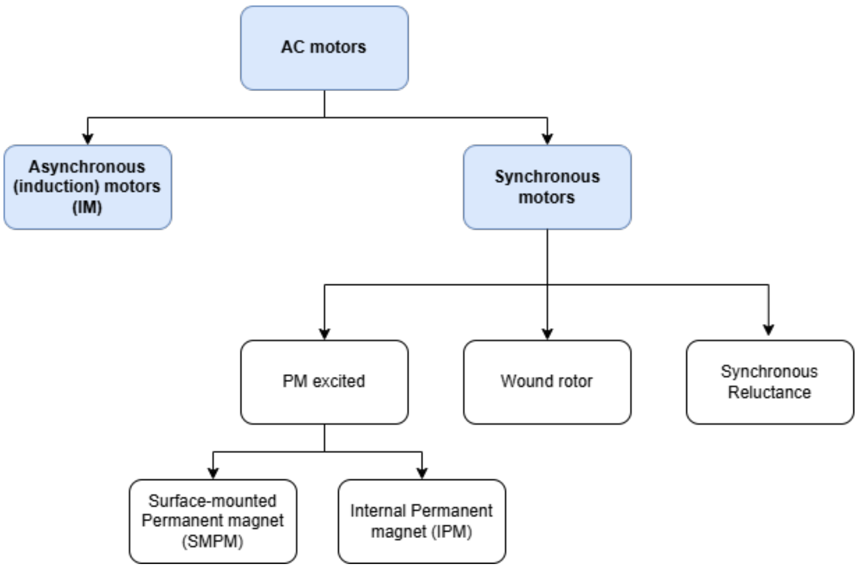
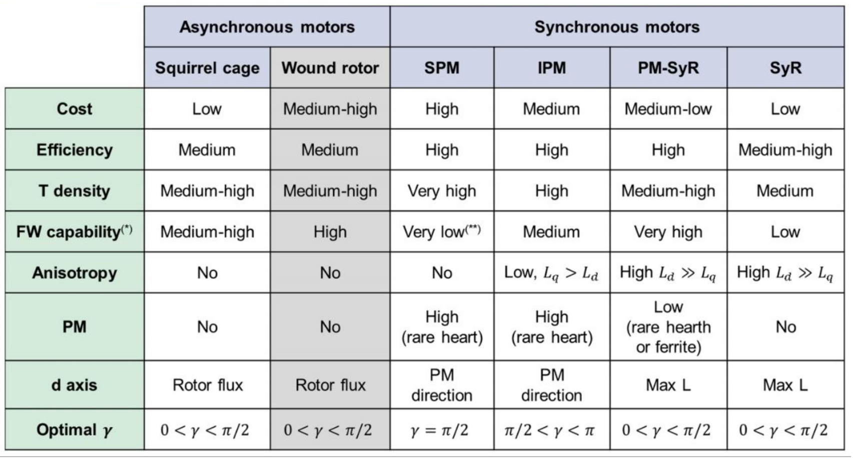

AC motors
Asynchronous (induction) motors (IM)
Synchronous motors
PM excited
Wound rotor
Synchronous Reluctance
Surface-mounted Permanent magnet (SMPM)
Internal Permanent magnet (IPM)

Figure 2.1: Schematic overview on the principal electrical machines solutions for automotive sector [3]

Synchronous electrical machines can be further categorized by the presence of Permanent Magnet (PM). Depending on their position it is possible to have:

- Surface Mounted Permanent Magnet (SMPM): permanent magnets are mounted on the surface of the rotor;
- Internal Permanent magnet (IPM): permanent magnets are buried inside of the rotor.

It is important to understand what are the main characteristics of each electrical machines and what are the advantages that a specific type could bring to the automotive sector. For this purpose, Fig. 2.2, is reported. Especially considering *cost*, *efficiency* and *torque density*, it is clear how IPM are a good choice for the automotive sector. Since the case study of this thesis is an Internal Permanent Magnet



	Asynchronous motors		Synchronous motors			
	Squirrel cage	Wound rotor	SPM	IPM	PM-SyR	SyR
Cost	Low	Medium-high	High	Medium	Medium-low	Low
Efficiency	Medium	Medium	High	High	High	Medium-high
T density	Medium-high	Medium-high	Very high	High	Medium-high	Medium
FW capability(*)	Medium-high	High	Very low(**)	Medium	Very high	Low
Anisotropy	No	No	No	Low, $L_q > L_d$	High $L_d \gg L_q$	High $L_d \gg L_q$
PM	No	No	High (rare heart)	High (rare heart)	Low (rare hearth or ferrite)	No
d axis	Rotor flux	Rotor flux	PM direction	PM direction	Max L	Max L
Optimal γ	$0 < \gamma < \pi/2$	$0 < \gamma < \pi/2$	$\gamma = \pi/2$	$\pi/2 < \gamma < \pi$	$0 < \gamma < \pi/2$	$0 < \gamma < \pi/2$

Figure 2.2: Summary of motor types and characteristics [6]

machine, the next section is dedicated to a brief description of the possible rotor geometries and torque production mechanism.

2.1.1 Internal Permanent Magnet (IPM)

For synchronous machines, independently from the stator winding (distributed or concentrated), the rotor design imposes the motor name and type. As already mentioned, these type of electrical machines are widely adopted in traction application, therefore the interest in optimizing their performances, achieving the best trade - off, is very high. Parameters like the number of magnets, their inclination and their position can affect the torque and torque ripple production. In literature, a wide number of possible rotor structures are described. In particular, focusing on IPM, two main families are present, as shown in Fig. 2.3:

- Spoke-type Internal PM motor: the magnets are placed radially;
- V-type IPM motor: the magnets are embedded in a "V" shape within the rotor.

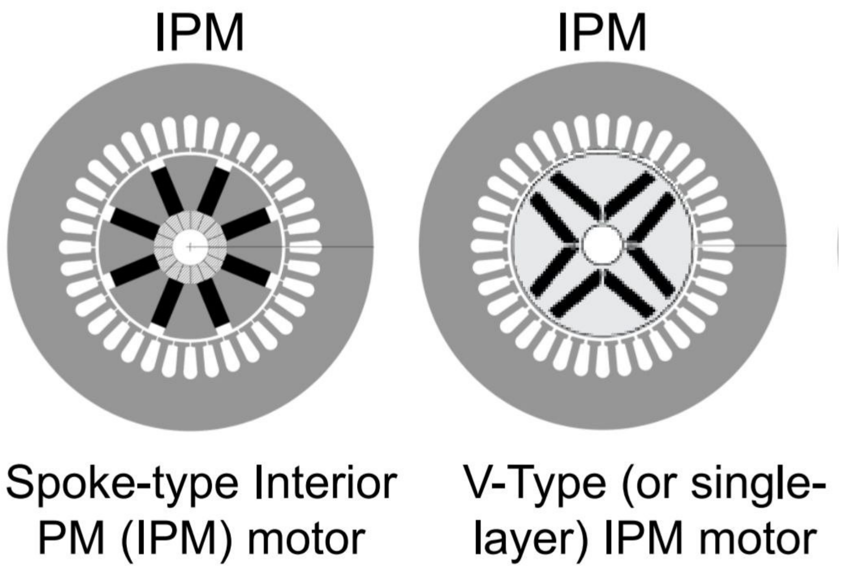

IPM
IPM
Spoke-type Interior PM (IPM) motor
V-Type (or single-layer) IPM motor

Figure 2.3: Examples of IPM machines [6]

The introduction of the permanent magnets inside the rotor, is allowing to exploit the concept of anisotropy. In particular, the inductance is not constant along the airgap. The anisotropy level of a given geometry is described by the Saliency ratio. It represents the ratio between the inductance of d-axis and the one on q-axis, as reported in Eq. 2.1.

$$\zeta = \frac{L_q}{L_d} \tag{2.1}$$

In case of IPM, saliency ratio is weak ($2 \div 3$) but not null [6].

Torque production mechanism The main outcome is that, besides the typology of rotor design, the torque produced by IPM is made by two components:

- Permanent magnet torque: each stator current produces a torque component in association with the permanent magnet field along the airgap (Lorentz law);

- Reluctance torque: due to anisotropy, the rotor tends to align with the direction of maximum inductance along the magnetic field imposed by the stator, to minimize the magnetic energy.

This concept can be resumed in Eq. 2.2, expressed in the *d*-*q* frame, that can be derived from the dynamic model of the analysed machine.

$$T = \frac{3}{2} \cdot p \left[\lambda_{PM} i_q - (L_q - L_d) i_d i_q \right] \tag{2.2}$$

It is clear how the two torque components have an opposite sign, therefore the optimization is fundamental to achieve the best trade - off performance wise.

Torque Ripple production mechanism torque ripple refers to a periodic oscillation of torque, that therefore is constrained between a maximum and a minimum value. A high torque ripple is undesired since it implies noise and also mechanical stresses that can damage the motor itself [6]. As shown in Eq. 2.2, torque ripple can be either affected by the linkage flux, by stator currents or by the *cogging* torque. In particular, the last one, is caused by the interaction between the permanent magnets on the rotor side and the stator slots [7] as well as the permeance variations during the magnet rotation. The result is the tendency of the rotor to stop in determined positions and it happens also with no current. It is therefore influenced by the rotation of the rotor. As reported in literature [8] multiple geometrical parameters are showing an influence on torque ripple production. In particular: stator slot opening, slot width, slot height, airgap length, stack length, pole numbers, magnet dimensions, magnet shape and position. It is important to underline that the relation among the torque ripple and these geometrical dimensions is highly not linear and therefore it is not possible to evaluate the a priori effect of any changes on the torque ripple value.

Iterative design and designer experience are crucial to achieve the best trade - off between those two competing objectives. In Chap. 3, a description of the different geometrical parameters used as design variable is given. Moreover, the influence on torque and torque ripple is discussed.

2.1.2 Case study - Honda Accord

After the initial discussion about IPM generalities, useful to understand the torque production mechanism and the main geometrical parameters, the case study can be introduced.
The reference IPM motor, with a published power rating of 124 kW, is belonging to the 2014 full hybrid Honda Accord. Thanks to the integration of the electrical machine, the power train can operate in:

- Full electric mode (motor EV drive): the power flows from the battery to the electrical machine that is connected to the wheels;

- Series electric mode (motor drive): the internal combustion engine is used in combination with a e-generator to fed a battery that provides power to the electrical machines;

- Engine mode (pure ICE drive): by means of a clutch, the ICE is directly connected to the wheels.

To better visualize these three operations mode, the possible energy routes are reported in Fig. 2.7.

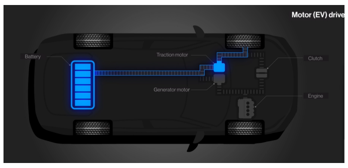


Figure 2.4: Full electric mode

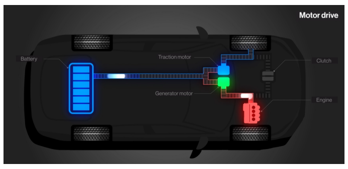


Figure 2.5: Series electric mode

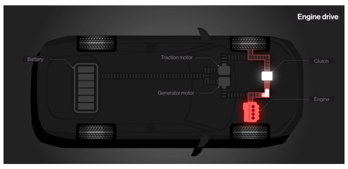


Figure 2.6: Engine mode

Figure 2.7: Honda Accord - Energy routes [9]

To proper implement this motor on SyR-e, the first task is to define the geometrical dimensions of the electrical machines. For this purpose CAD software has been used. In this way, the 2D drawing showed in Fig. 2.8 has been obtained. This has allowed to obtain all the necessary measures, ensuring to be as close as possible to the real geometry.

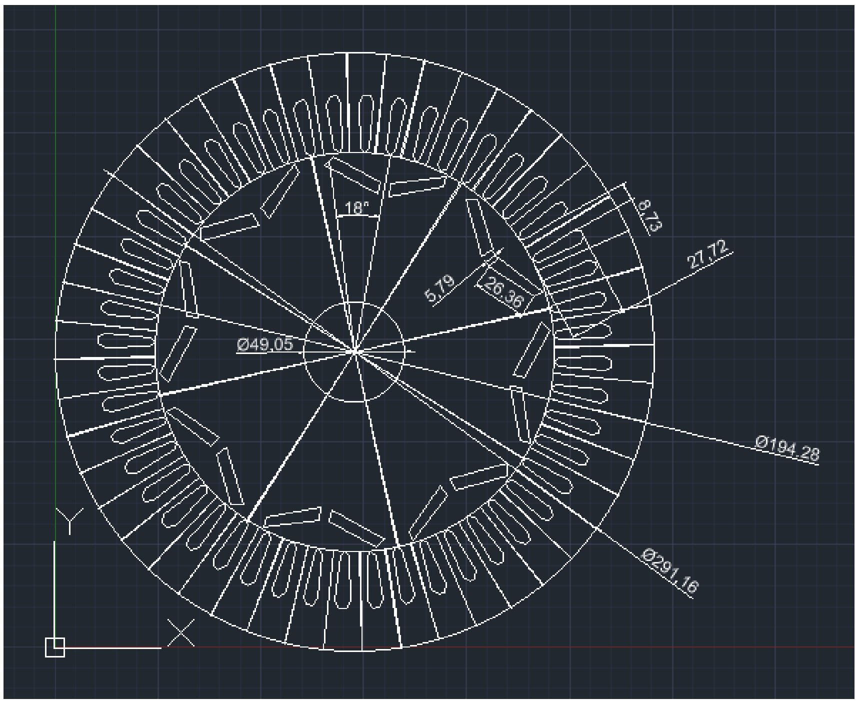


Figure 2.8: Honda Accord IPM machine CAD

Besides the geometrical dimensions, other important parameters for the simulation purposes are needed. In particular, table 2.1 reports the main motor structure specifications. Those quantities are obtained by looking at technical reports [10].

All these information will be later on combined in SyR-e. The final goal is to obtain the 2D geometry showed in Fig. 2.9. Due to its symmetry, only a portion is used. This allows to reduce the computational effort while maintaining solution quality.

Pole pairs	4
Number of slots	48
Number of barrier	1
Stator diameter	291,3 (mm)
Stack length	61.7 (mm)
Number of turns	20
Peak Thermal loading	45 ($\frac{kW}{m^2}$)

Table 2.1: Honda Accord 2014 structure specifications

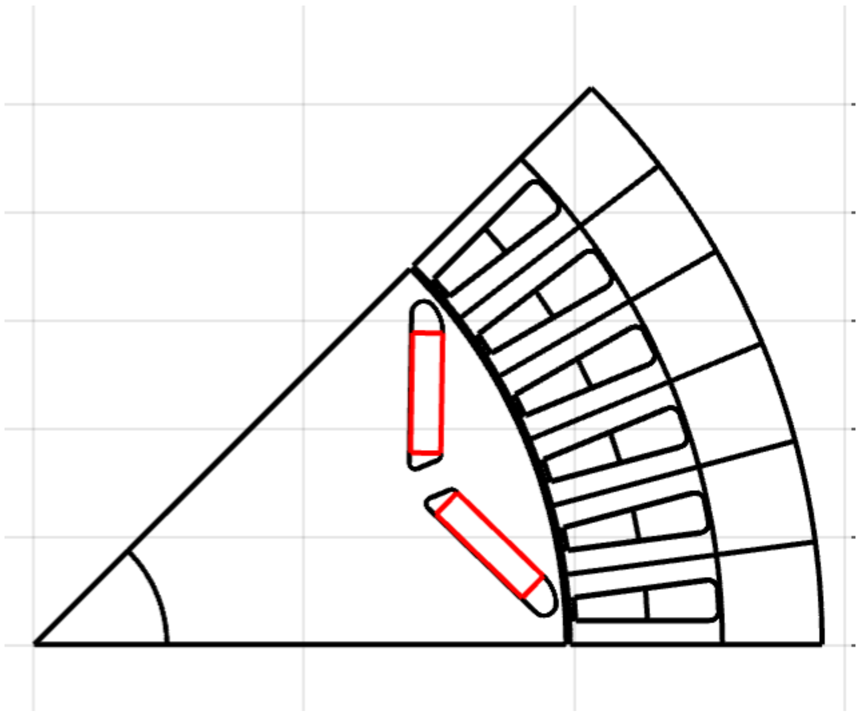


Figure 2.9: Final geometry

This geometry has been used as a reference to create the dataset, composed by 5000 elements, that will be used to train the Machine learning models. In particular, by means of Latin Hypercube Sampling, different geometries are sampled and evaluated with FEA in order to obtain the relevant output for the purpose of this thesis. On the contrary, structure specifications are kept constant for all the configurations. More details are given in Chap. 3.

Chapter 3

Parametric pre-processing of electric motor analysis: SyR-e

This Chapter is dedicated to the description of the logical sequence and the methodologies that have been used to create the *surrogate dataset* for Machine Learning model's training. After the definition of the Internal Permanent Magnet (IPM) geometry, discussed in Chap. 2, the open-source software SyR-e is used to perform the simulations.

SyR-e is the acronym of *Synchronous Reluctance – evolution* and it is a collection of codes developed in Matlab/Octave. It covers Synchronous Reluctance machine, PM-assisted SyR, Internal PM and Surface Mounted PM machine types [11]. It provides a tool for the design of the aforementioned machines based on Finite Element Analysis (FEA) and also multi-objective optimization algorithms. The operational capabilities of SyR-e are represented in Fig. 3.1. Matlab is used to realize a parametrized drawing of the machine as a *.fem* file that is analysed by FEMM, a software which can manage linear/nonlinear magneto static problems [12]. The results of the simulation are sent back to Matlab in order to evaluate the performances of the analysed machine.

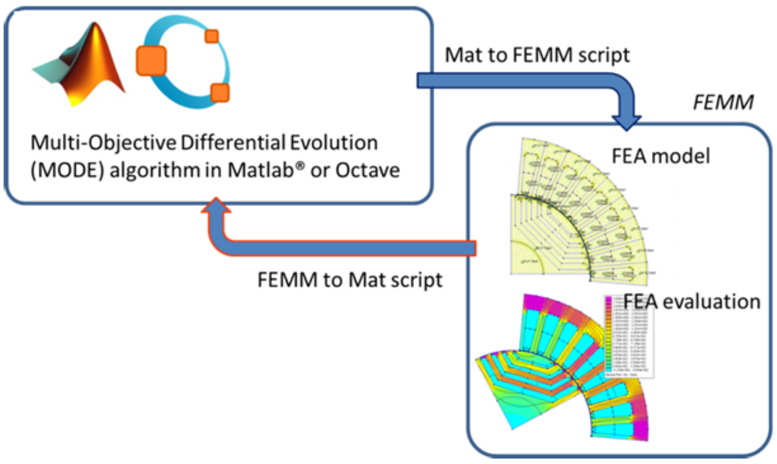

Multi-Objective Differential Evolution
(MODE) algorithm in Matlab® or Octave
Mat to FEMM script
FEMM
FEA model
FEMM to Mat script
FEA evaluation

Figure 3.1: SyR-e and FEMM data flow exchange [11]

For the development of this thesis SyR-e has been used to create the dataset for the Machine Learning training purpose. The goal is to realize a ML model that is able to predict within a certain accuracy the torque and torque ripple of a given IPM geometry, starting from the parametric description reported in the previous Chap. 2. By doing so, it is possible to use these models in the optimization, allowing to speed up the process, providing insights about the best parameter combinations able to satisfy different constraints and objectives. Finite Element Analysis can then be used then to validate the results. It is worth noting that, in Machine Learning literature, the term *feature* refers to the input variables used by the model for predictions. In this thesis, the geometrical variables are used to predict torque and torque ripple. For this reason, the terms *geometrical variables* and *features* are used as synonymous throughout the work. The same logic applies to *labels* and *outputs*. Once the parametric geometry of the electrical machines has been defined, the first operation is to decide the boundaries for each design variable. By sampling those values, many different configurations can be realized and for each of them FEA analysis can be conducted to evaluate the main output (torque, torque ripple, copper mass, permanent magnet mass, and power factor). Tab. 3.1 and Fig. 3.2 are reporting the variable's boundaries that have been used. These values have been chosen by trial-and-error process, ensuring to reach the minimum number of unfeasible configurations in the final dataset.

Variable	**Boundaries**
Rotor Radius r	[78 109] (mm)
Barrier width h_c	[0.3 0.45] (p.u.)
Barrier Shift $dxIB$	[-3 6] (mm)
Barrier Position $d\alpha$	[0.5 0.6] (pu)
Tooth length l_t	[15.2 32] (mm)
Tooth width w_t	[3.4 10] (mm)
Slot opening w_o	[0.1 0.4] (p.u.)
Current phase angle γ	[35 75] (°)

Table 3.1: Variables lower and upper boundaries

Fig. 3.3 shows the SyR-e optimization window with the aforementioned variables boundaries. The main elements in this interface are:

- The definition of the lower and upper boundaries for each design variables;
- The setup of the size of the dataset (5000 in this case). This value has been chosen as trade-off among the variability of the dataset and the computational time needed;
- The definition of the objectives and penalization limits.

For what regards the objectives and penalization limits, it must be considered that this interface has been primarily designed for the MODE (Multi-objective differential evolution) optimization, for which is required to define the objectives that are driving the evolution of the initial population. This is done to improve the convergence, since a penalty cost is paid when the quantities are becoming larger with respect the limits. If the limit is negative, the quantity is maximized (i.e. torque), while if the limit is positive, the quantity is minimized (i.e. torque ripple). Here, the goal is just to produce the dataset, without considering any penalization logic. Therefore,

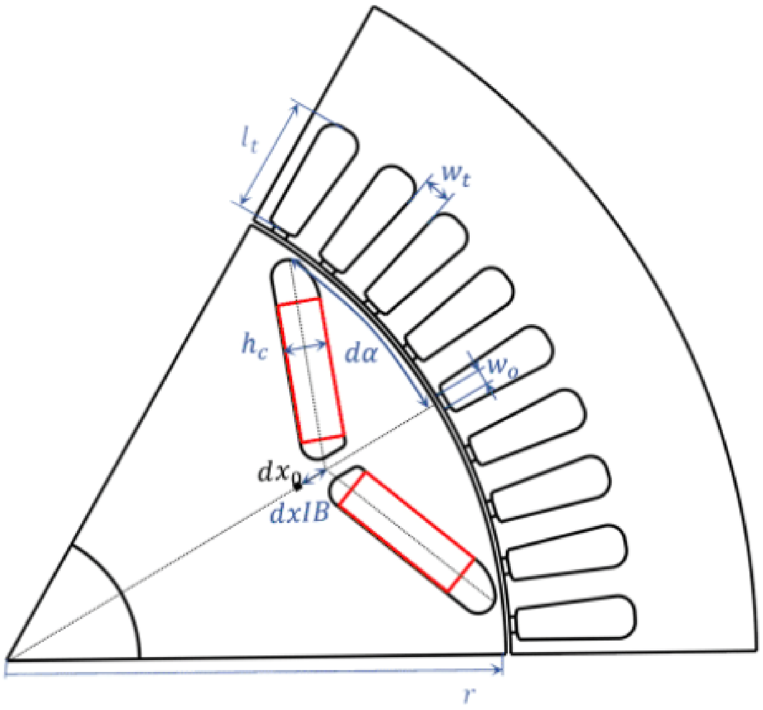
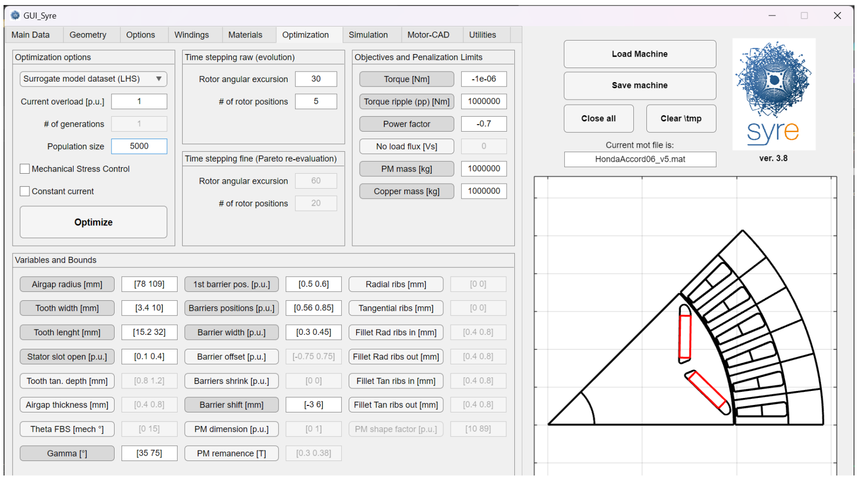


Figure 3.2: Variables representation

positive limits are set to be exceptionally large, while negative ones are low. This is producing a dataset without any particular attention to optimization. This is reasonable since the main purpose of the dataset is the training of the Machine Learning models.



Optimization options

Surrogate model dataset (LHS)

Current overload [p.u.]	1
# of generations	1
Population size	5000

☐ Mechanical Stress Control

☐ Constant current

Time stepping raw (evolution)

Rotor angular excursion	30
# of rotor positions	5

Time stepping fine (Pareto re-evaluation)

Rotor angular excursion	60
# of rotor positions	20

Objectives and Penalization Limits

Torque [Nm]	-1e-06
Torque ripple (pp) [Nm]	1000000
Power factor	-0.7
No load flux [Vs]	0
PM mass [kg]	1000000
Copper mass [kg]	1000000

Current mot file is:
HondaAccord06_v5.mat

Variables and Bounds

Airgap radius [mm]	[78 109]	1st barrier pos. [p.u.]	[0.5 0.6]	Radial ribs [mm]	[0 0]
Tooth width [mm]	[3.4 10]	Barriers positions [p.u.]	[0.56 0.85]	Tangential ribs [mm]	[0 0]
Tooth lenght [mm]	[15.2 32]	Barrier width [p.u.]	[0.3 0.45]	Fillet Rad ribs in [mm]	[0.4 0.8]
Stator slot open [p.u.]	[0.1 0.4]	Barrier offset [p.u.]	[-0.75 0.75]	Fillet Rad ribs out [mm]	[0.4 0.8]
Tooth tan. depth [mm]	[0.8 1.2]	Barriers shrink [p.u.]	[0 0]	Fillet Tan ribs in [mm]	[0.4 0.8]
Airgap thickness [mm]	[0.4 0.8]	Barrier shift [mm]	[-3 6]	Fillet Tan ribs out [mm]	[0.4 0.8]
Theta FBS [mech °]	[0 15]	PM dimension [p.u.]	[0 1]	PM shape factor [p.u.]	[10 89]
Gamma [°]	[35 75]	PM remanence [T]	[0.3 0.38]		

Figure 3.3: SyR-e optimization interface

After this setup, it is required to define how to sample the variables in the design space. This step is really important as the goodness of the final dataset, and therefore the accuracy of the results of the Machine Learning models, depends on this. SyR-e provides two different sampling strategies: *Sobol* and *Latin Hypercube* (LHS). During this thesis, Latin Hypercube has been used.

3.1 Latin Hypercube

To understand how the dataset has been created, it is necessary to recall some aspects about this sampling strategy. Latin Hypercube Sampling (LHS) [13] is an integration of stratified and random sampling. The main characteristics are:

- Stratified: process of dividing the initial population (each input variable) into homogeneous subgroups (also called *strata*) before sampling. For examples, in the case of this thesis, 5000 samples are considered. This means that the range of each variable (Tab.3.1) is divided into 5000 intervals, ensuring therefore to cover the whole variability of the population;

- Random Sampling: within the subgroups, a single value is chosen randomly, without considering any possible bias.

The combination of these two aspects is allowing for a deeper exploration of the input space, avoiding any possible concentration in a specific area. These concepts are clarified in Fig. 3.4. In this case, the initial population composed by 12 individuals, is divided into four subgroups. A random element is sampled, for each of them.

By extending this concept to the parametric geometry composed by the input features of table 3.1, it is possible to understand how the different configurations are created.

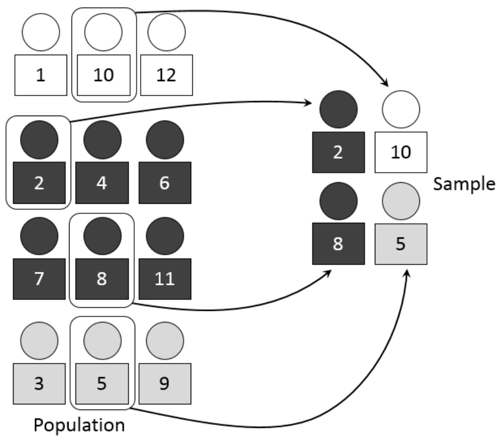


Figure 3.4: Stratified Random Sampling [13]

The effect of Latin Hypercube Sampling on the feature dataset are reported in Fig. 3.5. It is clear how this sampling method allowed to achieve a near-normal distribution for each input variable. The number of samples per bin is similar, proving the efficiency of LHS in sampling individuals from multi-dimensional distributions. This is beneficial as it ensures a comprehensive exploration of the input space.

It is evident that LHS is an effective method for dataset creation, as it leverages the variability of the initial variables while also incorporating a random component. Indeed, randomness is playing an important role in Machine Learning models, as it contributes to improve generalization and robustness, avoiding overfitting [14].

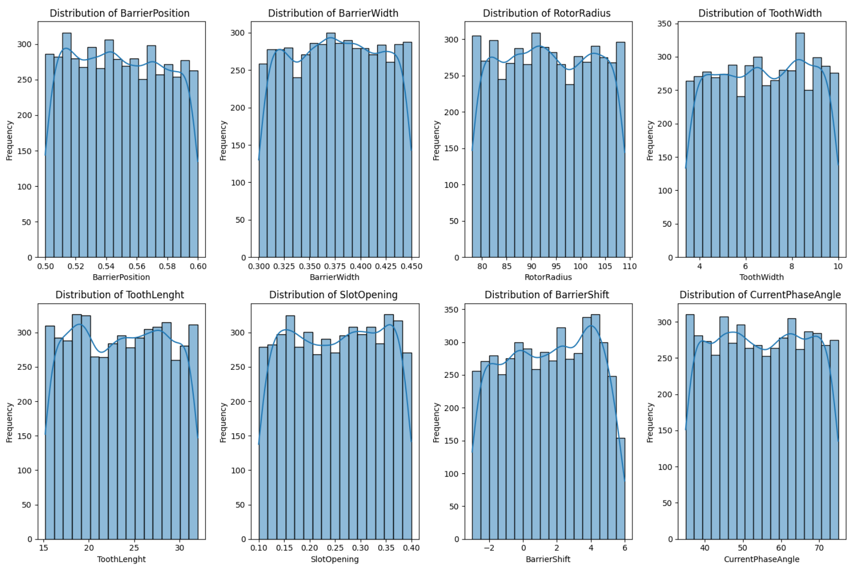


Figure 3.5: Input feature distribution

3.2 FEMM

As already mentioned, to evaluate the fitness (the outputs) of each configuration, it is required to use FEA to run the simulations. This step is crucial as it produces the dataset to train the ML models and the labels that are used at test time to evaluate their performances in terms of errors. The used software is FEMM [12], as it can be integrated in the SyR-e workflow.

In particular, the logical sequence used to obtain the label is reported in Fig. 3.6

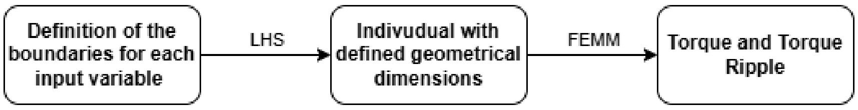
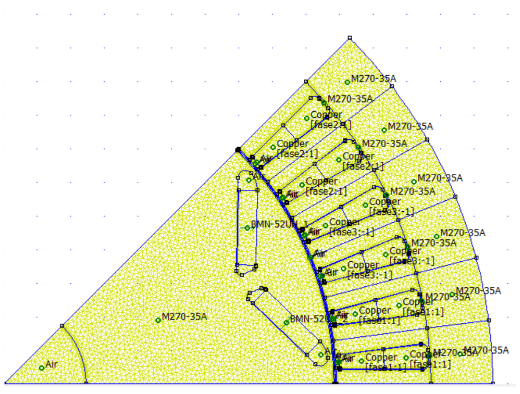

Definition of the boundaries for each input variable
LHS
Indivudual with defined geometrical dimensions
FEMM
Torque and Torque Ripple

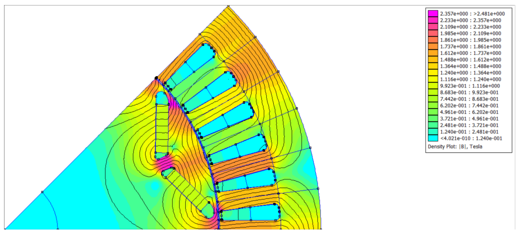
Figure 3.6: FEMM workflow integration

Thanks to Lua scripting [12], SyR-e and FEMM can exchange information. In particular, SyR-e is used to define a geometry by sampling with LHS the input features inside the boundaries. Once the geometry has been created, the Matlab script realizes the parametric drawing. Then FEMM is called to analyse it and to obtain the values of torque and torque ripple, by discretizing the geometry using a triangular mesh, as shown in Fig. 3.7a. In particular almost 27000 elements are used. To accurately determine the torque ripple, it is necessary to run simulations across multiple rotor's angular positions, as torque ripple is sensitive to its position relative to the stator. In particular, for the creation of the dataset, 5 rotor positions and 30° rotor angular excursion are used. Fig. 3.7b shows an example of results of FEMM analysis, where flux lines and magnetic flux density map are reported.

Fig. 3.8 reports the results of the single point simulation obtained for a fixed current phase angle and fixed current load. The torque output and the power factor, in function of the rotor position θ (expressed as electrical angle), are reported. The peak-to-peak distance, in the torque plot, is representing the torque ripple.



(a) FEMM mesh example



(b) FEMM results example

Figure 3.7: FEMM

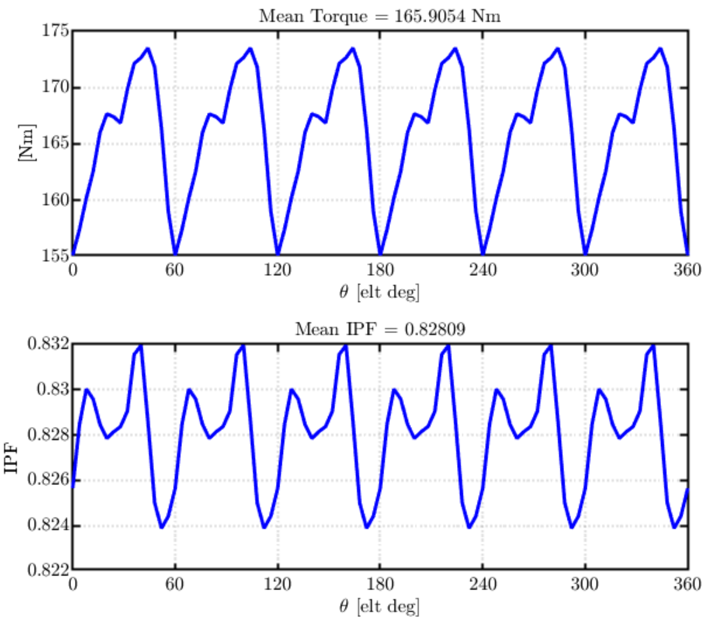
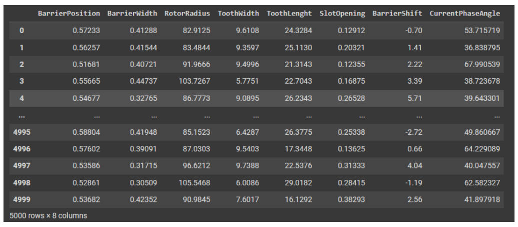


Figure 3.8: Torque plot vs θ

3.3 Dataset

After the evaluation of the 5000 sampled geometries, the fitness (torque, torque ripple, mass copper, mass of permanent magnet, and power factor) of each individual is calculated. The entire process lasted about 8 hours since multiple FEA simulations had to be performed. The reorganization of the elements is allowing to create a full dataset considering the inputs and the outputs of the simulations. To make an easier data manipulation, two datasets are considered: one for the input features and one for the output labels. Their structure is shown in Fig. 3.9. In particular, each row is corresponding to a certain geometrical configuration to which is associated the consistent fitness in the label dataset.

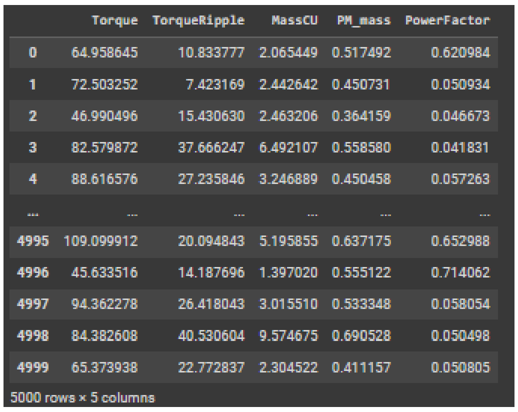
The main objective is to create a model able to predict the torque and the torque ripple. In Fig. 3.10 the Dataset is plotted in the Torque - Torque Ripple plane. It is possible to notice how with LHS sampling, the dataset is well distributed, without



	BarrierPosition	BarrierWidth	RotorRadius	ToothWidth	ToothLenght	SlotOpening	BarrierShift	CurrentPhaseAngle
0	0.57233	0.41288	82.9125	9.6108	24.3284	0.12912	-0.70	53.715719
1	0.56257	0.41544	83.4844	9.3597	25.1130	0.20321	1.41	36.838795
2	0.51681	0.40721	91.9666	9.4996	21.3143	0.12355	2.22	67.990539
3	0.55665	0.44737	103.7267	5.7751	22.7043	0.16875	3.39	38.723678
4	0.54677	0.32765	86.7773	9.0895	26.2343	0.26528	5.71	39.643301
...	...	...	...	...	...	...	...	...
4995	0.58804	0.41948	85.1523	6.4287	26.3775	0.25338	-2.72	49.860667
4996	0.57602	0.39091	87.0303	9.5403	17.3448	0.13625	0.66	64.229089
4997	0.53586	0.31715	96.6212	9.7388	22.5376	0.31333	4.04	40.047557
4998	0.52861	0.30509	105.5468	6.0086	29.0182	0.28415	-1.19	62.582327
4999	0.53682	0.42352	90.9845	7.6017	16.1292	0.38293	2.56	41.897918

5000 rows × 8 columns

(a) Features Dataset



	Torque	TorqueRipple	MassCU	PM_mass	PowerFactor
0	64.958645	10.833777	2.065449	0.517492	0.620984
1	72.503252	7.423169	2.442642	0.450731	0.050934
2	46.990496	15.430630	2.463206	0.364159	0.046673
3	82.579872	37.666247	6.492107	0.558580	0.041831
4	88.616576	27.235846	3.246889	0.450458	0.057263
...	...	...	...	...	...
4995	109.099912	20.094843	5.195855	0.637175	0.652988
4996	45.633516	14.187696	1.397020	0.555122	0.714062
4997	94.362278	26.418043	3.015510	0.533348	0.058054
4998	84.382608	40.530604	9.574675	0.690528	0.050498
4999	65.373938	22.772837	2.304522	0.411157	0.050805

5000 rows × 5 columns

(b) Labels Dataset

Figure 3.9: Structure of the Dataset

any extensive empty area. Moreover, the samples are distributed away from regions near the 0 Nm torque and 0 Nm torque ripple axes. This is a desired effect, since all the configurations are useful for the training purpose. Some more checks are needed to understand whether unfeasible configurations are present in the dataset. Therefore, the first analysis that has to be performed is the filtering of the initial dataset. A specific configuration is considered unfeasible when the mass of copper or the mass of permanent magnets are equal to 0. This can be done easily since those values are evaluated during the process (Fig. 3.11b). Fig. 3.11 reports the result of the analysis. In particular, it is possible to notice how all the configurations can be considered feasible with the previous hypothesis. Even if these constraints are quite simplistic, they are still valid, as the Machine Learning models should be able to offer valuable insight also with the far from optimal configurations. This is fundamental in a design optimization scenario. After these initial considerations, the filtered dataset is obtained, removing all the unfeasible geometries, and it can be used to train the ML regression models to make predictions.

At this point, it is possible to explore the statistics of each input variable. This is important as the Machine Learning models are mainly relying on statistics to capture the underlying pattern of the input data during training. Fig. 3.12 shows the mean, standard deviation, minimum and maximum value for for each feature. It is clear how the different parameters are distributed on different ranges and therefore it is necessary to perform some pre-processing to facilitate the regression task, avoiding any possible bias toward the bigger features (like rotor radius for instance).

3.3.1 Pearson Correlation Matrix

The last consideration takes into account the influence of each input variable on the specific output. It is important to remember that the dataset has been built starting from an initial geometry and using defined boundaries for the parametric geometrical input variables. It is therefore interesting to understand how a change in a feature can affect the output. Correlation coefficients are used to understand what the statistical relationship among two variables is. In particular, Pearson coefficient measures the strength and direction of the linear relationship. It is defined as the covariance of the variables divided by the product of their standard deviations as

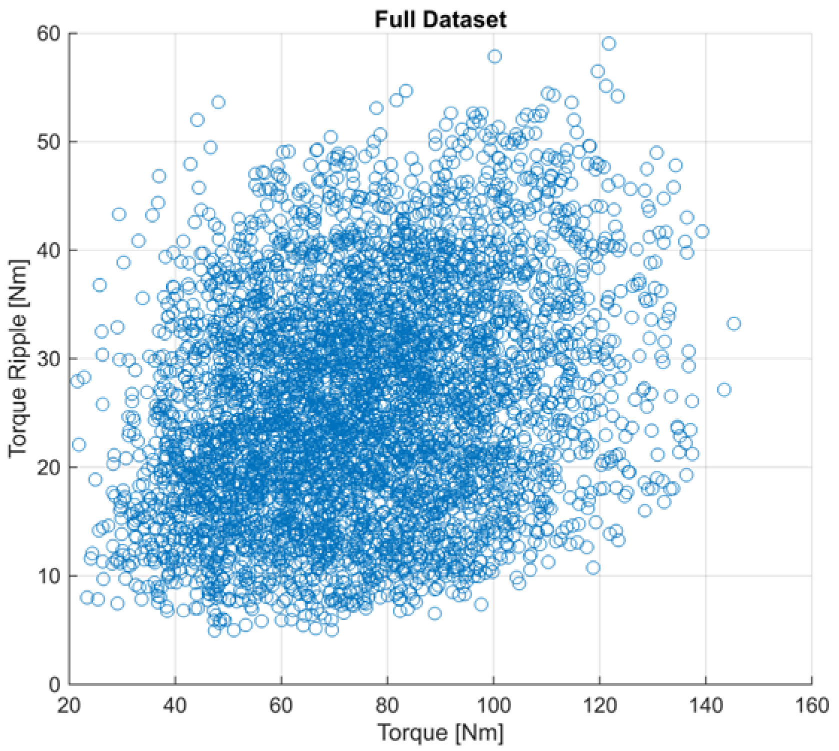
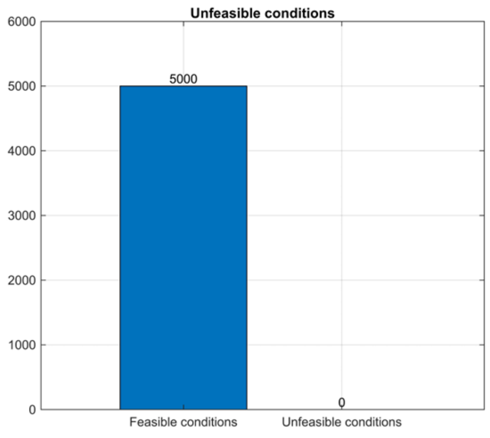
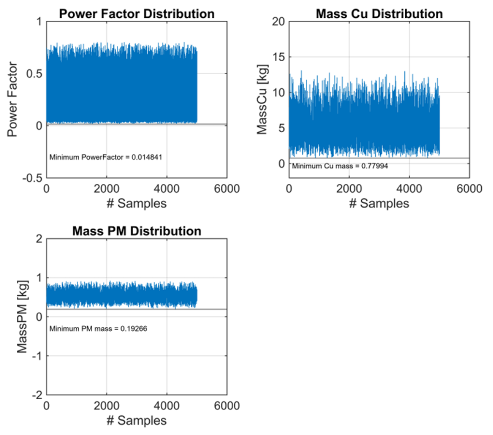


Figure 3.10: Dataset



(a) Unfeasible configurations



(b) Feasibility parameters distribution

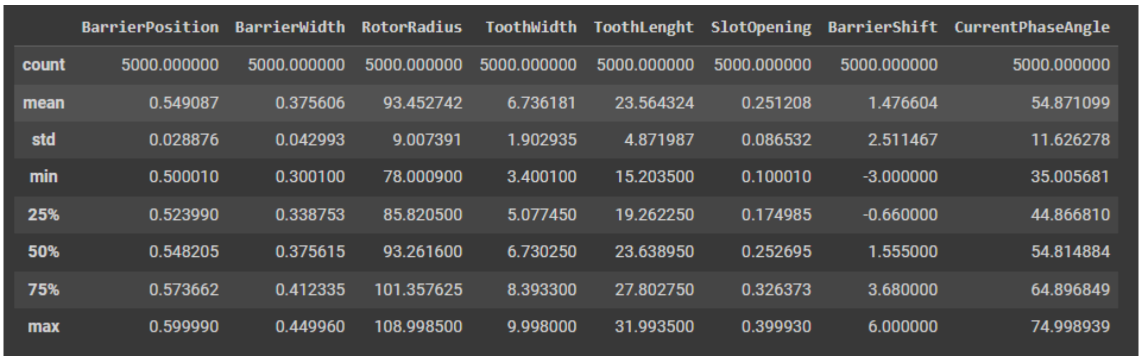
Figure 3.11: Unfeasible check on dataset

reported in Eq. 3.1 [15].

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} \tag{3.1}$$

By definition, correlation coefficients are bounded between [-1, 1], and as graphically explained in Fig. 3.13, if:

- ρ = -1, it means that the relation between the variables is perfectly linear and that Y decreases when X increases;



	BarrierPosition	BarrierWidth	RotorRadius	ToothWidth	ToothLenght	SlotOpening	BarrierShift	CurrentPhaseAngle
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000
mean	0.549087	0.375606	93.452742	6.736181	23.564324	0.251208	1.476604	54.871099
std	0.028876	0.042993	9.007391	1.902935	4.871987	0.086532	2.511467	11.626278
min	0.500010	0.300100	78.000900	3.400100	15.203500	0.100010	-3.000000	35.005681
25%	0.523990	0.338753	85.820500	5.077450	19.262250	0.174985	-0.660000	44.866810
50%	0.548205	0.375615	93.261600	6.730250	23.638950	0.252695	1.555000	54.814884
75%	0.573662	0.412335	101.357625	8.393300	27.802750	0.326373	3.680000	64.896849
max	0.599990	0.449960	108.998500	9.998000	31.993500	0.399930	6.000000	74.998939

Figure 3.12: Feature dataset statistics

- -1 $< \rho <$ 0, it means that the trend of the points has a negative slope;
- $\rho = 0$, it means that the points are linearly independent;
- $0 < \rho < 1$, it means that the trend of the points has a positive slope;
- $\rho = 1$, it means that the points are linearly dependent and that Y increases when X increases;

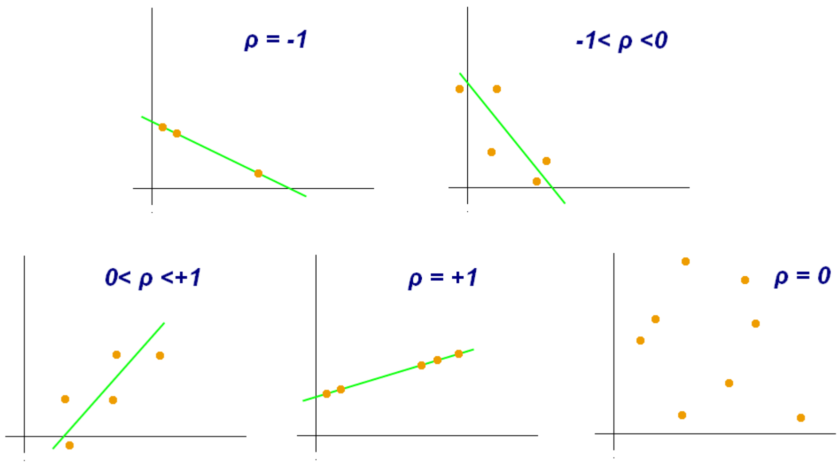
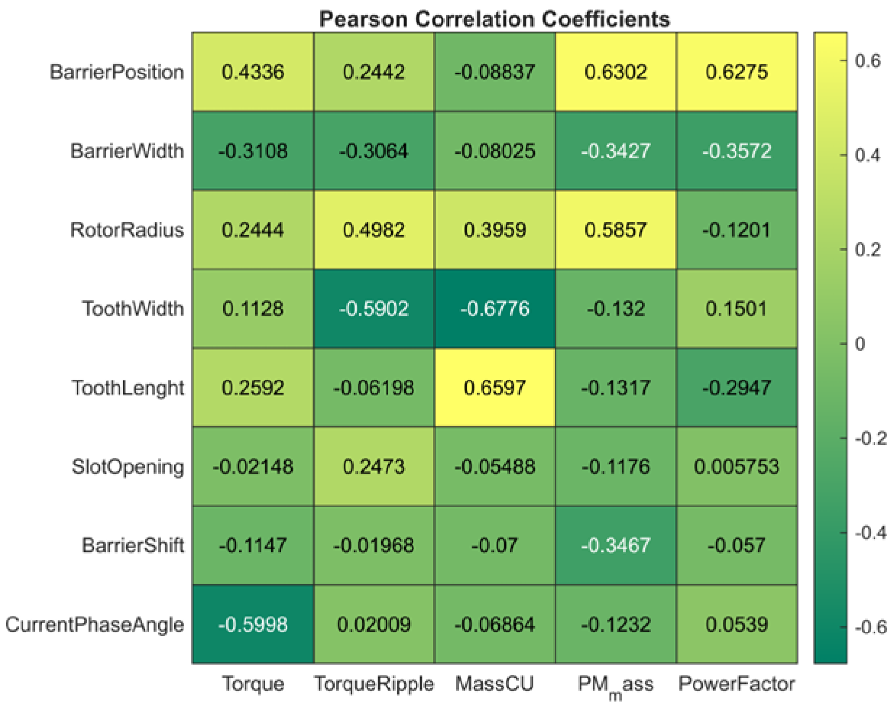


Figure 3.13: Correlation Coefficient interpretation [15]

Therefore, by examining the Pearson correlation matrix, it is possible to identify which input variables have the most significant impact on the output. This analysis helps to understand the underlying pattern in the data and can guide further optimization. To address this aspect, the Pearson correlation matrix is reported in Fig. 3.14 as heat-map. It is clear that:

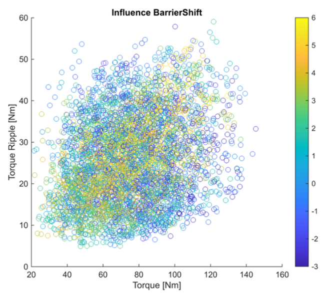
- Torque is mostly influenced by barrier width, current phase angle, rotor radius, and barrier position. On the contrary, it has almost no correlation with the slot opening;
- Torque ripple is mostly influenced by tooth width, rotor radius, and barrier width, while tooth length, barrier shift, and current phase angle exhibit little to no linear dependence.



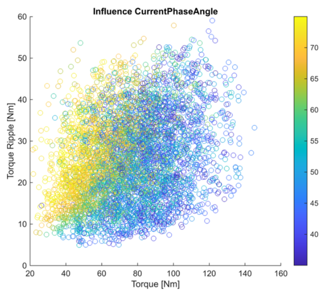
	Torque	TorqueRipple	MassCU	PM$_{m}$ass	PowerFactor
BarrierPosition	0.4336	0.2442	-0.08837	0.6302	0.6275
BarrierWidth	-0.3108	-0.3064	-0.08025	-0.3427	-0.3572
RotorRadius	0.2444	0.4982	0.3959	0.5857	-0.1201
ToothWidth	0.1128	-0.5902	-0.6776	-0.132	0.1501
ToothLenght	0.2592	-0.06198	0.6597	-0.1317	-0.2947
SlotOpening	-0.02148	0.2473	-0.05488	-0.1176	0.005753
BarrierShift	-0.1147	-0.01968	-0.07	-0.3467	-0.057
CurrentPhaseAngle	-0.5998	0.02009	-0.06864	-0.1232	0.0539

Figure 3.14: Pearson Correlation matrix

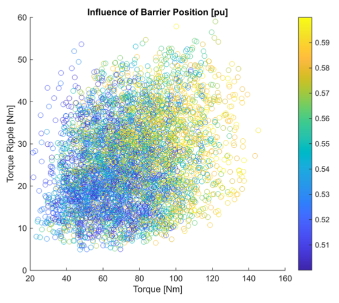
This correlation matrix is also useful to understand the complexity beneath the design of IPM. As it is possible to notice the correlation coefficients are bounded in [-0.6, 0.6] showing how the influence of the geometry is mainly nonlinear and requires FEA to carefully evaluate the performance of the machines. To further stress this concept, the influence of the parametric geometry variables is reported in the Torque - Torque Ripple plane in Fig. 3.15. In particular, a sensitivity analysis regarding the optimization parameter is obtained.



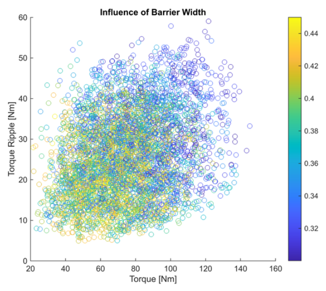
(a) Barrier Shift



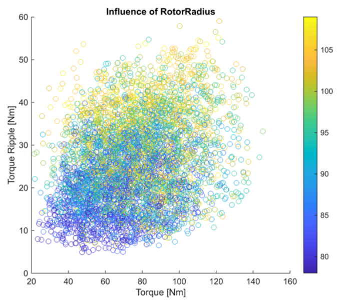
(b) Current Phase Angle



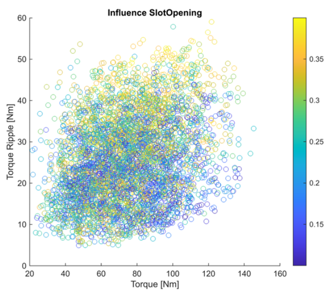
(c) Barrier Position (pu)



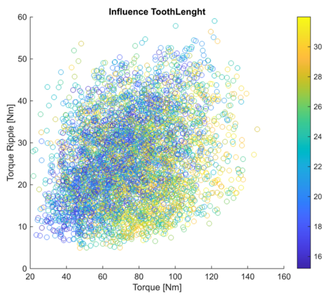
(d) Barrier Width



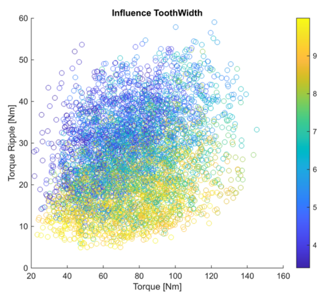
(e) Rotor Radius



(f) Slot Opening



(g) Tooth Length



(h) Tooth Width

Figure 3.15: Variable influence in Torque - Torque ripple plane

Chapter 4

Machine learning

This Chapter is dedicated to the description of the main Machine Learning models that have been used for the regression task. By introducing the theory behind them, it is possible to understand what is the effect of the hyperparameters on their performances, and moreover, how to optimize the traning dynamics to obtain the best results.

Machine learning is a branch of Artificial Intelligence (AI) that enables computer to learn from data. Relying on them, these algorithm are able to extract statistics and patterns that are used to build the knowledge of the model, mimicking the learning process of the human being.
It can be used to perform different tasks, as illustrated in Fig. 4.1:

- **Supervised Learning** which trains a model on known input and output data so that it can predict future outputs (i.e. *Classification* and *Regression*);
- **Unsupervised Learning** which finds hidden patterns or intrinsic structures in input data (i.e. *Clustering*).

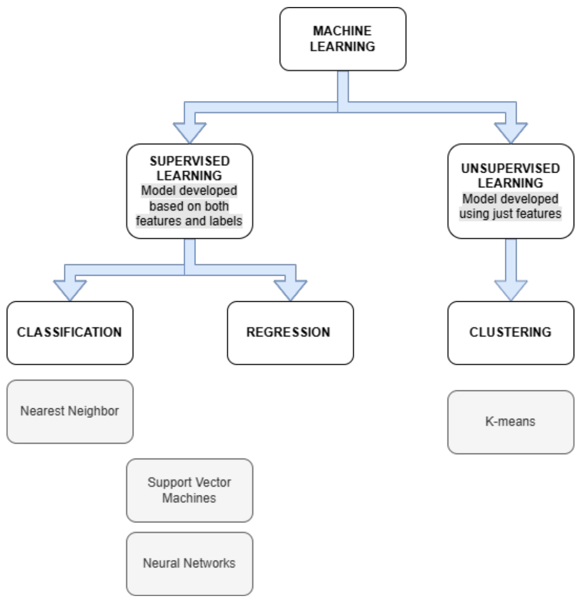

MACHINE LEARNING
SUPERVISED LEARNING Model developed based on both features and labels
UNSUPERVISED LEARNING Model developed using just features
CLASSIFICATION
REGRESSION
CLUSTERING
Nearest Neighbor
K-means
Support Vector Machines
Neural Networks

Figure 4.1: Machine Learning Tasks and relative methods. Adapted from [16]

The development of this thesis is based on Supervised Learning as the main goal is to train a model able to predict a continuous value, performing a regression task. Indeed, starting from the design variables used to define the parametric geometry of the IPM, it is required to identify some model able to produce accurate output results in terms of torque and torque ripple. Being a Supervised task, it is important to have both the features (the input data, i.e. the geometry of the IPM motor) and the labels (the output data, i.e. torque and torque ripple).

4.1 Regression

To understand how the models are working, it is worth to recall some aspects about regression as this would give the possibility to highlight what are the main hyper-parameters to tune to achieve the best performances out of the models.

Regression is a statistical method used to establish the relationship among a dependent variable (*label*, referring to AI terminology) and one or more independent variables (*features*). The primary goals is to derive the value of the output starting from the input values. Regression can be used to:

- predict the value of the dependent value starting from the independent ones;
- understand the strength of the dependence of the dependent value starting from the independent ones;
- identify and take important decisions.

For the development of this thesis, the main goal of the regression models is to be able to predict torque and torque ripple, starting from the input variables describing the geometry of the IPM motor.
Depending on the collected data, many regression models can be used as reported in literature [17]:

- Simple regression model;
- SVR;
- Neural Network.

The two techniques used in this thesis are the *Support Vector Regression (SVR)* and *Neural Networks*. More details about their working principles and the evaluation of their effectiveness are given in the dedicated Sections 4.3 4.4.

4.1.1 Regression metrics

To actually assess the quality of the regression model, different metrics can be used to evaluate the error between the predicted value and the real value. Each metric highlights different aspects on the performance of the model, giving insight about how well it is capturing the underlying relationships in the data. The most adopted metrics are now discussed and will be used at test time in the experimental results Sec. 4.3 4.4.

Mean Square Error - MSE Mean Squared Error is used to measure the average of the squared distances between predicted and real values as shown in equation 4.1.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4.1}$$

Its main characteristics are:

- it uses the mean to keep it independent from the the dataset size;
- due to the square operator, it penalizes more the larger errors, making therefore it more sensible to outliers;
- it is beneficial to use as loss function if large errors are particularly undesirable.

Mean Absolute Error - MAE It corresponds to the average magnitude of errors in a set of predictions, without considering their direction (overestimation or underestimation). The mathematical formulation is given in Eq. 4.2.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4.2}$$

The main characteristics are:

- it uses the mean to keep it independent from the the dataset size;
- all the errors are treated equally, since it is a linear relationship. This means that it is less sensitive to outliers;
- it gives direct control on the error related to the prediction.

In Fig. 4.2 is reported the main difference between MAE and MSE. The green line is representing the expected value (constant in this case), the red line is the predicted value. It is possible to notice how the MAE line has a constant slope regardless of whether it's an overestimate or an underestimate while the MSE increases faster, due to the squared operator, as the predicted value deviates more from the expected value, highlighting the influence of outliers on it.

Coefficient of determination R-squared R^2, also known as the coefficient of determination, measures the proportion of variance in the dependent variable that is predictable from the independent variables [19]. The mathematical description is reported in Eq. 4.3.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{4.3}$$

In particular:

- $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ represents the sum of squares of residuals (difference between the real value and the predicted one);
- $\sum_{i=1}^{n}(y_i - \bar{y})^2$ represents the sum of the squares (difference between the real value and the average value).

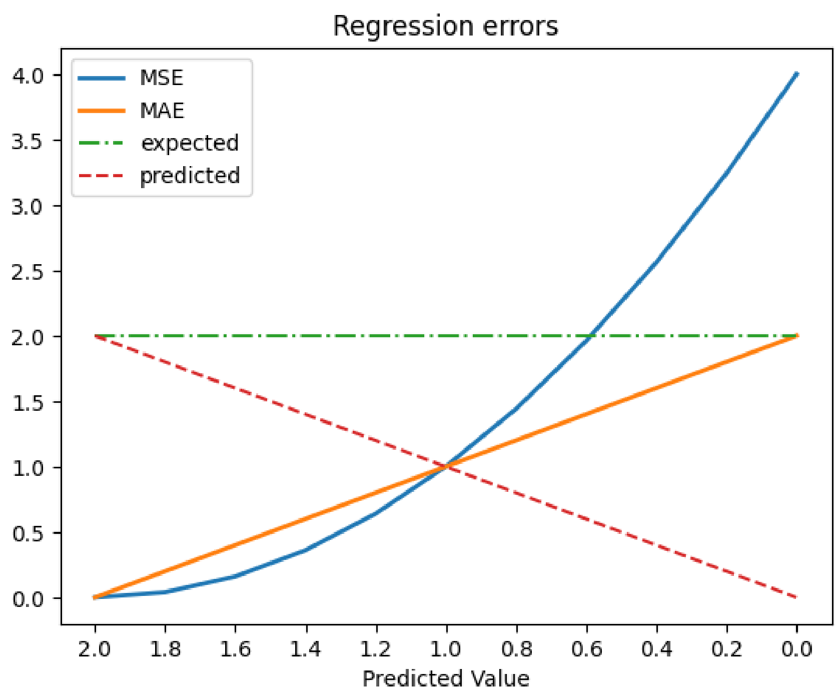


Figure 4.2: Comparison between MSE and MAE. Adapted from [18]

By definition, it is bounded between [0, 1]. This means that:

- If the prediction perfectly matches the real value $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = 0$, resulting in $R^2 = 1$;
- If the prediction is always close to $\bar{y}$, $R^2 = 0$.

For the purpose of the thesis, higher values of R^2 are desired since they are associated with better predictions and lower errors. It is particularly convenient to use this metric as it gives a direct measure of the goodness of fit of the model.

4.2 Training Dynamics

This section is dedicated to the aspects concerning the training dynamics of the implemented model. It is worth noticing that there are many parameters and methodologies to take into account when dealing with Machine Learning. The overall performances and the computational time are influenced by their choice. The goal of Supervised Machine Learning models is to extrapolate the statistics of the data and, based on this, make predictions.
In this thesis, different mechanisms are considered:

- Train-validation-test split;
- Pre-processing of the dataset;
- Overfitting;
- Hyperparameter optimization: Grid Search and Random Search;

- Cross-validation;
- Validation curve;
- Early stopping.

More details about their implementation are given in the relative Sec. 4.3 4.4.

4.2.1 Train - Validation - Test split

When dealing with Machine Learning models, it is important to develop the ability to generalize the unseen data. Given a new input, the model should be able to produce a new output within a certain accuracy. Since the model is learning on data, it is fundamental to measure its performances on a subset of the initial dataset, to avoid any possible bias at test set that may lead to overfitting, and consequently poor results.
This procedure, therefore, involves taking the initial dataset and split it to obtain different subsets. In particular, depending on the models, there are:

- Train Subset: used for the training phase to actually *learn* from data;
- Validation Subset: used for hyperparameters tuning to avoid overfitting;
- Test Subset: used to evaluate the performances and to understand the generalization capabilities of the trained model.

Fig. 4.3 gives a visual representation of the split. This is particularly useful when the size of the initial dataset is sufficiently large, allowing to have subset that are big enough and still representative of the original domain.
Moreover, splitting the dataset into subsets allows also to reduce the computational effort required to train the models, achieving a trade-off among the accuracy and the time needed to accomplish the task. In literature, different split ratios are used. The most common are:

- 60-20-20: 60% training, 20% validation, 20% test
- 70-15-15: 70% training, 15% validation, 15% test
- 80-10-10: 80% training, 10% validation, 10% test

Those are suggested ratios, but depending on the initial dataset, also other combinations may give better results. In this thesis, train - validation - test split has been performed using scikit-learn Python Machine Learning library implementing the function train_test_split [20].

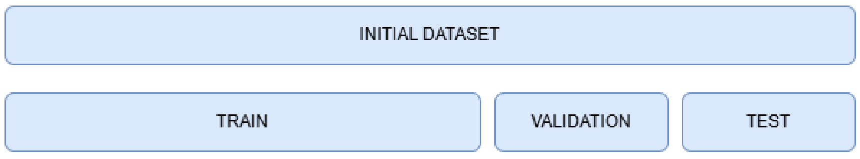
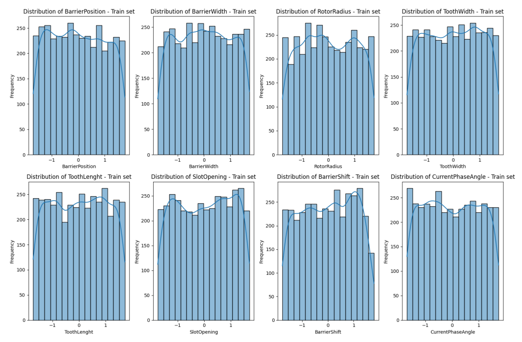


Figure 4.3: Train - Validation - Test split representation

4.2.2 Pre-processing of the dataset

As already discussed, the inputs features of the surrogate dataset are distributed on different ranges (Fig. 3.12). This may be a problem for regression task, since the results may be biased by the larger variables. The risk is to have a very unstable model that may perform poorly. To overcome this problem, it is important to scale the input features. Many scaling techniques are present in literature like *StandardScaler*, *MinMaxScaler*, *MaxAbsScaler*. The aim of these is to re-scale the input features in order to have similar characteristics, resulting in a simpler model. For this thesis work, *StandardScaler* has been used. It uses the statistics of the training set (both mean values and standard deviation) to remove the mean and scale the dataset to unit variance. It is important to use just the training subset characteristics to avoid any additional help at prediction time on the test subset. The effects of the scaling are reported in Fig. 4.4.


(a) Train Subset

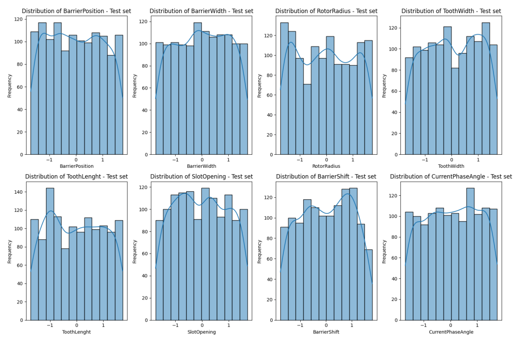

(b) Test Subset

Figure 4.4: Scaling of the dataset

4.2.3 Overfitting

Overfitting occurs when the predictor has excellent performances on the training set, but it poorly performs on the test set [21]. To address the concept, Fig. 4.5 is used. Let's assume to have a certain distribution of points. By means of the Machine Learning model, starting from the *Predictor variable*, the goal is to predict the *Output Variable*. Three main scenarios can occur:

- Underfit: the model is too simple and it is not able to capture the full complexity of the data, resulting in an under-performing model;
- Optimal: the model achieves the best trade-off among the complexity and the accuracy;
- Overfit: the model is too complex and somehow *learns by heart* the data, loosing the ability to generalize the unseen data. This means perfect fit at training time, but very poor accuracy at test time.

In Fig. 4.6 is shown the relation among the *capacity* (i.e. the complexity of the model) and the errors. The two curves represented are the training error and the generalization error (i.e. the test error). It is possible to notice how:

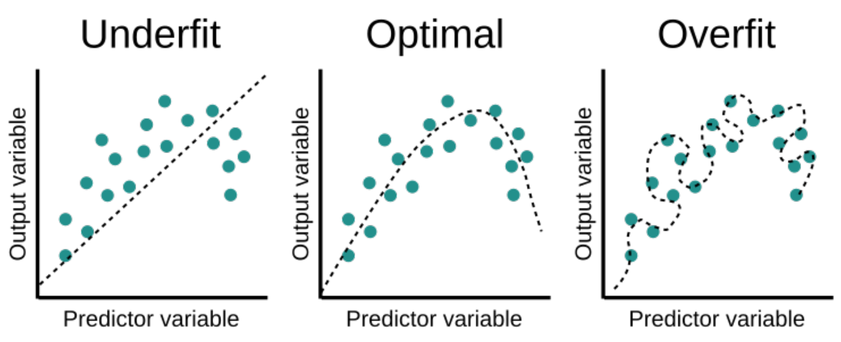


Figure 4.5: Three scenarios [22]

- A low-capacity model is producing high training and test error, meaning that the model is under-performing;
- A very high-capacity model is doing almost perfectly on train error, but on the test set the error is increasing, meaning that the generalization capability of the model is lacking;
- The optimal capacity is producing a comparable error at train and test time, providing the best performances.

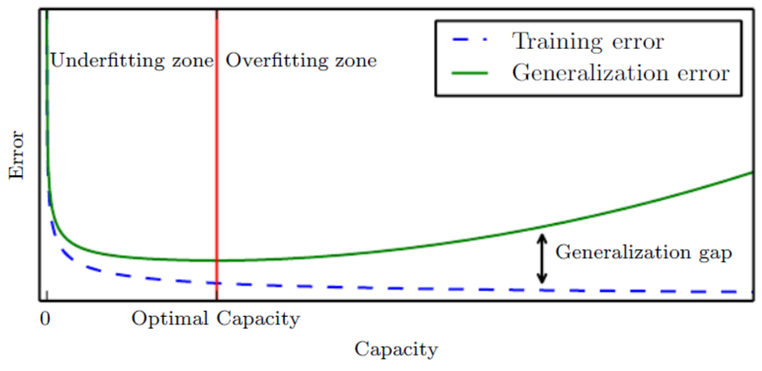


Figure 4.6: Error vs Capacity [23]

The best results are provided by the right model with a suitable complexity.

4.2.4 Hyperparameter optimization: Grid Search and Random Search

Hyperparameter can be defined as manually tuned parameters used to guide the learning process of the model. Those are different with respect internal parameters that, on the contrary, are learned automatically during training process. The model's performances are strictly related to hyperparameters. It is therefore especially important to optimize them as much as possible to obtain satisfactory results. In general, the effect of a specific hyperparameter on the model of interest is known, but defining the best combination of them is still challenging and depend on the specific dataset. Many studies proposed different methodologies to explore the search space [24] aiming at minimizing the error of the models. The search space is defined as the n-dimensional space in which each hyperparameter is representing a different dimension limited by the boundary, representing the feasible range or set of values that the hyperparameter can take. At the end of the process, the best combination of hyperparameter is found. In particular, different algorithm can be used:

- Grid Search: used to evaluate every possible combination of the hyperparameter, it is computationally expensive (especially when evaluating very poor performance combinations) but it allows to find the best combination;
- Random Search: used to evaluate n random combinations inside the search space. It is generally faster, but it does not ensure to find the optimal combination.

Fig. 4.7 [25], shows how the two algorithms are working inside a 2D search space. While Grid Search provides an even coverage of the search space, it explores only a small portion of the subspace defined by the single hyperparameter (i.e. on the x-axis). On the contrary, Random Search, allows a better investigation of the subspace where the single hyperparameter may give substantial changes at minimizing the error. Following this concept, in the development of this thesis, Random Search has been used as primary evaluation to understand which is the area of major interest. Once this is defined, Grid Search is used on the smaller search space to refine the results, obtaining the best combination within a suitable time. Both algorithms

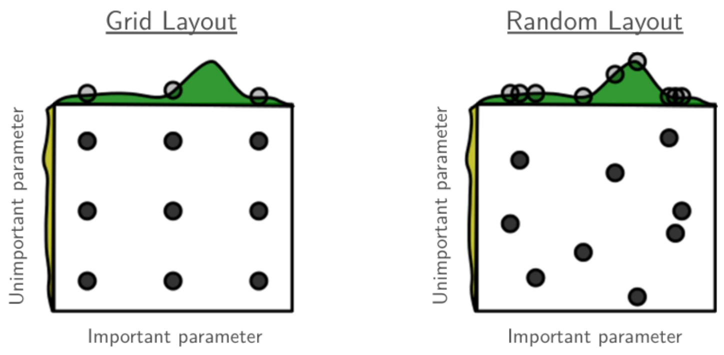


Figure 4.7: Grid Search and Random Search comparison in search space [14]

have been implemented using the scikit-learn Python libraries [26] and [27].

4.2.5 Cross-Validation

During training, dealing with hyperparameter optimization, it is crucial to find the optimal combination but at the same time to avoid overfitting, promoting a model able to generalize well at test time. It is essential to implement criteria to mitigate this risk, thus producing a robust model. In the context of Supervised models, like *SVR*, this risk can be reduced without the explicit need for a validation set, which could diminish the size of the training set, especially with small datasets. Numerous techniques for cross-validation exist in the literature.
For the development of this thesis, *k-fold cross - validation* from scikit-learn library has been used [28]. The working principle is shown in Fig. 4.8. Let's assume that

the dataset has been split into training data and test data. For this example, $k = 5$ is used. This means that the training set is split into 5 smaller subsets (called *folds)* and:

- The model is trained using $k\text{-}1 = 4$ folds
- The 5th fold is used as validation set to measure the performance of the model

This procedure is repeated for k *times* and at the end the performances are the average of the score obtained in the loop. By doing this, it is possible to avoid the definition of a fixed validation set and producing a model that is unlikely to overfit.

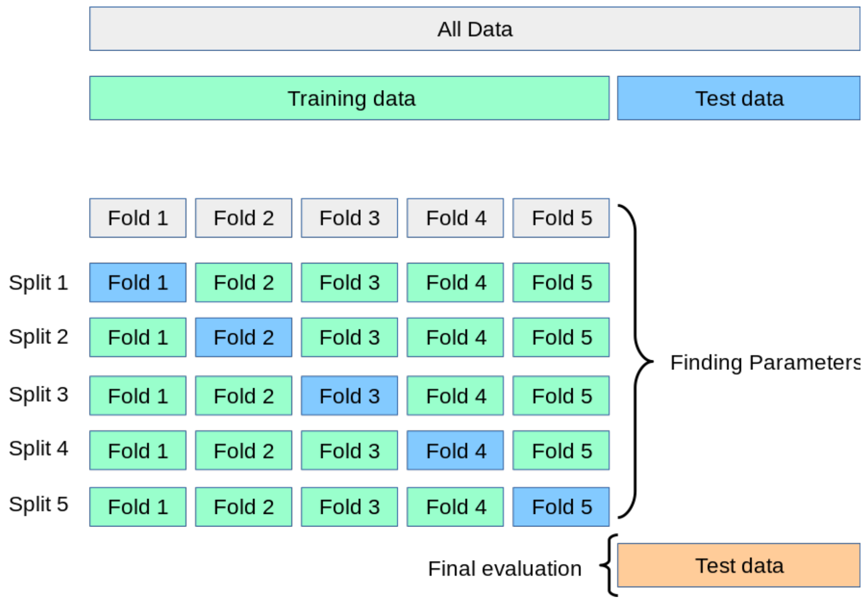


Figure 4.8: k-fold cross - validation [29]

4.2.6 Ensemble models

Ensemble learning is a particular approach in Machine Learning that aims at improving the predictive capabilities of a model, reducing the error, combining the outputs of multiple instances of the same model. Those algorithms are helping to improve the robustness, reducing the sensitivity to outliers and preventing overfitting at the cost of increasing the computational complexity. Different classes are present in literature:

- Bagging Regressor: trains the same model on different subsets of the training set, typically created by random sampling with replacement. The final output is obtained combining the results, in particular by means of voting (for classification) or averaging (for regression);
- Boosting Regressor: trains a sequence of models with the peculiarity of changing training data to focus attention on examples that previous fit models have gotten wrong;

A schematic overview is also given in Fig. 4.9.

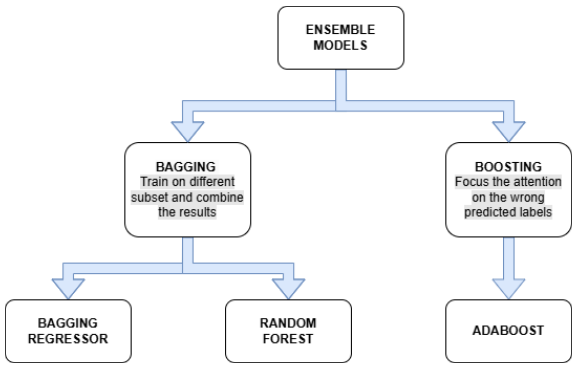

ENSEMBLE MODELS
BAGGING Train on different subset and combine the results
BOOSTING Focus the attention on the wrong predicted labels
BAGGING REGRESSOR
RANDOM FOREST
ADABOOST

Figure 4.9: Ensemble models

AdaBoost

In this Section, a more detailed description of *AdaBoost* algorithm is given. The peculiarity of AdaBoost (also called Adaptive Boosting) is that a certain number of base estimators, called *weak learners* [30], are added sequentially to the ensemble model. The consequent model is trying to correct the predictions of its predecessor. After each round, the algorithm changes the weights of the mispredicted outputs so that the following model is forced to focus on these data. The final output is a weighted sum of the predictions of all the cascade of weak learners. These weights are determined based on the individual accuracy of the weak learner. The importance

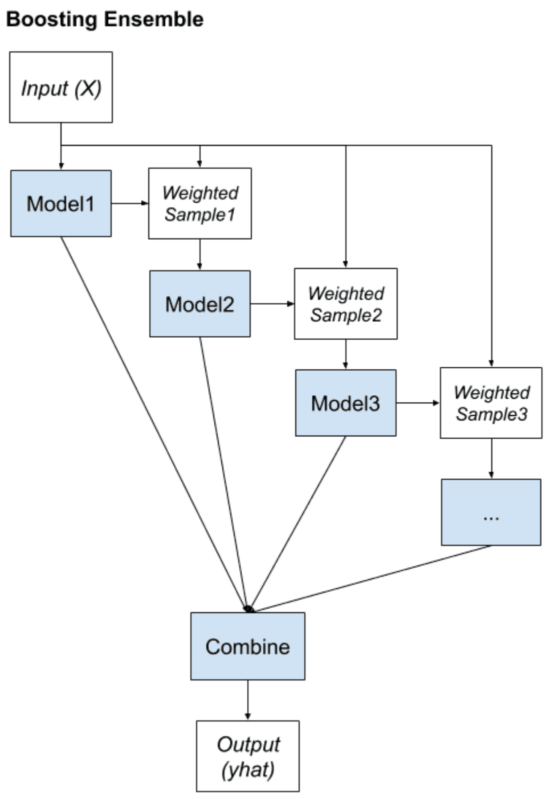


Figure 4.10: AdaBoost schematic [31]

that each weak learner has in the final ensemble is controlled by the learning rate hyperparameter. In particular, the weight update shown in Eq. :

$$\alpha_t = \text{learning rate} \cdot \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \tag{4.4}$$

A higher learning rate increases the contribution of each weak learner reducing the number of base estimators that must be trained. On the contrary, a smaller learning rate generally requires a higher number of rounds to reach the same performances. AdaBoost is implemented using the scikit-learn library [32].

4.2.7 Learning Curve

Another important aspect during the traning process of Machine Learning model, is the ability to monitor its learning capabilities. This is fundamental to understand if the model is overfitting, underfitting or achieving a good fit. Learning curves are particular useful when training Neural Networks, as it is required to check the loss during the training process. In the development of this thesis, two learning curves are used:

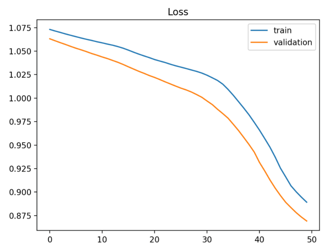
- Training and Validation Loss;
- Training and Validation MAE;

Training and Validation Loss are reporting the loss metrics during the learning process of the model. It is particularly useful to understand what are the performances as the model is gathering experience on the dataset. Specifically, it can be used to diagnose underfit, overfit or good fit by comparing the loss on the training subset and the validation subset. Fig. 4.11 reports these 3 behaviours.

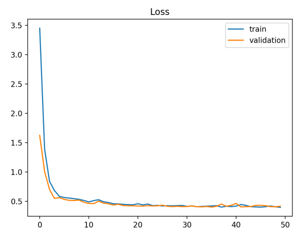
Training and validation MAE are also useful metrics to understand the entity of the Mean Absolute Error on the traning set and on the validation set, by comparing the gap among them. This is useful to assess the generalization capabilities of the model. A higher gap means that the model struggles on the test set, possibly under-performing. On the other hand, a smaller gap is indicating a that the model is able to generalize, ensuring similar performances on the training set and on the test set, which is highly desirable.

4.2.8 Early Stopping

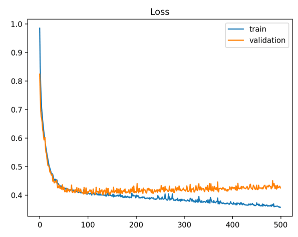
Early stopping is used to prevent overfitting during the training. It is used to stop the traning when the performances on the validation set are beginning to degrade with respect the training ones, like in Fig. 4.11c. This is ensuring that the model is not overfitting, reducing its generalization capabilities. This tool is implemented using Keras library [34] and it is controlled by the *patience* value that represents the maximum tolerated number of epochs with no improvement, before that the training is stopped.



(a) Underfit - example



(b) Good Fit - example



(c) Overfit - example

Figure 4.11: Examples of learning curves: Underfit, Good Fit and Overfit [33]

4.3 Support Vector Regression (SVR)

Support Vector Regression belongs to the family of the Support Vector Machines that can be used to perform both classification and regression tasks. Much of the literature on Support Vector Machines focuses on the classification task. Therefore, the main concept are going to be presented referring to the classification and later on these concept are going to be adapted to the regression task

4.3.1 Support Vector Machines for classification

The aim of the Support Vector Machine is to find the optimal *Hyperplane* that is able to maximise the *margin*. In order to understand what the margin is, the concept of Support Vector must be introduced. Support Vectors can be defined as the closest point to the *hyperplane* that is required to be found. In Fig. 4.12, a generic representation of this concept is presented. Given a distribution of points (for which the hypothesis of *independent and identically distributed (iid)* holds), Support Vector Machines aims at finding the *Hyperplane* that maximizes the margin, evaluated as a distance between the *Hyperplane* itself and the Support Vector.

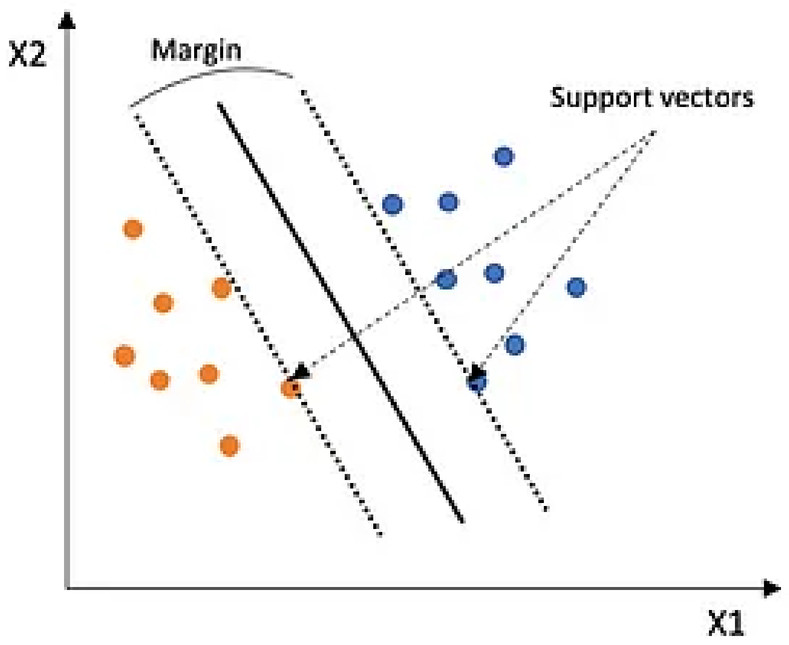


Figure 4.12: Support Vector Machine representation [35]

4.3.2 Support Vector Regression: concept

Starting from these definitions, an analogue concept is exploited when dealing with regression task. Literature refers to these models as Support Vector Regression (SVR)[36]. Specifically, SVR introduces a $\epsilon - tube$, around the *Hyperplane*, that is used for the optimization problem. The goal is to find the best tube able to solve the regression problem, balancing the complexity of the model and the losses. Fig. 4.13 shows the tube in case of a 1D problem. Since the goal is to predict a continuous value, conversely with respect classification, Eq. 4.5 and Eq. 4.6 must be used in case of 1D and multidimensional problem respectively. At training time, the model is learning the internal parameters, *w (weight)* and *b (bias)*, that are minimizing the objective function. Once these are determined, by means of Eq. 4.5 and Eq. 4.6, it possible to make predictions, starting from the test set.

$$y = f(x) = \langle w, x \rangle + b = \sum_{j=1}^{M} w_j x_j + b, \quad y, b \in \mathbb{R},\ x, w \in \mathbb{R}^M \tag{4.5}$$

$$f(x) = \begin{bmatrix} w \\ b \end{bmatrix}^T \begin{bmatrix} x \\ 1 \end{bmatrix} = w^T x + b, \quad x, w \in \mathbb{R}^{M+1} \tag{4.6}$$

4.3.3 Optimization problem and hyperparameters

As already mentioned, SVR finds the best fit by means of an optimization problem. For this reason, it is needed to consider both the objective function and some constraints. The mathematical description of those is quite useful to understand what are the hyperparameters that are required to optimize the problem. At this

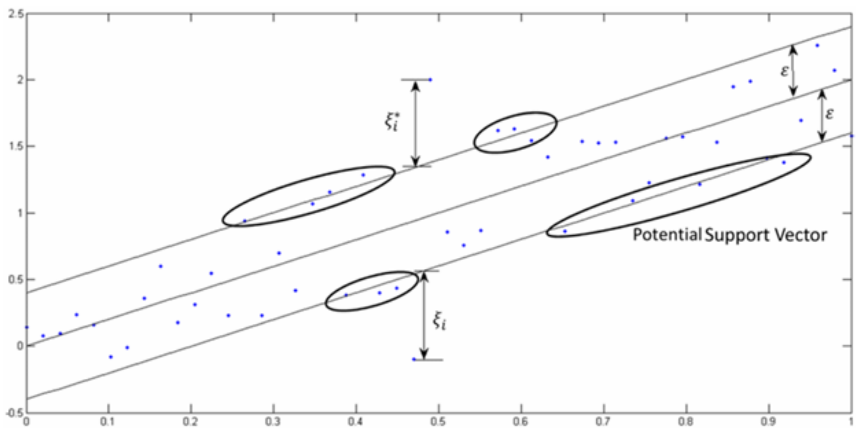


Figure 4.13: Support Vector Regression representation [36]

point, it is clear how the model should trade-off between the complexity and the accuracy of the results. It should be accurate enough to well generalize to unseen data and at the same time should be easy enough to avoid overfitting (more details in Sec. 4.2.3).
From the mathematical point of view, therefore, the objective function can be expressed as shown in equation (4.7).

$$\min_{w} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{where} \quad \mathbf{w} = \begin{bmatrix} w \\ b \end{bmatrix} \tag{4.7}$$

By means of this objective function, it is ensured that the weights that the model learns from the training set, are as small as possible. This can be seen as a regularizer effect that allows to provide the simplest solutions, avoiding to have complex models that are likely to overfit [21].

The constraint, on the other hand, is to minimize the error between the prediction of the model and the ground-truth (the real label). The error is controlled by the $\epsilon - tube$ in which no penalization is given. It is linked to the hyperparameter ϵ that directly affects the width of the tube. This value is related the cost associated to an error in the loss function. In particular:

- a smaller ϵ is providing a lower tolerance for errors, as the width of the tube will be shrunk and therefore a larger number of samples are contributing to the loss function.

- a larger ϵ on the contrary, is associated with a higher tolerance, as the number of points contributing to the loss is decreasing.

The loss function can therefore be expressed as shown in Eq. 4.8 and in Fig. 4.14.

$$L_\varepsilon\left(y, f(x,w)\right) = \begin{cases} 0 & \text{if } |y - f(x,w)| \le \varepsilon \\ |y - f(x,w)| - \varepsilon & \text{otherwise} \end{cases} \tag{4.8}$$

If the absolute value of the distance among the ground-truth and the prediction is lower than ϵ the loss associated is zero (i.e. samples are inside the tube). On the contrary, the loss function depends on the error and on ϵ. Since the width is strictly

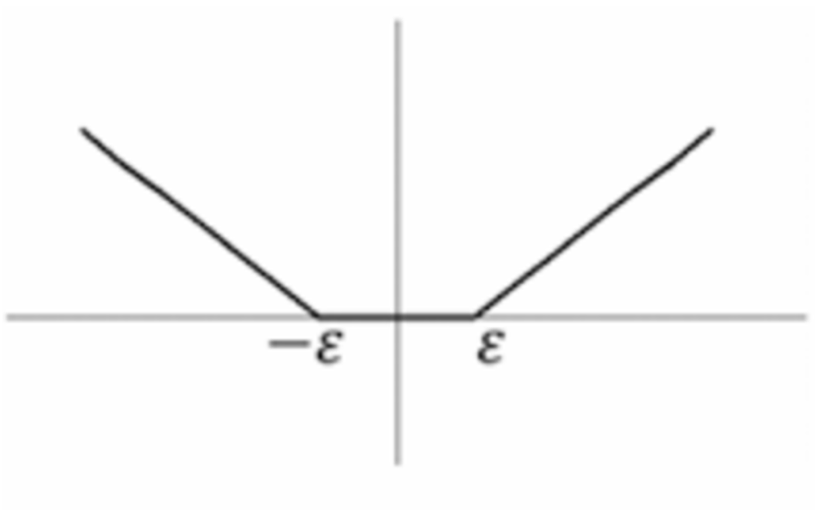


Figure 4.14: ϵ linear loss [36]

related to the support vectors, it's straight forward how these points are playing a relevant role on the performance of the model.

The aforementioned description of the optimization problem is called hard-margin as inside the objective function (Eq. 4.7) there is no relation with the admissible error. As consequence, the model is trying to learn the weights (w and b) to fit all the data inside the $\epsilon - tube$. Depending on the distribution of the data, this condition may be too strict proving actually no solution.

To overcome this problem, in the literature [36], a soft-margin approach is described. The main difference with the hard-margin is that, in the latter, it is introduced the possibility to violate the constraints by introducing a slack variable (ζ), ensuring an higher level of flexibility. The objective function is therefore modified as shown in Eq. 4.9:

$$\min_{w} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\zeta_i \tag{4.9}$$

This representation is introducing a new hyperparameter (C), that can be regarded as a penalty factor associated with the error. It is added to the L2 norm of **w** in the objective function. In particular, C is linked to the cost of making an error:

- A large C aims at minimizing the error, since the model will *pay* a huge cost whenever it commits a mistake. This means that the model tries to fit more closely the training data, but this may also lead to overfitting
- A smaller C is giving the possibility to make some mistakes, ensuring to have a simpler model that may better generalize to unseen data.

Fig. 4.15 shows the effect of various C on the model fit. In particular, three different models with different values of C (0.1, 10, 1000) are trained on the same training set. It is possible to notice how:

- C = 0.1 is producing a high error, since the penalty cost associated is very low;
- C = 10 allows to better exploit the trend of the data;

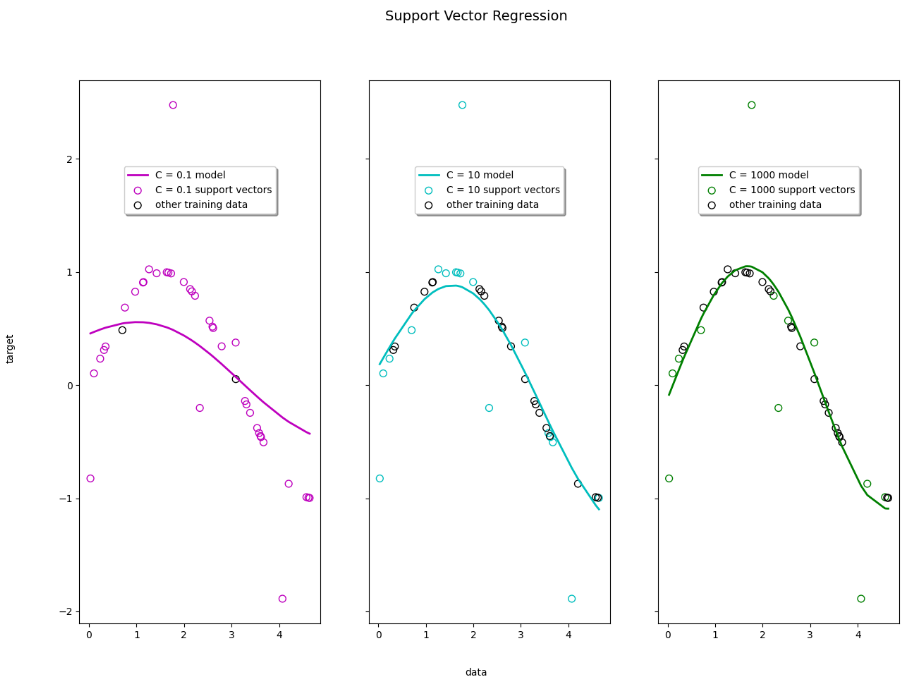


Figure 4.15: Effect of C on the performances of SVR [37]

- C = 1000 is almost perfectly fitting the data with the risk of overfitting.

What has to been considered is that there is not a unique solution. Machine learning models are a trade-off among the complexity of the models, the performances on the test set (in terms of Mean Squared Error and/or Mean Absolute Error) and the time required to accomplish the task. Therefore, is really important to make several test in order to be able to find the best compromise for the specific application.

The Kernel Trick

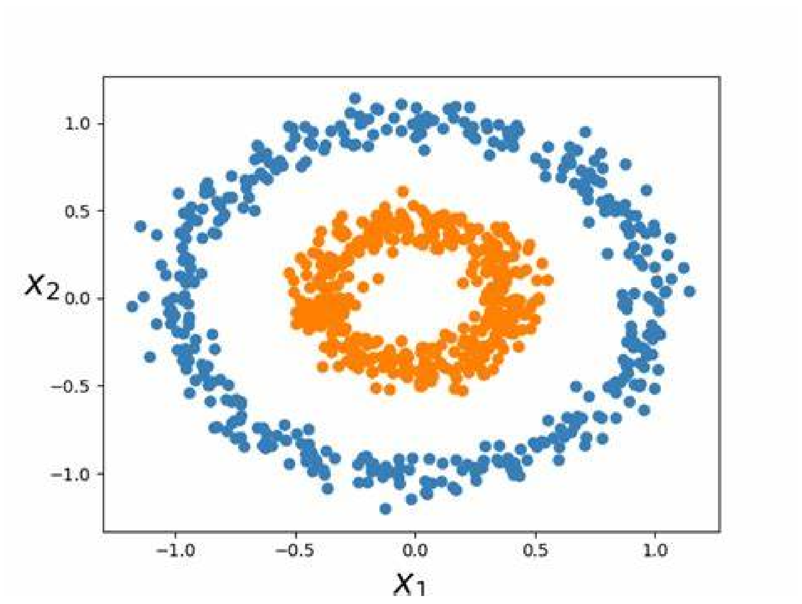
The theory discussed until now, is assuming that *f(x)* is linear. This is a strong assumption since in real word applications, data are showing a high degree of non-linearity. The correct choice of the kernel can heavily affect the performance of the model [38]. When data are not linearly separable in the original input space, it is possible to project them into a higher dimensional space where the points are linearly separable. This concept, shown in Fig. 4.16, is known in literature as *Kernel trick*.

From the mathematical point of view, if we have a data $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ and a map $\phi : \mathcal{X} \rightarrow \mathbb{R}^N$ then

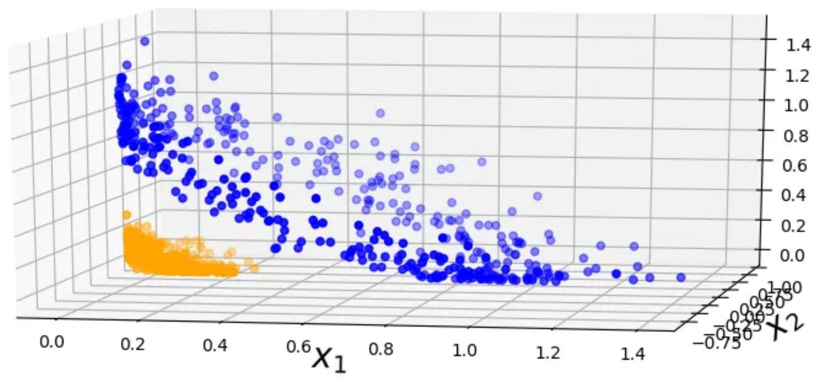
$$k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle \tag{4.10}$$

is representing a kernel function. In particular, it takes in input the vectors in the original dimensional space (x and z) and returns the dot product of the vectors in the higher dimensional space.

This new representation is allowing to solve the non-linearly separable problem. Indeed, looking at Fig. 4.16a as example, it is clear that it's not possible to define a linear hyperplane able to separate the two classes (blue and orange points). By



(a) Example of not linearly separable data



(b) Projection of not linearly separable data in a higher dimensional space

Figure 4.16: Kernel trick visualization [14]

leveraging, on a *nonlinear transformation*, those points are projected in a higher dimensional space 4.16b where they can be correctly separated by the hyperplane.

The big advantage, from which the name *trick* comes, is that it is not required to know how the points from the original space are mapped after the *nonlinear transformation* (i.e. the actual function). For example, the transformation that allows to move from the original space 4.16a to the higher dimensional space Fig. 4.16b is shown in Eq. 4.11:

$$\phi(\mathbf{x}) = \phi\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix} \tag{4.11}$$

Kernels are special function for which the dot product of the transformed vectors is just a function of the coordinates in the original dimensional space. In particular, this is clear in Eq. 4.12. This is corresponding to project points in the higher dimensional space without having to evaluate $\phi(x)$.

$$\begin{aligned}
\phi(\mathbf{x}_1)^\top \cdot \phi(\mathbf{x}_2) &= \begin{pmatrix} x_{11}^2 \\ \sqrt{2}x_{11}x_{12} \\ x_{12}^2 \end{pmatrix}^\top \cdot \begin{pmatrix} x_{21}^2 \\ \sqrt{2}x_{21}x_{22} \\ x_{22}^2 \end{pmatrix} \\
&= \begin{pmatrix} x_{11}^2 & \sqrt{2}x_{11}x_{12} & x_{12}^2 \end{pmatrix} \cdot \begin{pmatrix} x_{21}^2 \\ \sqrt{2}x_{21}x_{22} \\ x_{22}^2 \end{pmatrix} \\
&= x_{11}^2 x_{21}^2 + 2x_{11}x_{12}x_{21}x_{22} + x_{12}^2 x_{22}^2 \\
&= (x_{11}x_{21} + x_{12}x_{22})^2 \\
&= \left(\mathbf{x}_1^\top \cdot \mathbf{x}_2\right)^2
\end{aligned} \tag{4.12}$$

Well known examples of SVM kernels are:

- **Radial Basis Function (RBF)** :

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right) \text{ for } \sigma > 0 \tag{4.13}$$

- **Linear** :

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}' \tag{4.14}$$

- **Polynomial**:

$$K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^\top \mathbf{x}')^r \text{ for any } r > 0 \tag{4.15}$$

In Fig. 4.17 is shown a comparison of three different models with three different kernels. In particular, *RBF*, *linear* and *polynomial* are assessed. It's clear how the choice of the kernel is affecting the results, resulting in a different fit. Indeed, kernel directly influences how well the model adapts to the underlying structure of the data.

For the development of this thesis, RBF kernel has been used and therefore it's important to discuss about the related theory since it will introduce a hyperparameter that has been used in the optimization process of the model.

Radial Basis Function is extremely popular among the different kernel because it allows to exploit a similar concept with respect the Gaussian distribution [39]. Referring to Eq. 4.13:

- γ represents the variance and it's an hyperparameter;
- $\|\mathbf{x} - \mathbf{x}'\|^2$ represents the L_2 norm between point x and x'.

Since this kernel equation depends on the distance, two scenarios can be found:

- When points are really close ($\|\mathbf{x} - \mathbf{x}'\|^2 \approx 0$), they are *similar*;
- When points are far away, kernel values is less than 1 and closer to 0 as points are more spaced. In this case points are said to be *dissimilar*

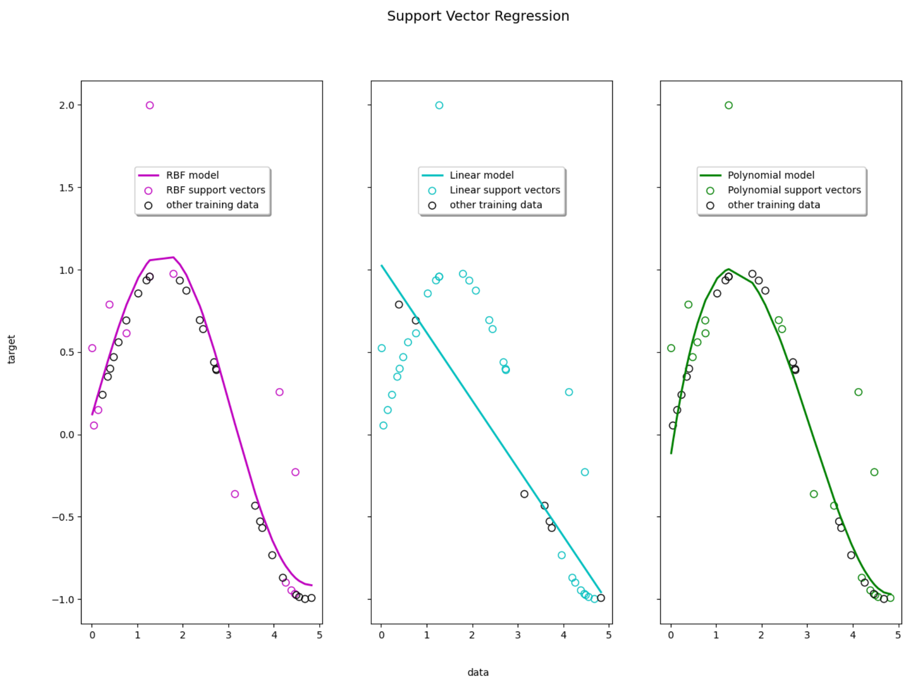
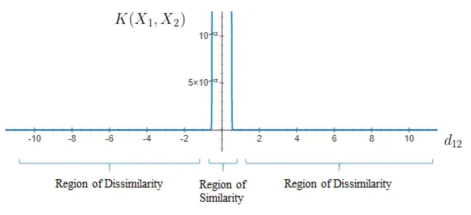
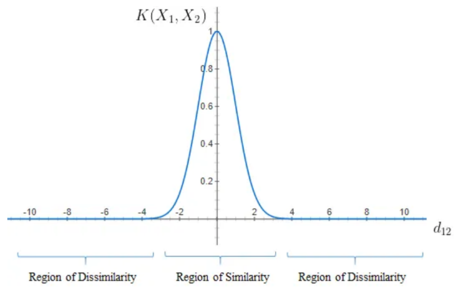


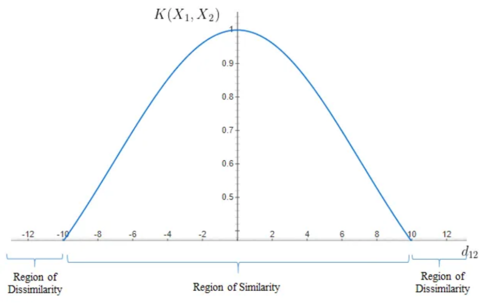
Figure 4.17: Effect of the Kernel on model performance [37]



(a) RBF kernel with γ=0.1



(b) RBF kernel with γ=1



(c) RBF kernel with γ=10

Figure 4.18: Region of Similarity in function of γ [14]

The condition of similarity is controlled by γ. The effect of the hyperparameter is shown in Fig. 4.18 that is reporting the similarity condition (on the y-axis) versus the distance among the points (on the x-axis) for different values of the hyperparameter. It is clear how, as γ is increasing, also the shape of the *region of similarity* is becoming larger. This means that points that are farther away can be considered similar. On the contrary, small values of γ (i.e. 0.1) are providing a very narrow region. It is very important therefore to perform several tests in order to find the best values of γ able to provide the best performance of the model.

Lastly, in Tab. 4.1 all the hyperparameter used in the developing of the SVR model are reported:

Hyperparameter	**Description**
Kernel	For non-linearly separable dataset
C	Penalty factor on the errors
ϵ	controls the width of the $\epsilon - tube$
γ	controls the region of similarity of the RBF kernel

Table 4.1: Resume of the SVR Hyperparameters

4.3.4 Experimental Results

This part is dedicated to the discussion of the results obtained using SVR models. Here the output predictions are:

- Torque;
- Torque ripple;

Torque analysis

The first analysed output is Torque. To train and test the model, the split ratio shown in Fig. 4.19 has been used. This means that on a total of 5000 samples, 3500 are used for training and 1500 for testing.

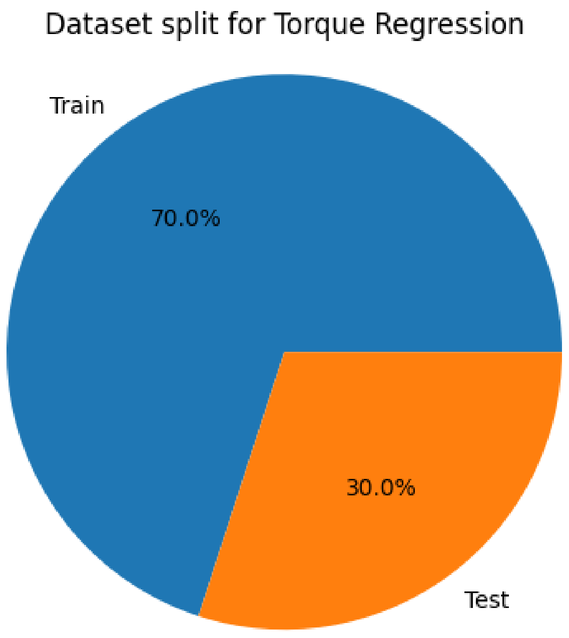


Figure 4.19: Dataset split ratio - Torque regression

The optimal hyperparameters found after the optimization process are shown in Tab. 4.2: Due to the high value of the hyperparameter C, the model is paying a

Hyperparameter	**Kernel**	**C**	ϵ	γ
Value	rbf	1000	1	0.1

Table 4.2: Resume of the SVR hyperparameters for Torque regression

very large cost whenever it makes an error. Since this cost is linked to the objective function, that has to be minimized, it means that the model is forced to minimize as much as possible the error, achieving a very good fit on the training dataset. At the same time, γ is producing a very narrow region of similarity. This setup achieved a R^2 score of 0.997 on the training set with a total fit time of 16.188 seconds. This result is expected due to the aforementioned considerations regarding the hyperparameters.
It is quite interesting to understand the features importance of the different input. In particular, those can be regarded as the weights that the specific input feature has on the final prediction of the output. This is shown in Fig. 4.20 where some analogies with the Pearson correlation can be found (Fig. 3.14). In particular, the trained model is identifying *Current Phase Angle*, *Rotor Radius*, *Barrier Position* and *Tooth Length* as the most important features for the torque regression evaluation.

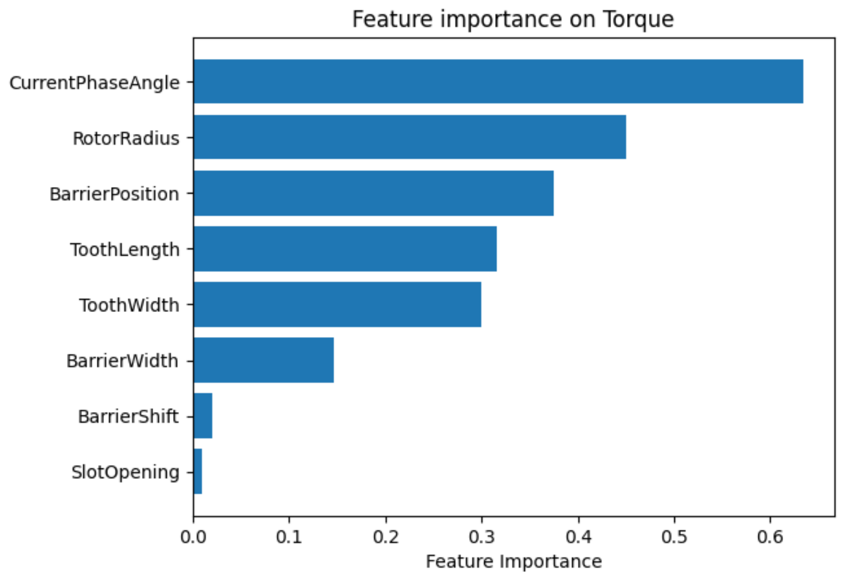


Figure 4.20: Feature importance SVR model Torque regression

Performances on test set At this point, the performances at test level are evaluated. In particular, Fig. 4.21 reports the real values of torque plotted against the predicted values of the model. The red line is representing the regression line.

As it is possible to notice, the points are well distributed along the line, meaning that the prediction are very close to the real values. The R^2 score is 0.998 as the model is well able to exploit the variability of the test set. The total fit time is 0.256 seconds and the Mean Squared Error on the test set is 1.12.
It is interesting to understand what is the error associated with every prediction.

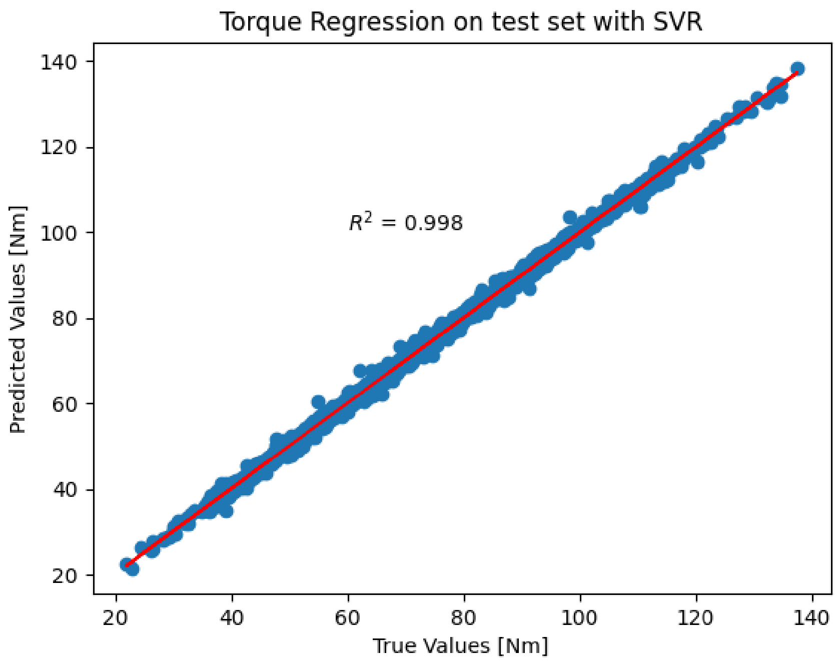


Figure 4.21: Torque regression on test set- SVR

To asses to this information, the relative error for each prediction is evaluated. The results are reported in Fig. 4.22, Fig. 4.23, and Fig. 4.24.

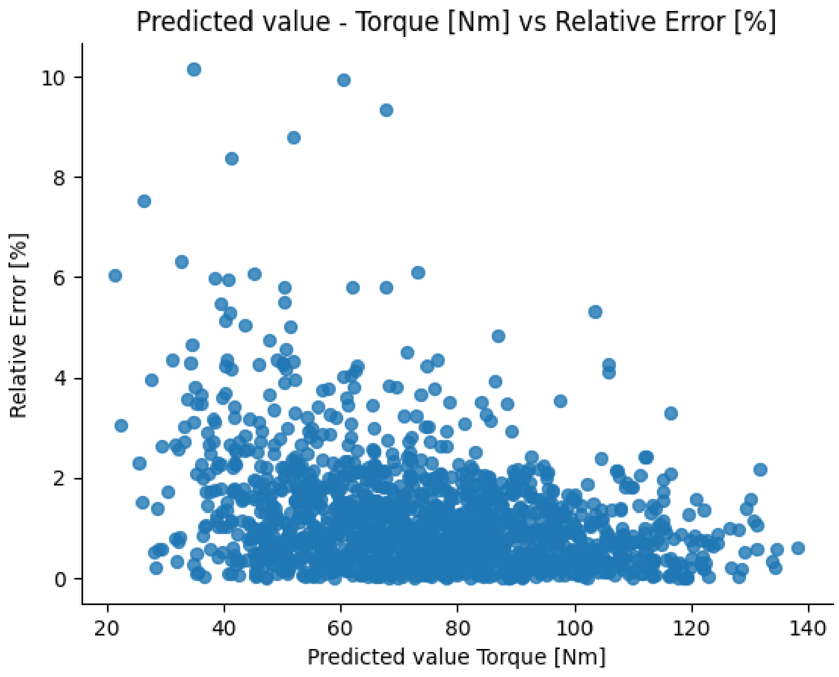


Figure 4.22: Prediction with relative error on test set - SVR

It is apparent from these graphs how:

- The relative error is below 10% and the majority of the predictions are below 6% (Fig. 4.22);

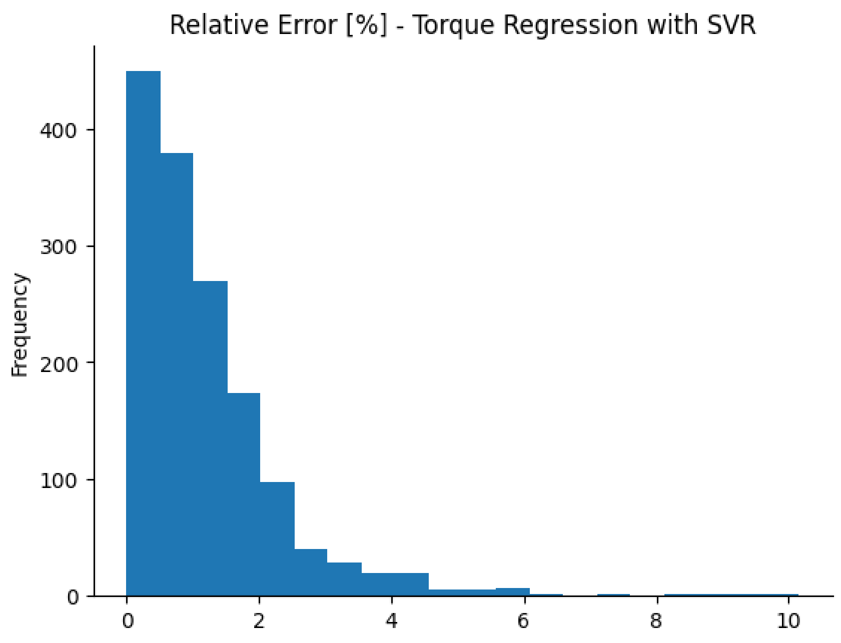


Figure 4.23: Distribution of relative error on test set - SVR

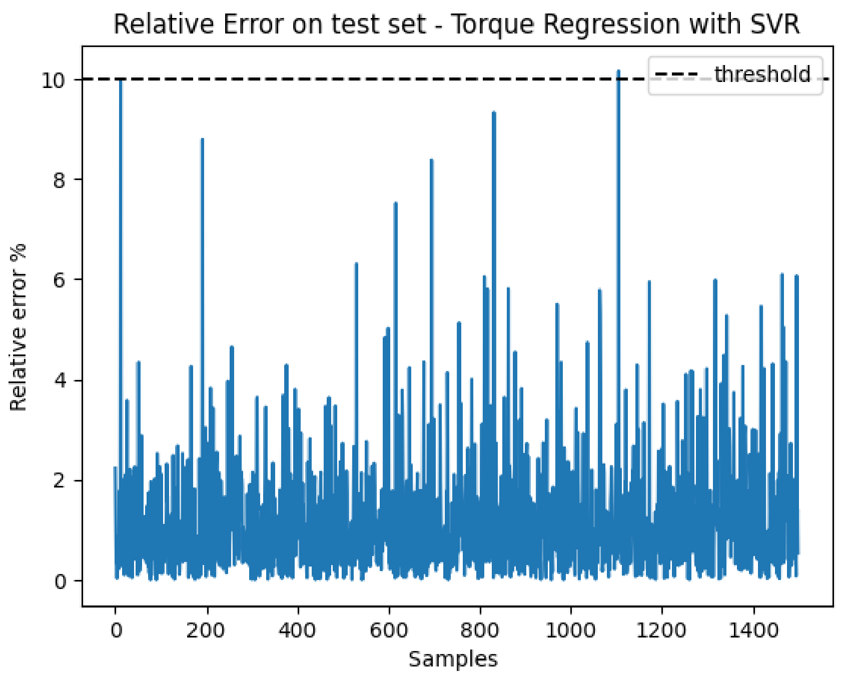


Figure 4.24: Samples with relative error on test set - SVR

- The lower portion of torque prediction is associated with the highest values of relative errors, while the error decreases as the torque values increase (Fig. 4.22);
- Over 1200 (out of 1500 total samples) are predicted with a relative error lower than 2% (Fig. 4.23);
- Fig. 4.24 is used to understand the number of points that are above an arbitrary threshold and their relative position in the Torque-Torque Ripple plane.

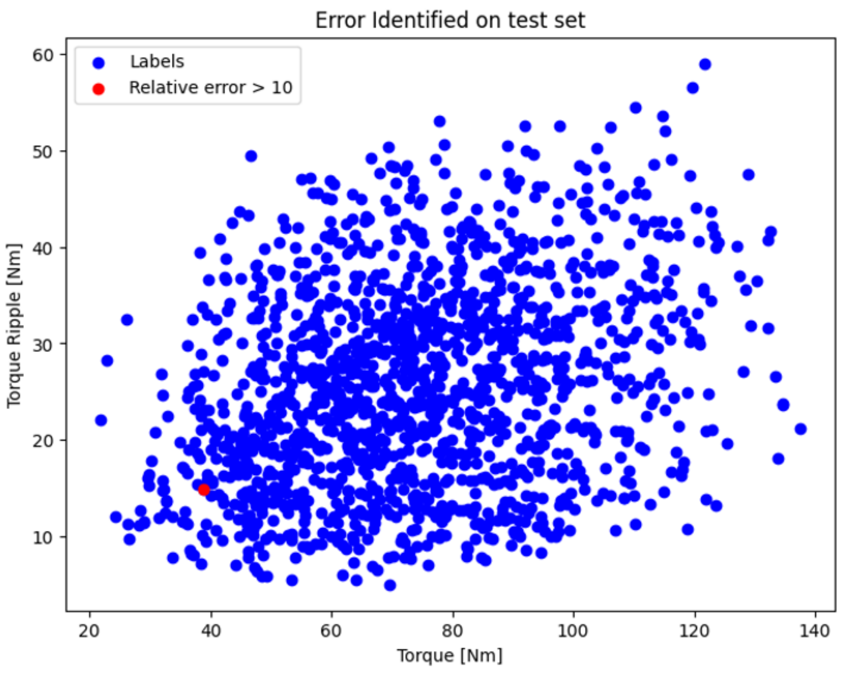


Figure 4.25: Samples with relative error on test set design space - SVR

Given a threshold of 10%, just one point is above, and its position is shown in Fig. 4.25.

Influence on Training size These results are suggesting that the optimized model is able to well capture the statistics of the training set and to obtain low errors at test time. It is interesting therefore to understand how the performances of the model are changing when using a limited amount of training samples. Ideally, this allows to define what is the minimum number of samples that can be used to achieve satisfactory results. To evaluate this aspect, the best model has been trained using a subset of the original train split ratio. In particular, starting from the original ratio (Fig. 4.19, different train sizes have been considered. While the test set has been fixed to 1500 samples to make a fair comparison, different percentage of the train samples size are evaluated (10%, 25%, 50%, 75% and 100%), which are translating in respectively 350, 875, 1750, 2625 and 3500 samples. The performances of the different tests are compared in Fig. 4.26. As can be seen, training time in increasing as the number of samples for training is increasing. The advantage of smaller train size can be seen is the R^2 and MSE plots, where it is clear that the performances of the model are reaching a plateau after 2625 samples while the training time is almost 4 seconds faster. This aspect is also evidenced at test time, where the prediction time of 75% train size is lower with respect of the 100% subset. The important finding is that if optimization of the computational resources is a concern, the SVR model trained on a portion of the total training set, could lead to a comparable result within a limited time.

Torque ripple analysis

The discussion now moves to the second output to be analysed which is the torque ripple. Preliminary tests have been performed in order to understand which is the

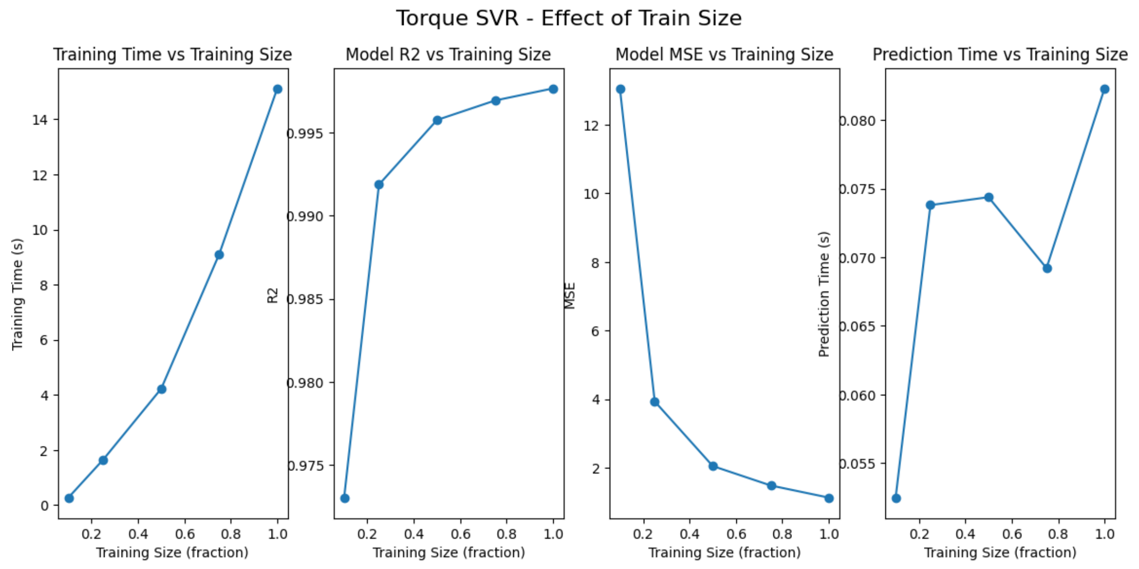


Figure 4.26: Effect of train size on torque regression

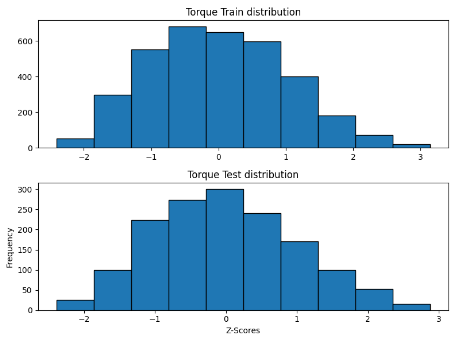
best implementation to consider. In particular, torque ripple regression has been evaluated considering:

- Torque ripple as direct value obtained from SyR- e simulations;
- Torque ripple normalized to the corresponding torque output;

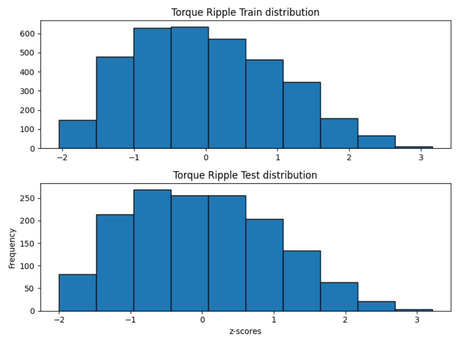
The second simulation setup is considered since it allows to better scale the torque ripple and to correlate it to the geometry of the considered IPM machine. Throughout this thesis, the term *torque ripple percentage* is used to refer to the latter setting. To actually evaluate which was the best configuration to use, the z-score distribution of the torque ripple and torque ripple percentage are compared to the torque distribution, as accurate predictions were obtained. In Fig. 4.27, the comparison among them is shown:

Since torque is well predicted, it is taken as example for the ideal statistics to achieve. By means of z-score analysis, it is possible to notice how the torque behaviour (Fig. 4.27a) is close to a normal distribution with a central tendency around 0 and almost symmetrical tendency. This condition is desirable as it allows models to better capture the statistics of the data, providing accurate results. Moving to the *torque ripple percentage* (Fig. 4.27c), it is clear how the distribution is more spread with respect the previous case. Moreover, it is right-skewed, showing how a certain amount of data-samples are far away with respect the mean value of the distribution. Those points may be seen as outliers and they may represent a problem for the model as SVR performances are affected by the presence of noise and out of distribution points [40]. This makes the *torque ripple direct configuration* the most indicated to be solved with Support Vector Regression. Indeed, even thought the distribution is slightly different with respect a normal one, the presence of outliers is less prominent (Fig. 4.27b).

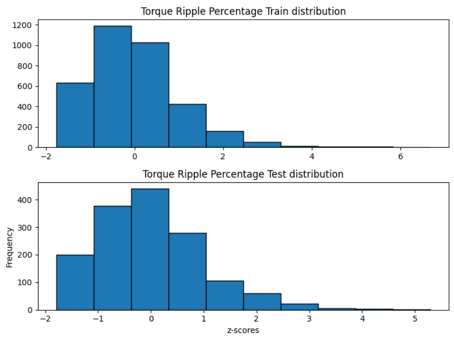
Torque Ripple on direct values As mentioned before, the torque ripple analysis has been performed using the direct values obtained with SyR- e simulations,



(a) Torque Z-Score Distribution



(b) Torque ripple Z-Score Distribution



(c) Torque ripple Percentage Z-Score Distribution

Figure 4.27: Z-score Distribution comparison

without normalizing to the corresponding torque value. The first operation is to define the split ratio among the train and test subset, remembering that in total the dataset is composed by 5000 samples. In Fig. 4.28, the split is reported. With respect the torque case, now 3750 samples are used for traning and 1250 for testing. This decision has been taken after several test that are showing that this split is a good trade - off among the performances and the risk of overfitting.

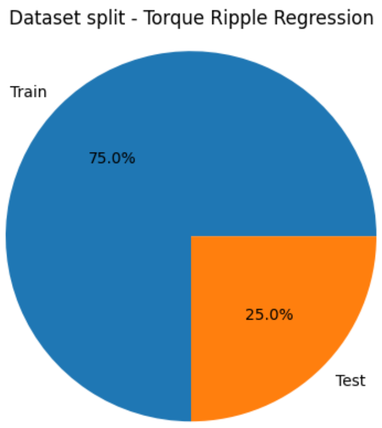


Figure 4.28: Dataset split ratio - Torque ripple regression

With this setup, the optimal model found, after the optimization process, is shown in Tab. 4.3.

Hyperparameter	**Kernel**	**C**	ϵ	γ
Value	rbf	620	0.44	scale

Table 4.3: Resume of the SVR hyperparameters for torque regression

Conversely with respect before, in this case, the value of C is lower, meaning that the model is accepting to make some mistakes, paying a smaller *cost*. The optimal model is therefore found relaxing the constraints about the strict fit, increasing the flexibility. For what regards the hyperparameter γ, for the region of similarity, it is set to *scale*, whose value can be seen in Eq. 4.16. This value is linked to the statistics of the dataset, allowing to obtain a region of similarity that is more representative of the data. With this setup, the model achieved a R^2 score on training subset equal to 0.975 and a total fit time of 101.128 seconds. As already done also for the torque regression, it is possible to extrapolate the feature importance, to understand how much each feature is contributing to the final output prediction. The results are shown in Fig. 4.29 and they are pretty consistent with what has been previously evaluated in the Pearson Correlation Plot (Fig. 3.14), giving the majority of the decision weight to features as *Rotor Radius, Tooth Width, and Barrier Position.*

$$\text{scale} = \frac{1}{n_{\text{features}} \cdot X.var()} \tag{4.16}$$

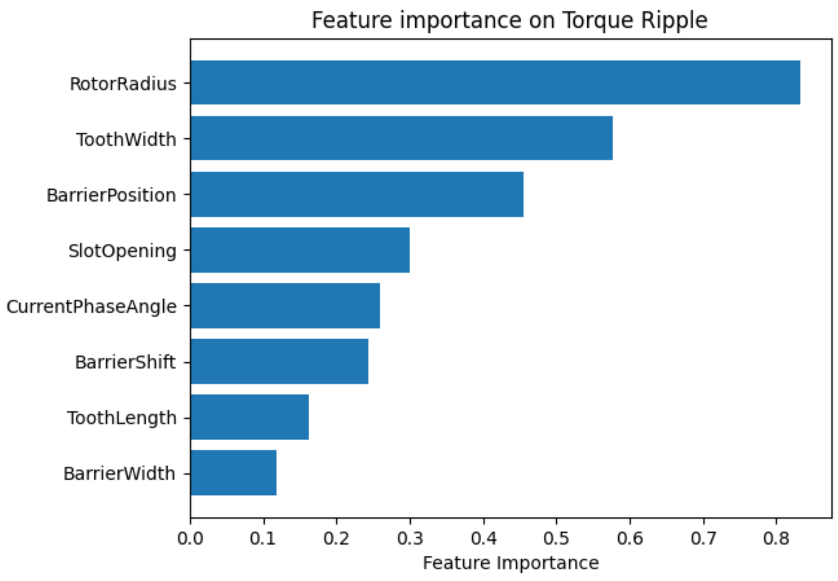


Figure 4.29: Feature importance SVR model torque ripple regression

Performances on test set Moving with the discussion, the performances at test time are now presented. Starting with the torque ripple regression, Fig. 4.30 shows how the prediction is less accurate with respect torque analysis, reporting multiple points that are far from the red regression line. The R^2 on test subset is 0.983 indicating that the model is anyway able to exploit the variability of the test set. The prediction time is 0.378 seconds and the Mean Squared Error is 1.86. The greater errors are concentrated in the lower part of the torque ripple region, while higher values are more closer to the regression line. This reflects also in Fig. 4.31, that shows how the range between 0 Nm and 35 Nm is predicted with a relative error that reaches its peak at 80%, which is much larger with respect the torque model. However, this does not mean that the model is behaving very poorly at prediction stage.

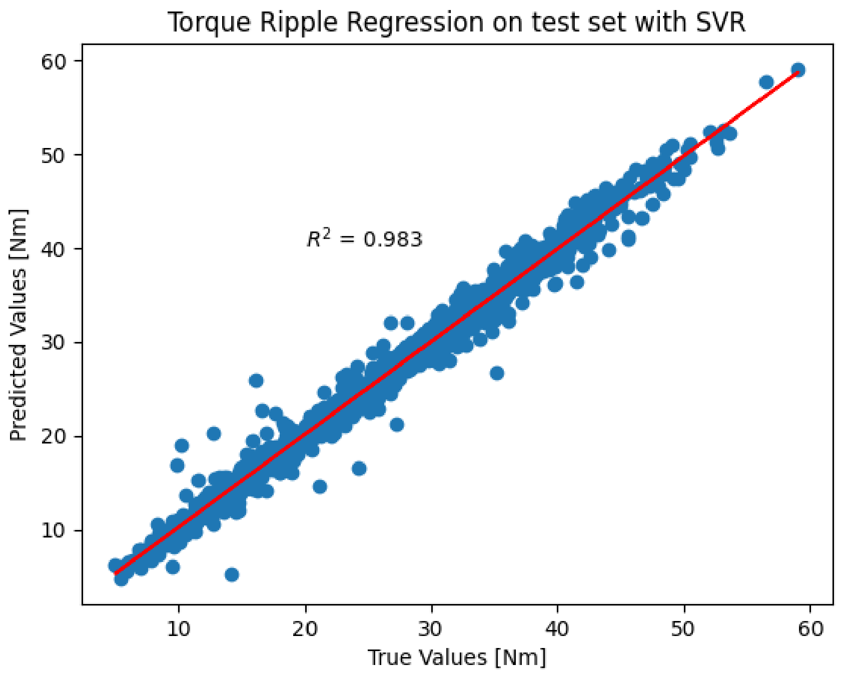


Figure 4.30: Torque ripple regression on test set - SVR

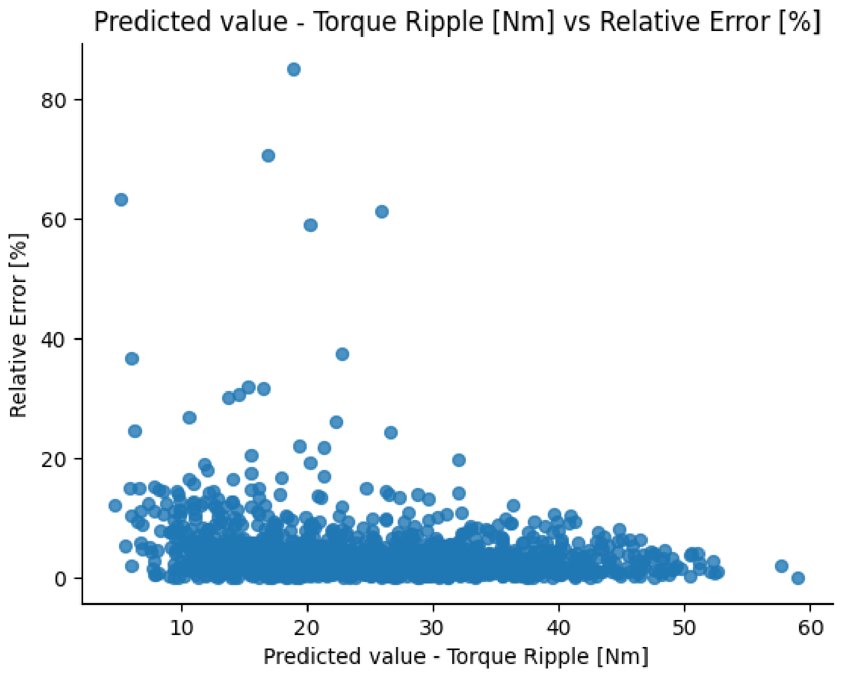


Figure 4.31: Torque ripple prediction with relative error on test set - SVR

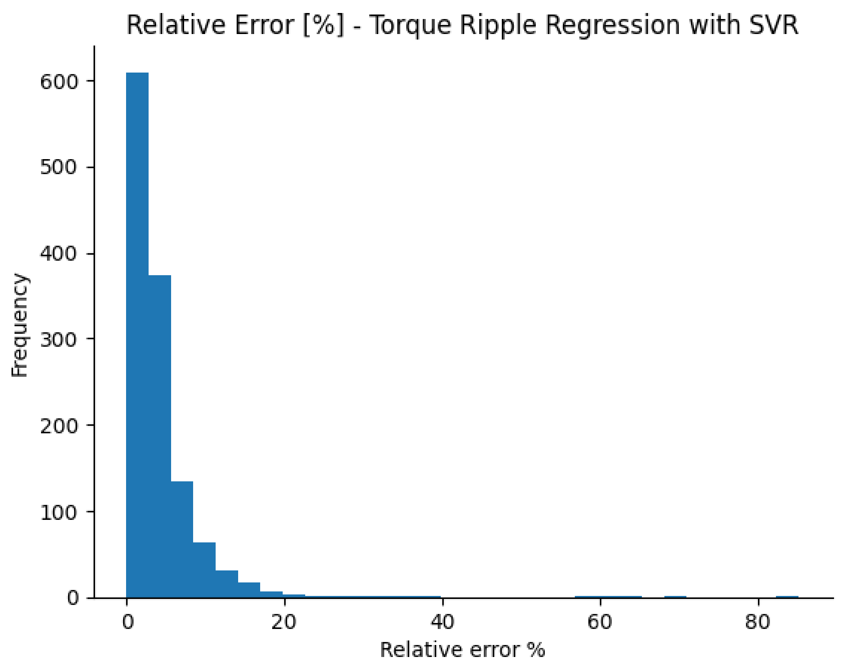


Figure 4.32: Distribution of relative error on test set - SVR

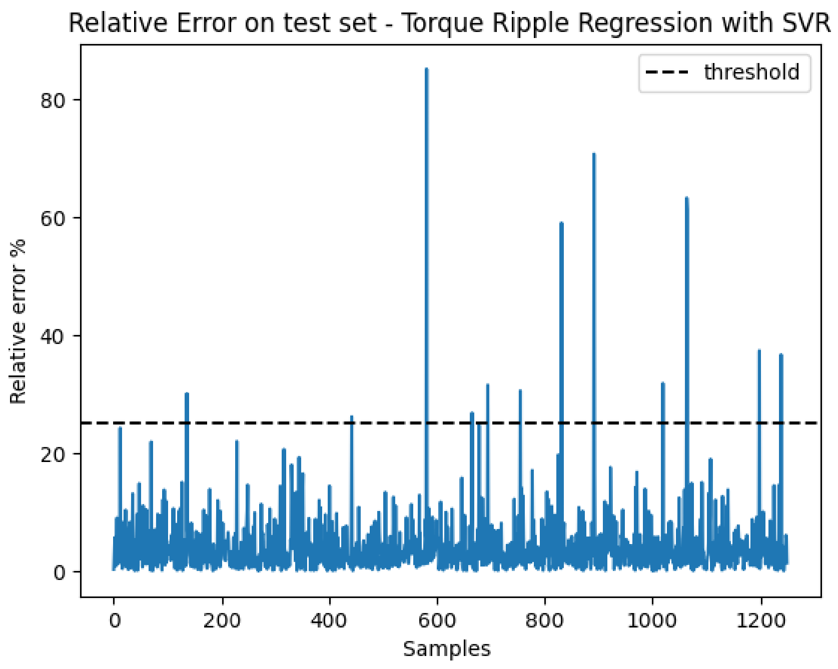


Figure 4.33: Samples with relative error on test set - SVR

Indeed, looking at Fig. 4.33, where the relative errors are grouped into bins, it is possible to see that the more than 600 points are predicted with a relative error which is lower with respect 10%. Lastly, the relative error for each sample is compared with a threshold manually set to 25%. This limit is larger with respect the torque regression since the model is performing slightly worse. Additionally, this value delineates the cluster of well-predicted points from the mispredicted ones. The number of points that are exceeding this limit is 13 and their location is shown

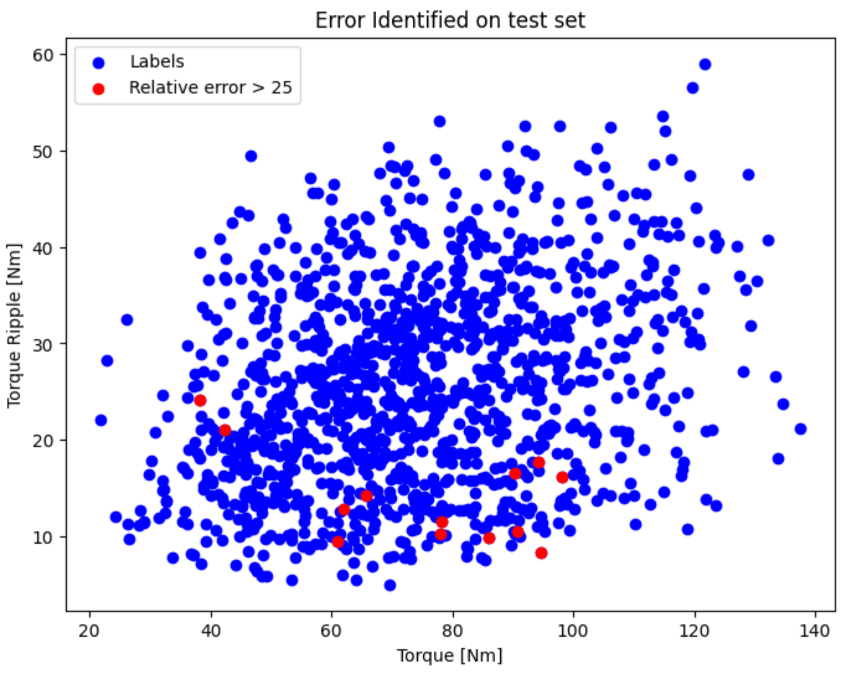


Figure 4.34: Samples with relative error on test set design space - SVR

in figure, where the Torque Ripple - Torque plane on the test set is represented 4.34.

Influence on Training size Also for torque ripple, the analysis regarding the importance of the training size has been performed. The methodology that has been followed is the same that has been described in paragraph 4.3.4 for the torque regression.
In this case, the number of training samples used are: 375, 937, 1875, 2812, 3750.
The effect of the different training sizes are reported in Fig. 4.35. Despite the trend

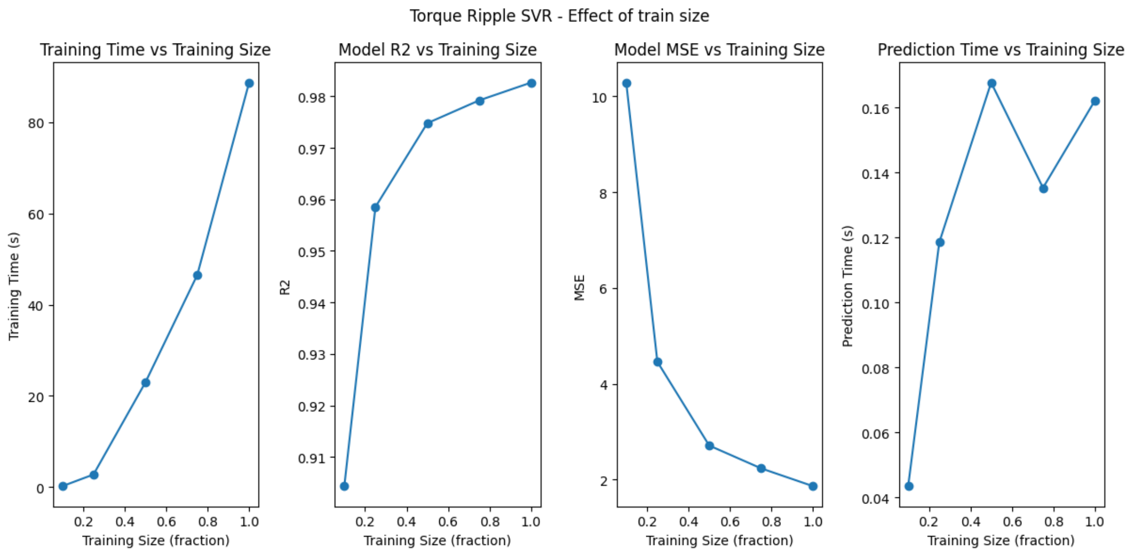


Figure 4.35: Effect of train size on torque ripple regression

being very similar, it is clear how the complexity of the torque ripple regression is higher with respect torque regression. Indeed, looking at the plot describing the

R^2 trend, it is possible to notice how the advantages in having a larger number of training samples is more evident and results in a higher prediction accuracy. This is also the reason why, at the beginning, the split ratio among the train and test subset has been chosen slightly enlarging the training size over the test size.

AdaBoost Due to the inferior performances of the torque ripple model, it is interesting to see if any kind of *ensemble model* can actually reduce the error. Different models where considered (more details in Sec. 4.2.6), such as:

- Bagging Regressor;
- Random Forest;
- AdaBoost;

All these models are requiring a base estimator, and in this case, both decision trees and the optimized SVR are tested. The outcomes are:

- Bagging Regressor is requiring a huge amount of time for the training while the final predictions are practically unchanged with respect the best SVR;
- Random Forest is much faster when trained using decision trees as base estimator, but the final errors are higher compared to optimal SVR;
- AdaBoost is very computationally intensive and the results are just slightly improved.

For the development of this thesis, the results of the AdaBoost model are reported since it may produce some positive effect on the final predictions. In Tab. 4.4, the optimized hyperparameters are reported:

Hyperparameter	base estimator	n estimators	learning rate	loss
Value	Torque Ripple SVR	30	0.011	linear

Table 4.4: Resume of the AdaBoost hyperparameters for torque regression

With this setup, obtained at the end of the optimization process by means of Random Search, AdaBoost model scored $R^2 = 0.974$ on the training set. The total time for the fitting is 608 seconds (almost 6 times higher with respect the optimised torque ripple SVR).

Performances on test set For what regard the prediction on the test subset, the model obtained $R^2 = 0.983$ and a prediction time of 6.88 seconds. The results of the regression are shown in Fig. 4.36. Comparing the results with the SVR regression (Fig. 4.30), it is clear how the AdaBoost advantage is coming from a *redistribution* of the error, proving better results on some points while worsening the predictions on other. Fig. 4.38 and Fig. 4.37 are showing better this trend. On the left part, it is reported the results obtained with the torque ripple SVR regressor while on the right the AdaBoost outcome.

A better comparison is reported in Fig. 4.39, where the relative errors from the SVR are compared with the relative errors of AdaBoost. The presence of the bisector line is used to understand what are the points associated with a higher accuracy and those with a worse prediction. In particular:

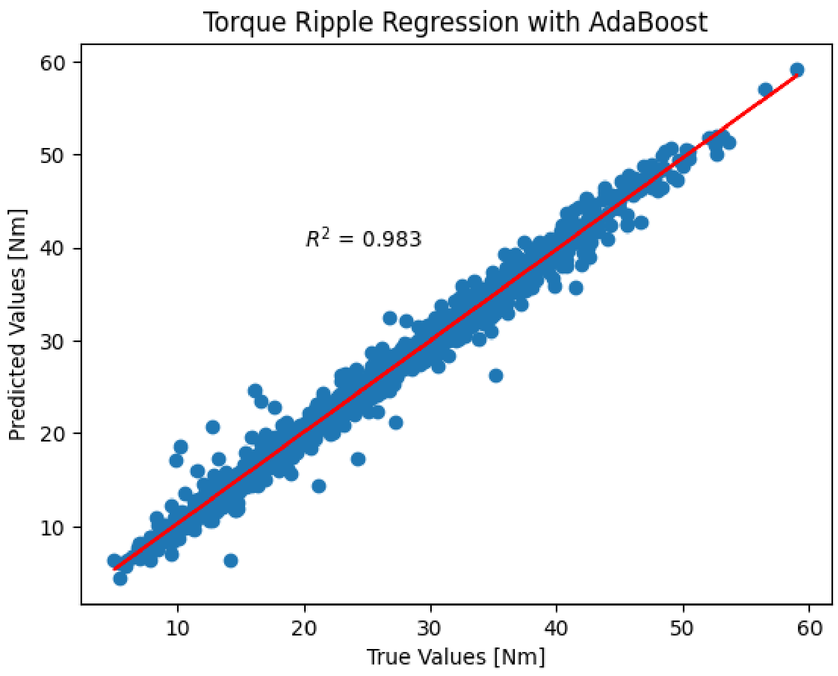


Figure 4.36: Torque ripple regression on test set - AdaBoost

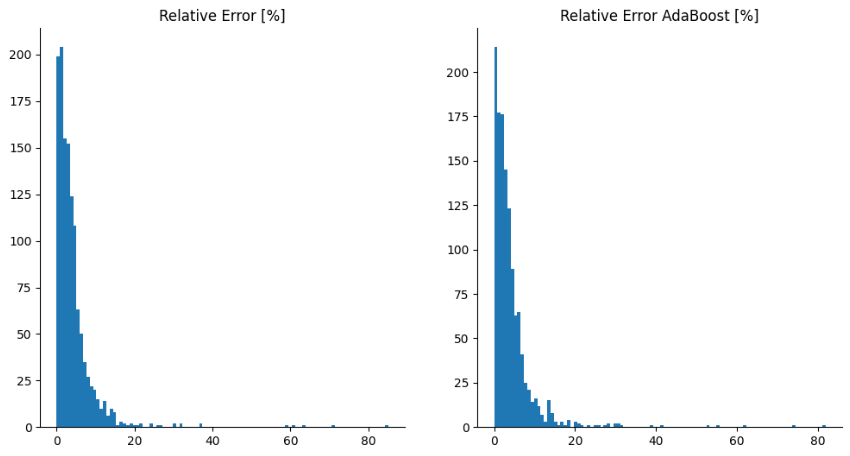


Figure 4.37: Distribution of relative error on test set - SVR and AdaBoost

- Red points are the ones whose prediction error improved after AdaBoost;
- Blue points are the ones associated with higher error after the ensemble model.

The final outcome is that AdaBoost may actually provide better results, and in this case it helped to reduce the higher mis-predicted point at the cost of increasing it in the low error area. In the end it provides a sort of *redistribution of the error* with the downsize of increasing the computational effort required for the model.
Finally the mean squared errors are compared in Tab. 4.5 to actually understand if this ensemble model can be beneficial and to quantify it. In the end AdaBoost provided an increase of the Mean Squared Error of 2.5% starting from a value of 1.86 and reaching the value of 1.82. This shows that even if the AdaBoost model is increasing the robustness of the results, the complexity beneath the torque ripple prediction, still requires further investigation. For this reason, the same analysis

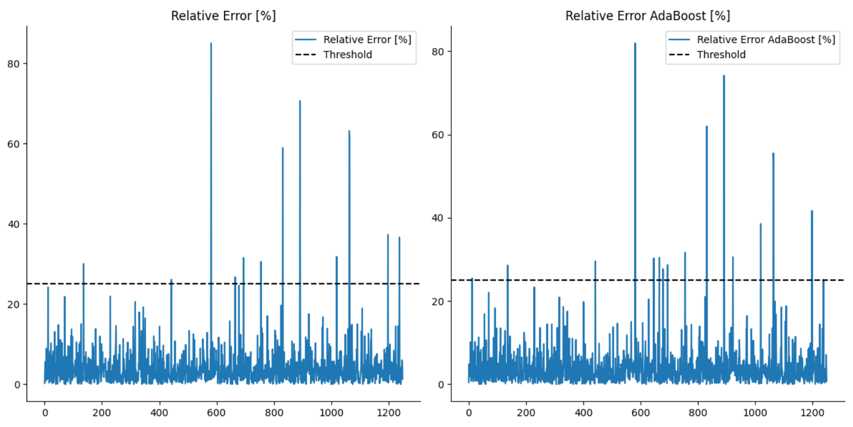


Figure 4.38: Samples with relative error on test set - SVR and AdaBoost

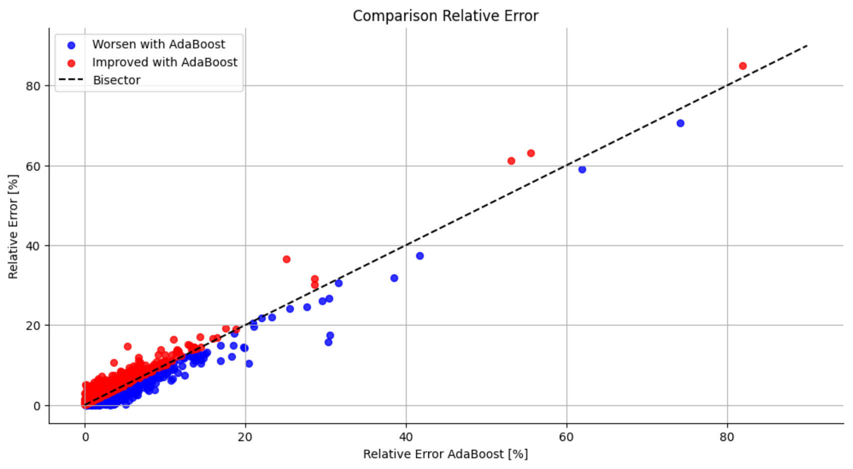


Figure 4.39: Comparison among SVR and AdaBoost relative errors on test set

performed with the Supervised Support Vector Machine is repeated using a Deep Learning Approach.

Model	**MSE Value**
Optimized SVR	1.86
AdaBoost	1.82
AdaBoost Improvement	2.50%

Table 4.5: Mean Squared Error on test set - AdaBoost vs SVR

4.4 Neural Networks

Neural Networks are Machine Learning models inspired by the structure of human brain. Their aim is to mimic the learning capabilities of human being, understanding how to move from a given input to achieve a desired output. They are very powerful tool in the context of Artificial Intelligence as they show very good generalization capabilities. The basic unit is called *neuron* and many neurons can be interconnected to create the network. Since they have a very high degree of customization, they are suitable to solve many complex problems. For this reason, Neural Networks, implemented using *Keras* libraries, are tested to understand if they can be a valid choice at predicting the torque and torque ripple starting from the features representing the geometrical dimensions of the Internal Permanent Magnet (IPM) motor. The results will be compared with the outputs of the Support Vector Machine (SVR), to evaluate which is the best model for the task.

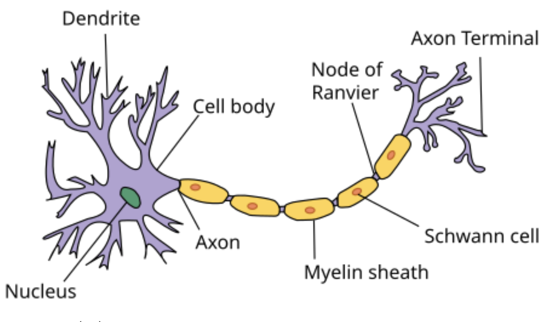
4.4.1 Neuron and Layers

As already mentioned, the basic element of the Neural Network is the neuron with a structure similar to the ones present in human brain, as shown in Fig. 4.40. The main components are:

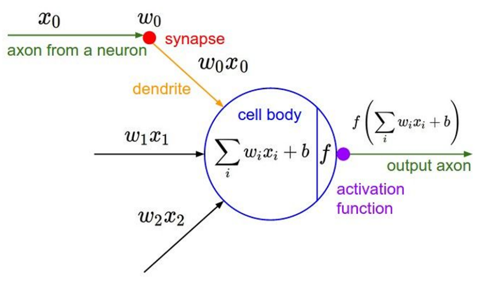
- Inputs (x_0, x_1, x_2, ...): input signals to the neuron. These are from a previous layer or an external input;
- Weights (w_0, w_1, w_2, ...): internal parameters of the model that are adjusted during the learning phase. w_i gives the strength of the associated input x_i. In this way, the network learns which inputs are important to achieve the desired output. From mathematical point of view this is equal to 4.17;

$$\sum_i w_i x_i \tag{4.17}$$

- Bias (b): additional internal parameter learned during training. It allows to shift the weighted sum of the inputs to improve the learning process;
- Activation function: used to introduce non-linearity inside of the model. This is crucial since real world patterns are complex and highly nonlinear. Those are used to decide if the neuron should be activated or not;
- Output: the final outcome is sent to the next layer.



(a) Brain Neuron representation



(b) Neural Network Neuron representation

Figure 4.40: Neuron Analogy [41]

The composition of several neurons is producing a layer. Different layers are creating a Neural Network, as shown in Fig. 4.41. In this example, a Neural Network composed by 4 layers is represented. The layers between the *Input layer* and *Output Layer* are called *Hidden Layers*. Depending on the network architecture, one or more *Hidden layers* can be present. More layers are associated to more parameters, and therefore, more complex network. These types of network are also referred as *Multi - layer Perceptron*.

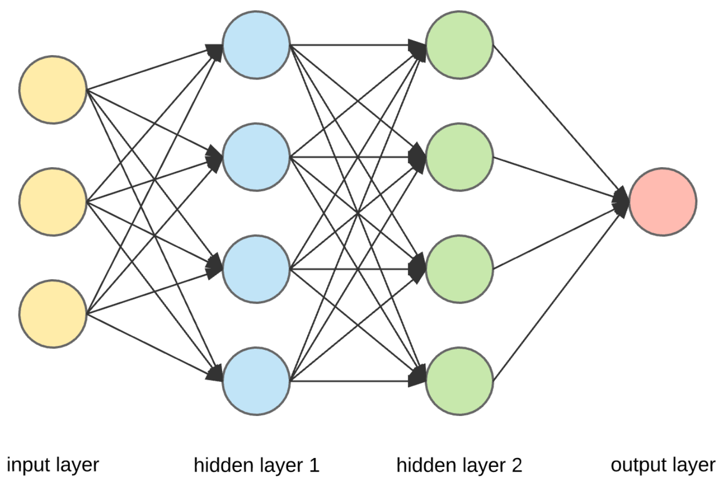


Figure 4.41: Structure of a simple Neural Network [42]

4.4.2 Activation functions

As already mentioned, activation functions are required to introduce some non-linearity inside the Neural Network. This is improving its learning capabilities [43]. In literature many activation functions are described, each with advantages and disadvantage.

Sigmoid function The mathematical equation describing the sigmoid function is reported in Eq. 4.18 while the graphical representation is shown in Fig. 4.42. For every input, the associated output range is [0, 1]. Due to this, it has several drawbacks. Indeed, being the function saturated, it introduces the *vanishing gradient* problem [43] for which the gradient of the objective function becomes really small leading to almost no update in learning process. Moreover, since the outputs are not zero - centered, there may be problems of convergence. Lastly, the exponential function can be expensive to compute.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.18}$$

Tanh function The second activation function analysed is the *tanh* whose expression is reported in Eq. 4.19 and represented in Fig. 4.43. In this case, the output is zero - centered as it belongs to the range [-1, 1]. This is improving the convergence dynamics of the model. On the other hand, it suffers from *varnishing gradient*, being saturated, and also from computational complexity, due to the presence of the exponential.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4.19}$$

ReLU function Rectified Linear Unit function is shown in Eq. 4.20 and in Fig. 4.44. The output are constrained in the range [0, inf]. This comes with different advantages, since it does not suffer from saturation in the positive region, it provides faster convergence with respect *tanh* and *sigmoid* and it is also computationally efficient. It still has the problem of *Dying ReLU* [43] when the input is negative or close

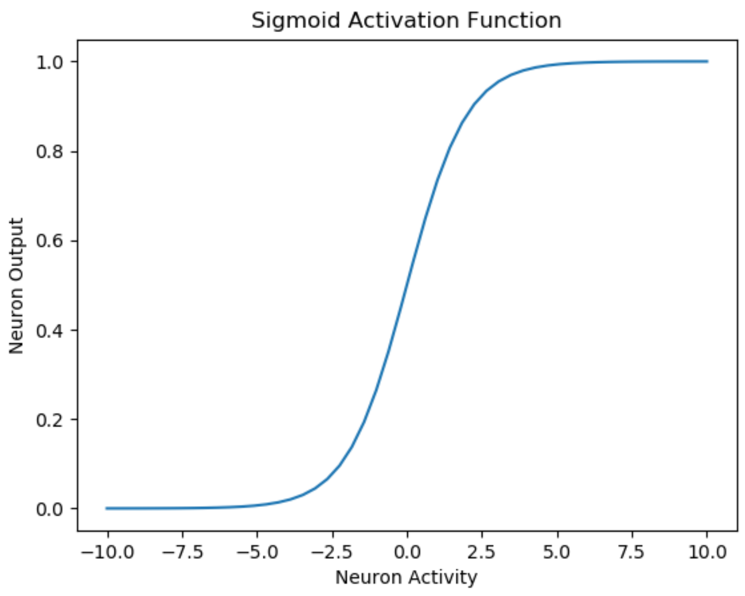


Figure 4.42: Sigmoid activation function [14]

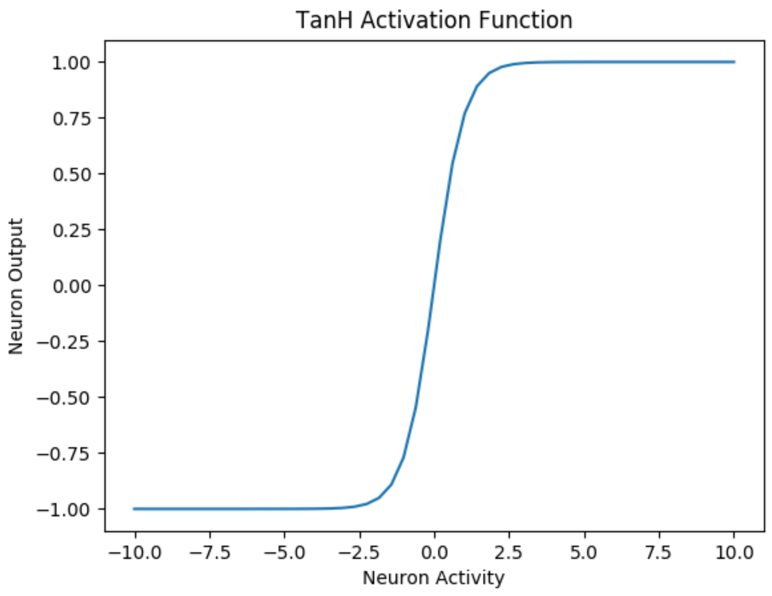


Figure 4.43: Tanh activation function [14]

to zero, with the downsize of *killing* the gradient and thus reducing the learning capabilities of the model.

$$\mathrm{ReLU}(x) = \max(0, x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.20}$$

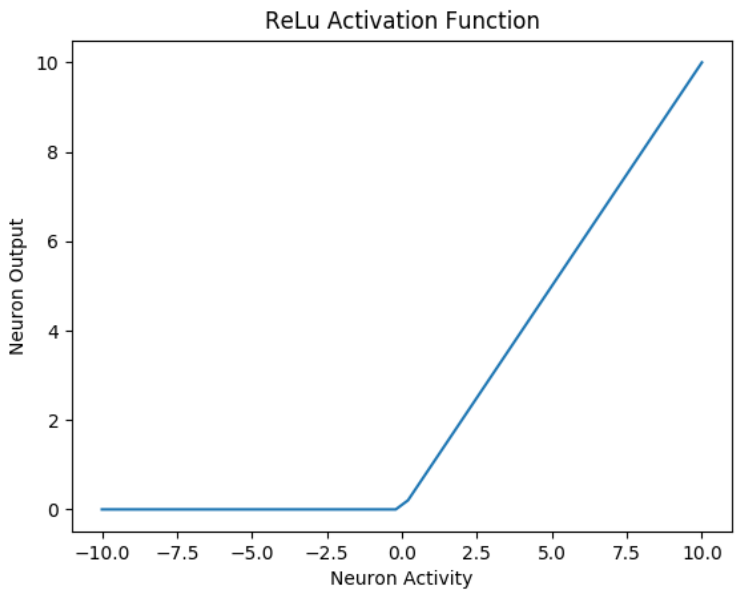


Figure 4.44: ReLU activation function [14]

4.4.3 Training

Training deals with the minimization of the loss function, by learning the internal parameters (weights w_i and bias b) that are producing the lowest error possible. Training process requires different elements to be described, in particular:

- Loss Function;
- Optimizer;
- Back-propagation;
- Epoch and Batch Size.

Loss Function

As the name suggests, loss functions are used in Neural Network to evaluate the losses. The goal of the training is to learn the best parameters (weights w_i and bias b) that are minimizing the loss function, providing the minimum error. Depending on the task that the network has to perform, different loss functions can be adopted. Since this thesis is discussing about regression, the main regression losses are:

- *Mean Square Error [MSE]*: to penalize larger error more heavily, potentially at the expense of smaller errors, in order to achieve a *closer* fit.
- *Mean Absolute Error [MAE]*: to balance the trade - off between a good fit and maintaining robustness to outliers, in order to achieve a more balanced performance across the entire dataset.

More details are given in the corresponding Sec. (4.1.1).

Optimizer

At this point, a loss function is defined. Depending on the number of features that are given in input, the loss function can be described in a very high dimensional space where each dimension corresponds to a model parameter. For example, Fig. 4.45, shows the shape of a generic loss function that depends on two different model parameters (x-axis and y-axis), while the loss is on the z-axis. For each combination of x and y, a value of loss is produced. The function features multiple peaks and multiple valleys, therefore it is *non - convex*, as it comes with multiple local minima and local maxima. The aim of the optimizer is to update the parameters of the model such that the combination of them is producing a minimum in the loss function. The optimal combination would lead to the global minimum of the function.
During training, the model is learning how to update the parameters starting from a random point on the loss function curve to minimize the errors. The update is done every epoch, which represents the number of times that the network *sees* all the training set.
It is clear how this already complex scenario, can be further complicated as the model parameters are increasing. Therefore, the choice of the optimizer is crucial. In literature many optimizers are present [44]. For this thesis, two main approaches are evaluated:

- Stochastic gradient descent (SGD);
- Adaptive Moment Estimation (ADAM);

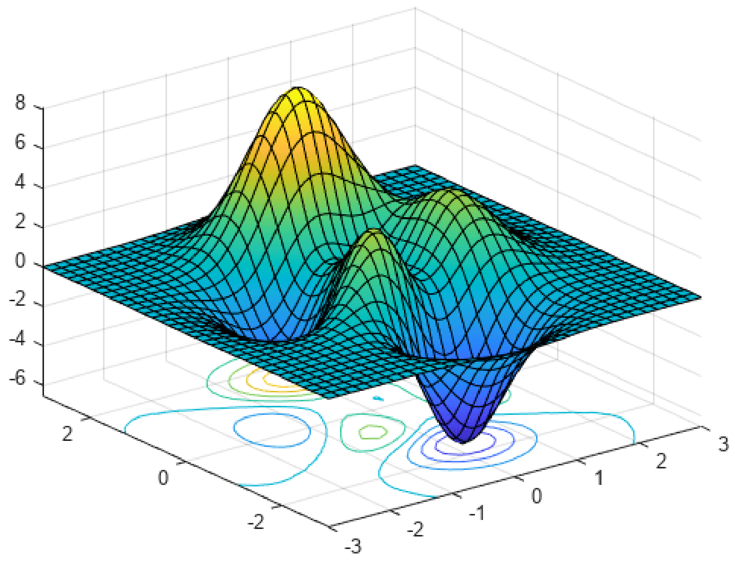


Figure 4.45: Loss function example [14]

Stochastic gradient descent (SGD) The optimization problem deals with the evaluation of the minima of the loss function. It is required therefore to evaluate the gradient of the loss, using a mini-batch of training set (to speed up the process), to understand what is the direction pointing toward a minimum. Then the internal weights θ (vector composed by w_i and b) are updated following Eq. 4.21:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta J(\theta_t) \tag{4.21}$$

This updating process is repeated until convergence. A special mention is given to the hyperparameter α that represents the *learning rate* and it is used to control the step size of the parameters updating. The choice of this value is critical, as shown in Fig. 4.46: In particular:

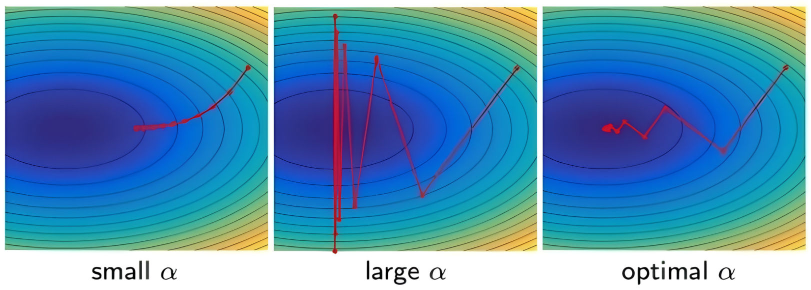


Figure 4.46: Effect of learning rate on the update of internal parameters [14]

- Small α can lead to a slow convergence speed, requiring larger time to reach the minimum;
- Large α produces overshooting, for which the solution is bouncing around the minimum;
- Optimal α has to be searched with some hyperparameter tuning process.

SGD provides a memory efficient (since it works with mini-batches) and easy to implement optimization algorithm, but on the other hand it may get stuck in local minima or saddle point. To solve this problem, formulation taking into account for *momentum* are considered. This is used to speed up convergence by considering the past history of the gradients in the update process. The mathematical expression is given in Eq. 4.22 and it is used to update the velocity.

$$v_{t+1} = \lambda v_t + (1 - \beta)\nabla_\theta J(\theta_t) \tag{4.22}$$

Then the velocity is used to update the parameters as reported in Eq. 4.23.

$$\theta_{t+1} = \theta_t - \alpha v_{t+1} \tag{4.23}$$

Adaptive Moment Estimation (ADAM) The concept of momentum is also explored in Adaptive Moment Estimation (ADAM) optimizer. In particular, it leverages on adaptive estimation of first-order (the mean) and second-order (uncentered variance) moments to adjust the propagation speed by dynamically evaluating the learning rate. This means that, unlike SGD, ADAM is adapting the learning rate for each parameter individually. In particular 4.24 shows the implementation of the two moments:

$$\begin{aligned} m_{t+1} &= \beta_1 m_t + (1-\beta_1)\nabla_\theta J(\theta_t) \\ v_{t+1} &= \beta_2 v_t + (1-\beta_2)\nabla_\theta J(\theta_t)^2 \end{aligned} \tag{4.24}$$

Then these are corrected (Bias-Corrected Estimates) to counteract the fact that initially the moment estimates are biased toward zero. This improves the stability of the algorithm. β_1 and β_2 are the two hyperparameters in charge of controlling the behaviour of the estimation.

$$\begin{aligned} \hat{m}_{t+1} &= \frac{m_{t+1}}{1-\beta_1^{t+1}} \\ \hat{v}_{t+1} &= \frac{v_{t+1}}{1-\beta_2^{t+1}} \end{aligned} \tag{4.25}$$

Finally the update of the parameters is occurring (4.26).

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}} + \epsilon} \tag{4.26}$$

By leveraging on these aspects, ADAM is able to provide smoother and more consistent convergence compared to SGD, making it suitable for many applications. In the development of the thesis, both SGD and ADAM are tested. The final outcome is that ADAM optimizer requires less epoch to reach convergence, also ensuring to obtain smaller errors.

Back-propagation

Keeping in mind all the aspects that have been discussed in this section, it is possible to introduce the actual traning mechanism that is used by the network to actually learn from the data.
The term *back-propagation* [45] refers to the algorithm used to update the internal parameters of the model by computing the gradients of the loss function in order to minimize the difference between the label and the predictions. It is therefore an efficient application of the chain rule in the Neural Network context. The training process is composed of the following key phases:

- *Forward Pass*: this refers to the propagation of the input from the input layer till to the output layer. Each neuron applies the linear transformation followed by the non-linear activation function, according to the network architecture (i.e. Fig. 4.41). Initially, the weights of each neuron are set randomly. At the end the prediction is compared with the label to quantify the loss.

- *Back-propagation*: as the name suggest, the flow of information starts from the output layer and moves back to the input layer. In this phase, for each layer, the gradient of the loss with respect the weights is computed. This involves differentiating the loss function considering the activation function used for the specific neuron. The gradient gives information about how much each weight is contributing to the final loss, and therefore it gives insights of the direction in which the which the weights should be adjusted to reduce the loss. The gradient is propagated through the network using the chain rule of calculus, which is crucial for updating the weights efficiently.

- *Parameter update*: this phase involves the optimizer that is used as update policy for the weights based on the gradients computed during back-propagation. Example of optimizer are discussed in Sec. 4.4.3. This process of forward pass, back-propagation, and parameter update is iteratively repeated over multiple epochs to progressively reduce the loss, ideally resulting in predictions aligning more closely with the true labels.

Epoch and Batch Size

Before moving to the discussion of the experimental results, it is necessary to introduce the concept of *epoch* and *batch size*. Both terms are hyperparameters used to control the training dynamics of a Neural Network model. In particular:

- *Epoch*: it refers to one complete pass trough the entire training dataset. In particular, at the end of the epoch, the network has processed all the training samples, performing the forward phase, the backward phase, and the update of the internal weights. Generally the training process is requiring multiple epochs to minimize the loss function. It is really important to set this value correctly to fully exploit the network potential, avoiding both overfitting and underfitting.

- *Batch size*: it refers to the number of training samples that are used in one forward and backward pass. The training subset is divided into smaller batches, as shown in Fig. 4.47, each one composed by a *batch size* number of samples. The internal weights are updated after one batch is processed, therefore this hyperparameter is linked to the convergence dynamic of the model.

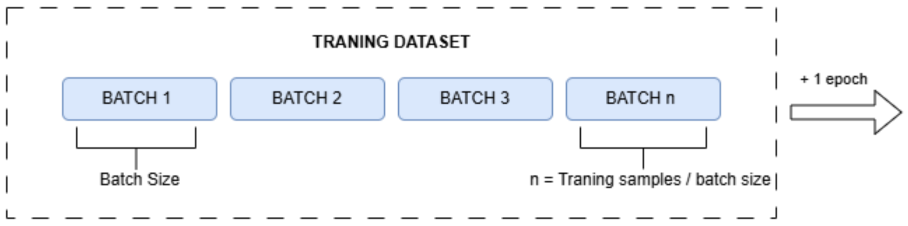


Figure 4.47: Epoch and batch size relation

4.4.4 Experimental Results

This part is dedicated to the presentation of the results obtained for the torque and torque ripple regression using Neural Networks. The implementation has been done with *Keras* libraries [46] that are coming with some predefined function useful for the definition of a Network. In this case, the same training-test-validation split has been used both for torque and torque ripple regression. The ratio is shown in Fig. 4.48. This is corresponding, considering the total set of 5000 samples, to: train subset 3200 samples, validation subset 800 samples and test subset 1000 samples. With respect the SVR models, the presence of the validation set is required since it will be used as a metrics to check the performances of the model during the learning process. This is necessary to ensure that no overfitting or underfitting are affecting the process.

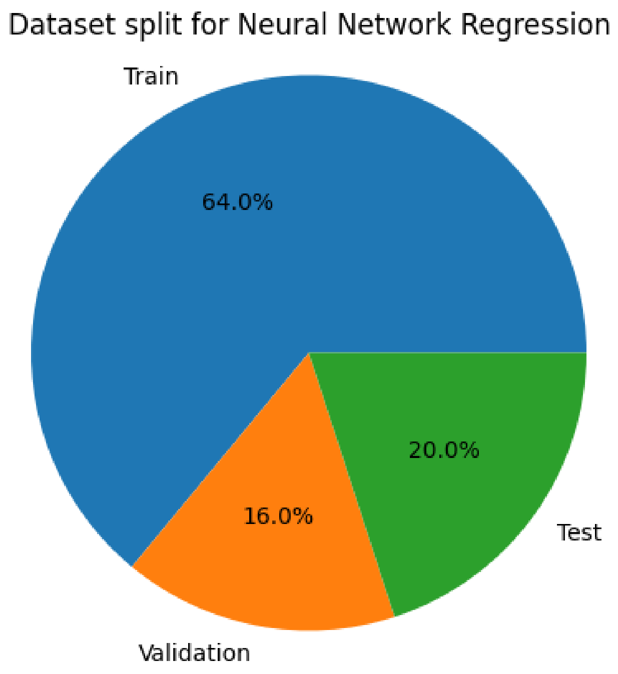


Figure 4.48: Dataset Split For Neural Network

In both cases, the structure of the network has been optimized using *Keras Tuner*. The design variables considered are:

- The number of neurons for each layer;
- The activation function for each layer;
- The number of layers;
- The Learning Rate;
- The possibility of having dropout layers for regularization purposes;

This optimization process required some computational time, but it allowed to identify the best combination of the parameters and hyperparameters useful to achieve the minimum error. More details about the structure of each Network are given in following sections.

Torque analysis

The first model is the Neural Network for torque regression. In this case the network has been designed using the *Keras Random Search Tuner* which is based on the *Random Search* algorithm (Sec. 4.2.4) on a maximum number of iterations equal to 30. Once the best configuration has been evaluated, the torque regression network has been created. The results are shown in Fig. 4.49.

The Network architecture is reported in Tab. 4.6:

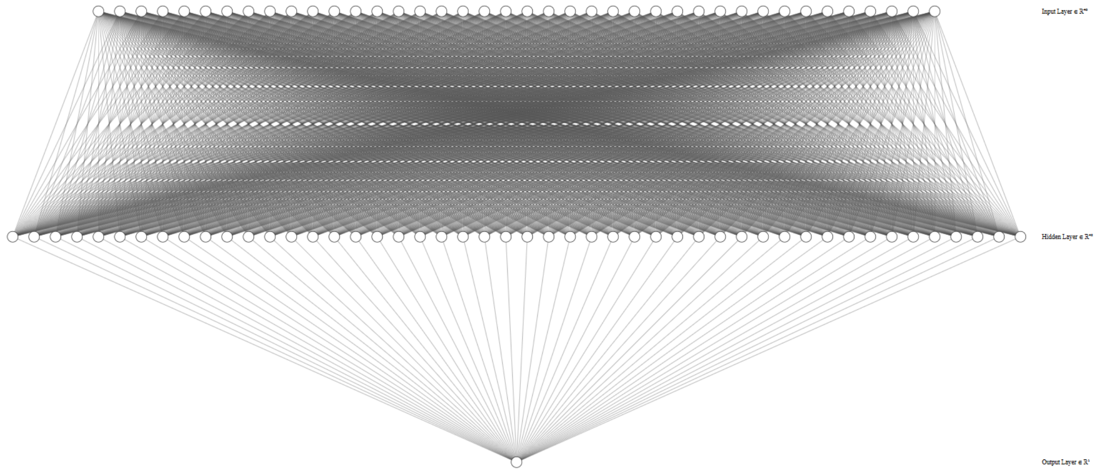

Input Layer $\in \mathbb{R}^{40}$
Hidden Layer $\in \mathbb{R}^{48}$
Output Layer $\in \mathbb{R}^{1}$

Figure 4.49: Torque regression Neural Network representation

Layer (type)	**Output Shape**	**Activation Function**	**Parameters #**
Input (Dense)	(None, 40)	tanh	360
Hidden (Dense)	(None, 48)	relu	1968
Output (Dense)	(None, 1)	linear	49
Total parameters:			2377 (9.29 KB)
Trainable parameters:			2377 (9.29 KB)
Non-trainable parameters:			0 (0.00 Byte)
Loss function:			MSE

Table 4.6: Torque regression Neural Network Layer Summary

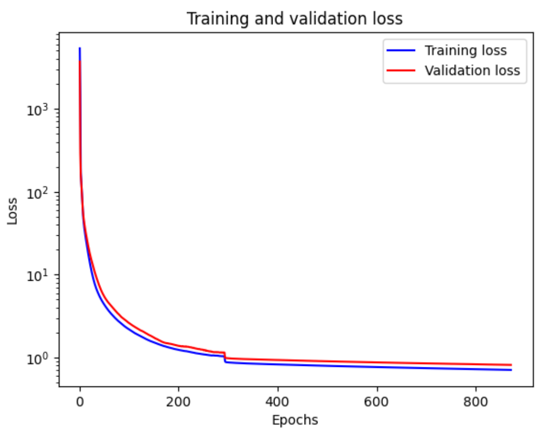
The Network is composed by 3 layers with respectively 40, 48 and 1 neurons. The necessity of having just a neuron in the output layer is justified by the fact that the desired output is just a value (i.e. torque in this case). The total number of parameters to learn is 2377. A lower number indicates a simpler network, while the complexity increases as # parameters is increasing. This has to be considered as a trade-off among the complexity of the network, the prediction's error and the risk of overfitting. Before discussing the results, some details about the training process are given. In particular, Fig. 4.50a and Fig. 4.50b are reporting the validation curves obtained in the learning phase. Some considerations can be done:

- The training and the validation loss are decreasing together, with the validation curve that *mimics* the shape of the training loss. This behaviour is indicating that the model is well performing, without any risk of overfitting or underfitting;

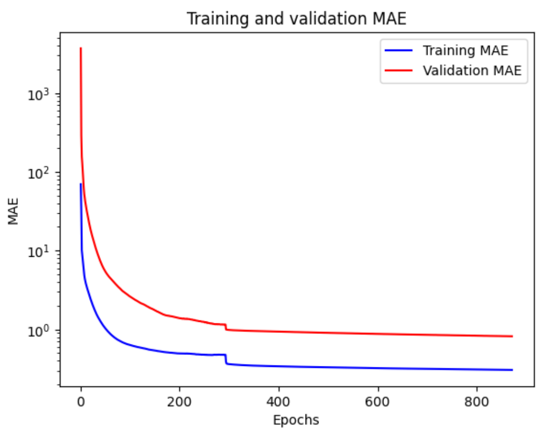
- 871 epochs are required before the stopping criterion is met. The total training process lasted 2401.47 seconds;

- Training and validation Mean Absolute Error are decreasing at the same time, but the second one is showing higher values;

- In terms of learning rate, at the beginning it is fixed to $1e-3$ and drops to $1e-4$ around epoch 350. ADAM is used as optimizer;
- The current results are obtained with a batch size equal to 16.



(a) Torque regression Neural Network - Training and Validation Loss



(b) Torque regression Neural Network - Training and Validation MAE

Figure 4.50: Validation Curves - Torque Neural Network

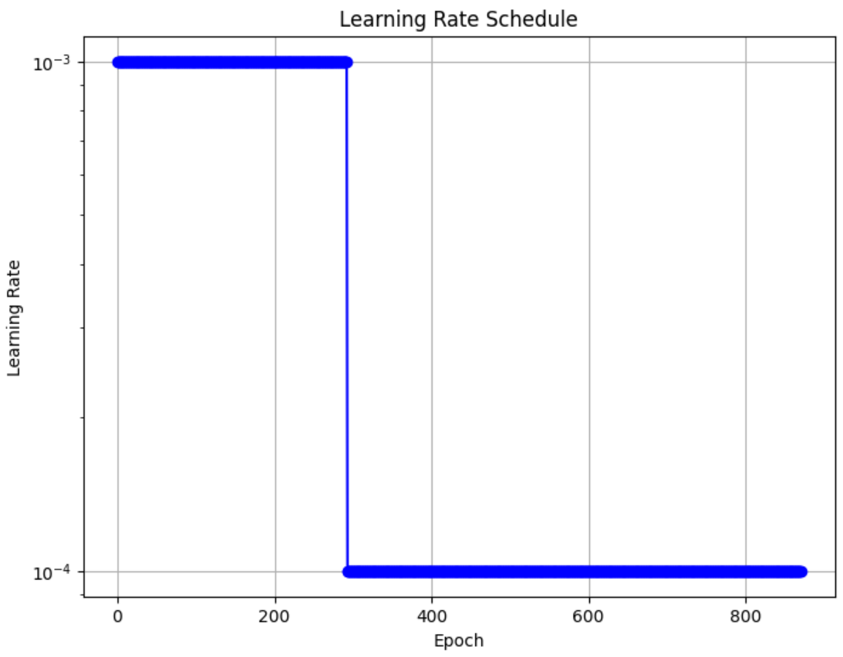


Figure 4.51: Torque regression Neural Network - Learning Rate scheduler

This setup reached a mean squared error on validation set of 0.26 and a mean absolute error on validation set of 0.37.

Performances on test set Once the model is trained it is possible to evaluate its performances on the test set. Fig. 4.52 shows the torque regression between the true values (on x-axis) and the predicted values (on the y-axis). As already mentioned, the red line is indication the regression line. Looking at the distribution of the points, it is clear how the Neural Network model is showing excellent performance at predicting the torque starting from the geometrical dimensions of the IPM motor. Indeed, blue points are lying very close to the red line. The obtained R^2 is 0.999, showing the the model is capturing very well the variability of the test set. To link

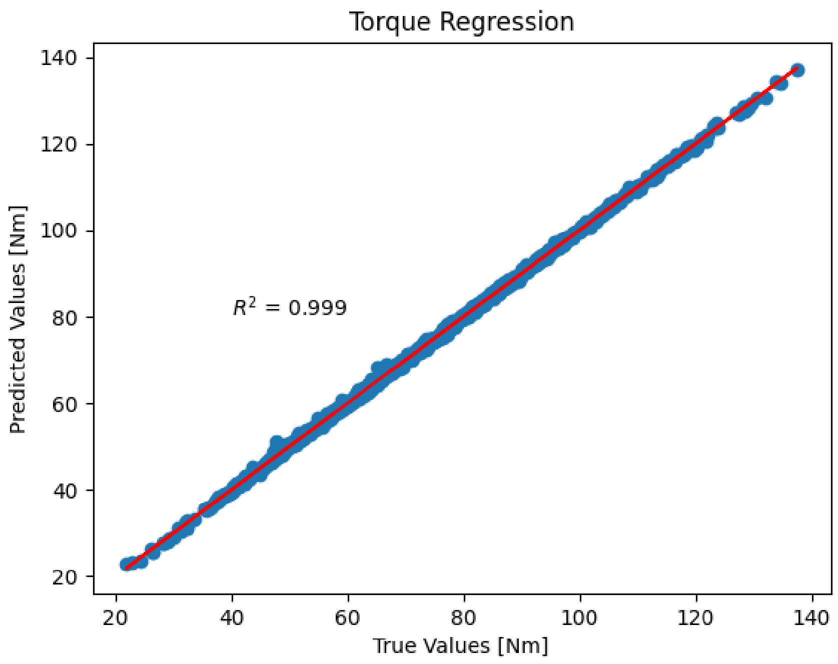


Figure 4.52: Torque regression on test set - Neural Network

the relative error of the predictions with the true value, Fig. 4.53 is reported. With respect the SVR model, the predictions are improved, showing a maximum relative error slightly above 7% (compared to 10% of SVR). The majority of the outputs are below 2%. In terms of trend, also in this case, the higher errors are occurring on the low torque area [0, 80] Nm.

The same results can be obtained looking at Fig. 4.54, showing the distribution of the relative error. The current setup is therefore able to predict almost the majority of the test point with a suitable error. Lastly, Fig. 4.55, shows the correlation among the samples of the test set and their relative error. The 10% threshold has been set manually and it is used to visually see the difference of performances with the SVR model. In this case, none of the points is exceeding this limit.

Influence on Training size It is interesting to understand how much the performances are affected by the size of the training subset. Also in this case, different train sizes are tested keeping the dimension of the validation subset and test subset fixed. This has been done to have a fair comparison among the different candidates. The goal is to understand what is the minimum number of training samples that can be used to obtain satisfactory results. In particular, the tested train sizes are 10%, 25%, 50%, 75% and 100% of the train subset described in Fig. 4.48. Looking at Fig. 4.56, it is possible to understand that the two evaluation metrics (R^2 and MSE) are becoming almost flat at 75% of the test set. This point is also associated with a lower training time with respect the next configuration, being the number

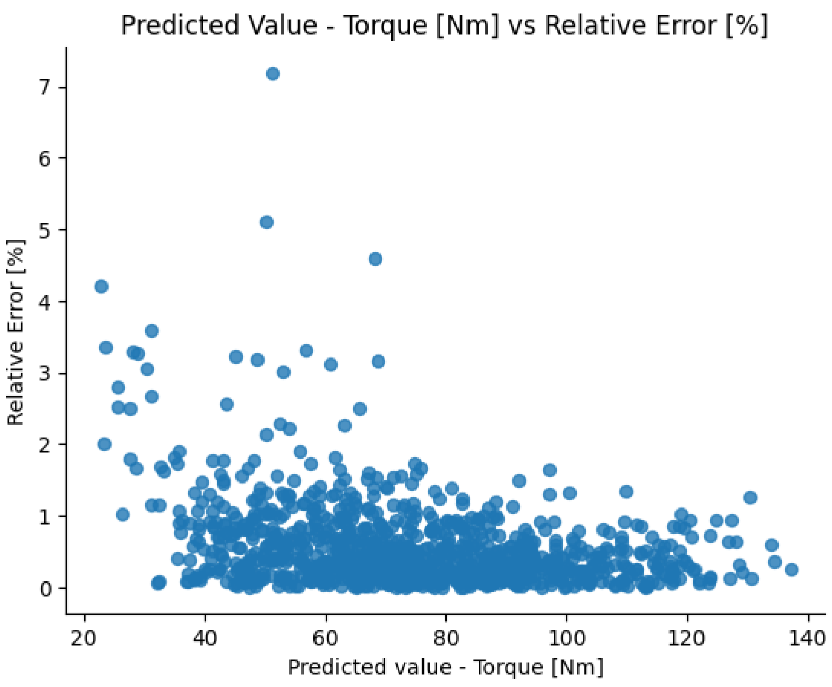


Figure 4.53: Torque prediction with relative error on test set - Neural Network

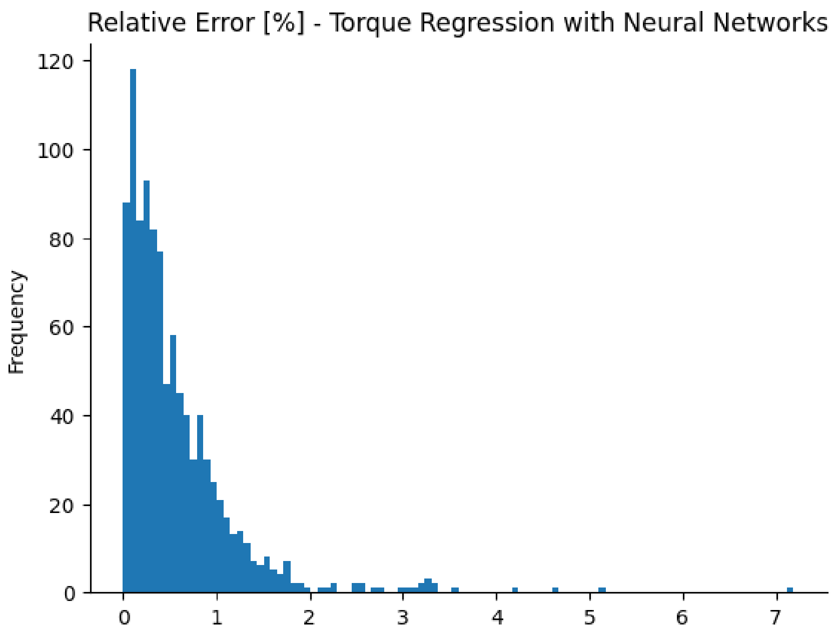


Figure 4.54: Distribution of relative error on test set - Torque Neural Network

of samples lower. This means that if optimization of the computational resources is a concern, the 75% model could be used to obtain still accurate results, requiring less time for the training. In the same way, also the maximum number of epoch for the 75% configuration is lower with respect the 100% test size. Also the validation curves for the different test setup are reported in Fig. 4.57 to ensure that none of them is overfitting. The main outcome is that as the number of training samples is increasing, the training and validation curves are becoming closer, meaning that the performances of the network are increasing. Moreover, the configuration with

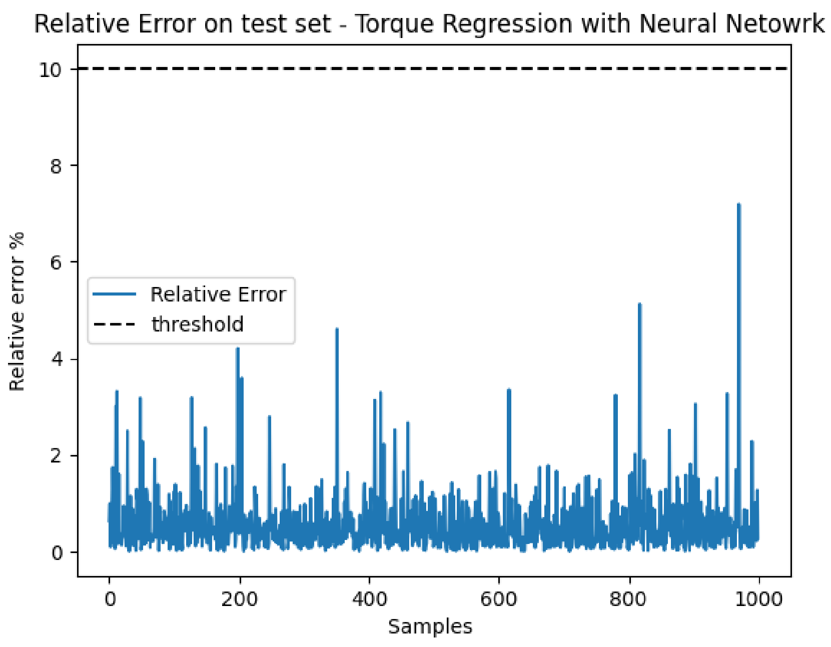


Figure 4.55: Samples with relative error on test set - Neural Network

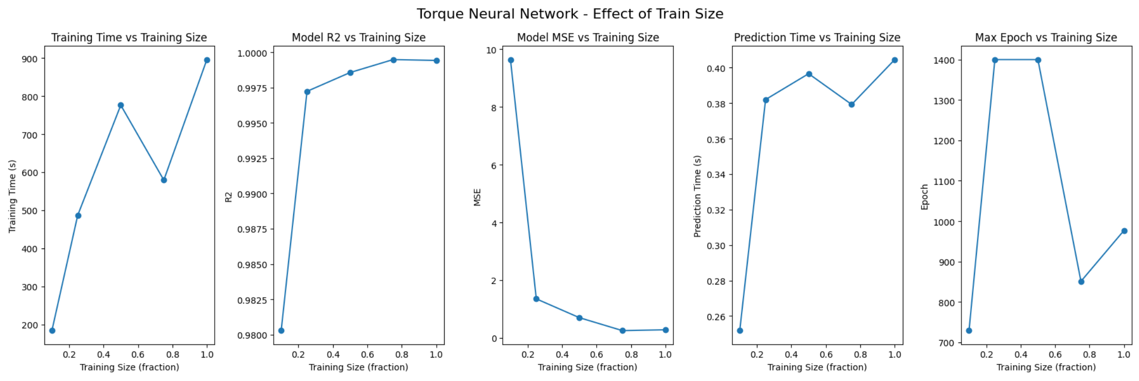


Figure 4.56: Effect of train size on torque regression - Neural Network

320 samples (10%) is the one in which overfitting may occur. Indeed, it is possible to observe how without the adoption of a stopping criterion, the training loss would decrease more, while the validation curve is almost flat. This means that the network is gaining knowledge on the training samples, but this is not useful at prediction stage.

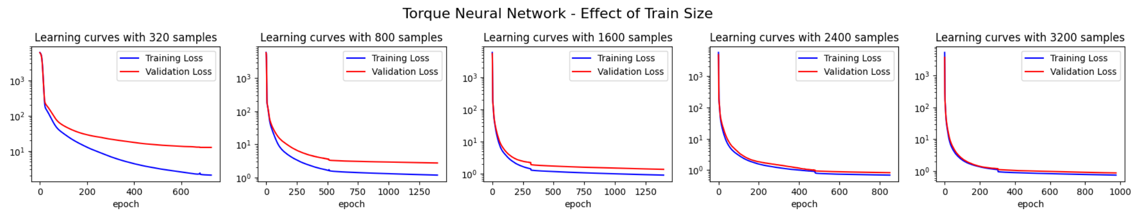


Figure 4.57: Effect of train size on torque regression - Neural Network Validation Curve

Influence of Batch Size Another interesting tuning factor is related to the batch size. As already mentioned, batch size controls the number of samples that are processed at each time, and consequently how frequently the model weights are updated. Fig. 4.58 reports a sensitivity analysis regarding the effect of different batch sizes. In particular, it is shown how the 16 batch size configuration is associated with the highest R^2 and the lowest MSE. This setup is not the most optimized in terms of training time, prediction time and maximum number of epoch since the it requires 200 iterations to complete an epoch. In any case, since the computational effort is not a concern, it has been chosen as the best one, since it lead to the highest accuracy. From Fig. 4.59 it is evident that smaller batch sizes are associated with

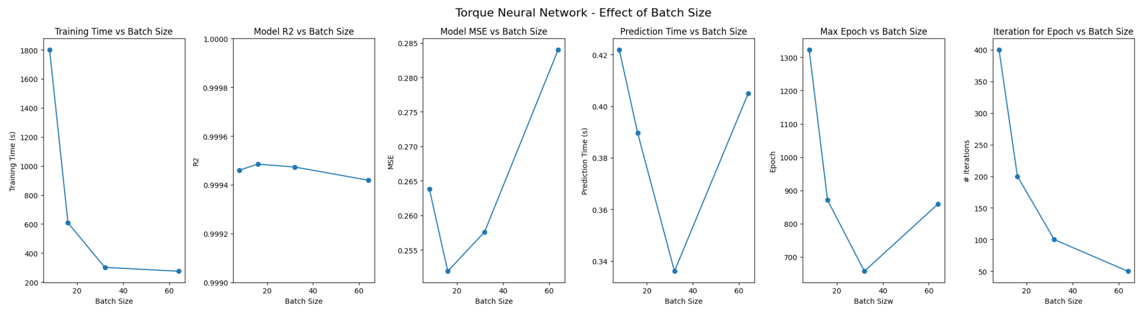


Figure 4.58: Effect of batch size on torque regression - Neural Network

faster convergence. This is because smaller batches lead to more frequent weight updates within each epoch, resulting in a larger number of iterations and allowing the model to adjust its parameters more quickly. However it in important to notice that this behaviour is influenced by several factors like the adopted learning rate, the number of layers and the activation function, that together are contributing to the overall convergence of the model.

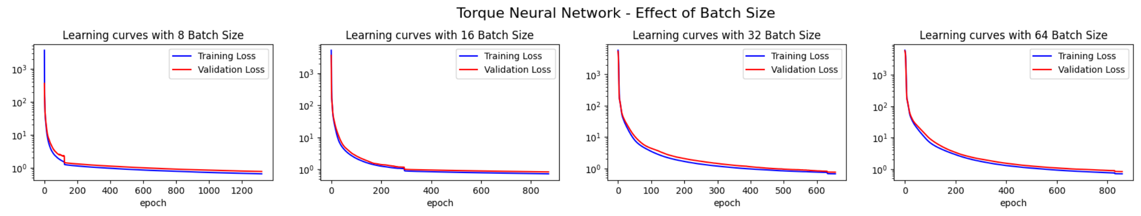


Figure 4.59: Effect of batch size on torque regression - Neural Network Validation Curve

Torque ripple analysis

The second prediction output is the torque ripple. In particular, direct values are used to train the model. Also in this case, the optimization procedure has been performed using *Keras tuner* library, in particular with the *Hyperband optimization* framework [47]. This procedure allows to gradually allocate more computational power to the most promising configurations. In particular, it starts training for few epoch a large amount of models, comparing the results among them. The best ones are trained for additional epochs, till converging to the best model. By doing so, computational resources are focused on the configurations with the highest potential, avoiding to waste time and power on the sub-optimal solutions. The whole tuning process lasted for 6 hours, but it gives a deep exploration of the architecture and hyperparameter space of the models.

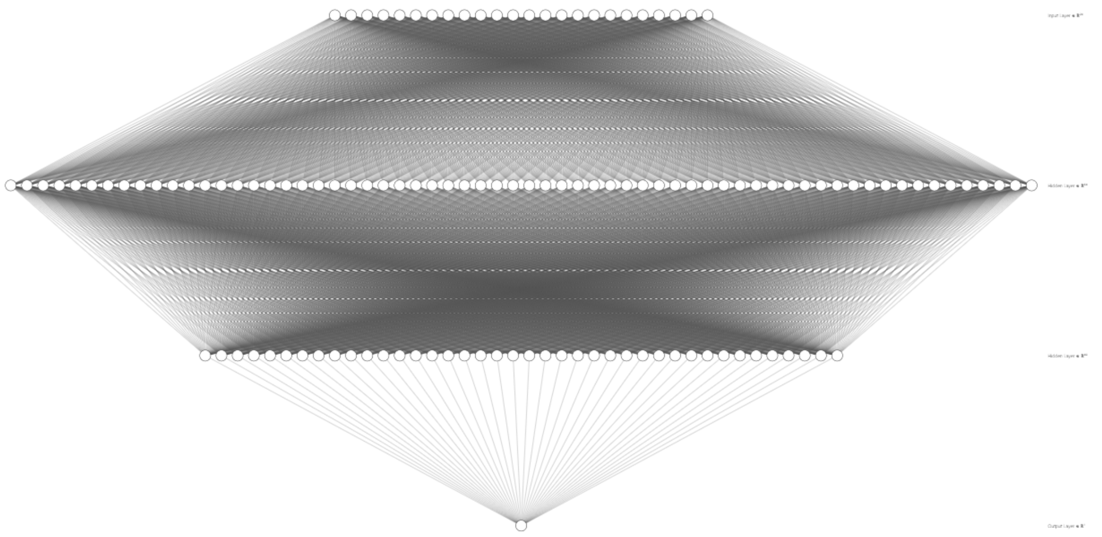

Input Layer $\in \mathbb{R}^{24}$
Hidden Layer $\in \mathbb{R}^{64}$
Hidden Layer $\in \mathbb{R}^{40}$
Output Layer $\in \mathbb{R}^{1}$

Figure 4.60: Torque ripple regression Neural Network representation

The architecture of the optimized network is reported in Fig. 4.60 and in Tab. 4.7. In this case, an extra layer is required to achieve satisfactory results. This is leading to a more complex model with a larger number of trainable parameters. This compromise is anyway accepted as it leads also to an increase of accuracy of the final model.

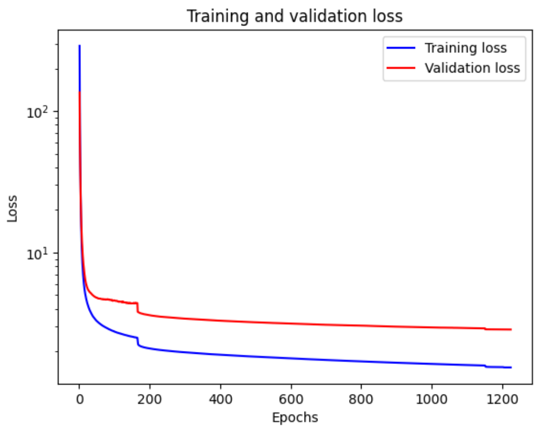
Layer (type)	**Output Shape**	**Activation Function**	**Parameters #**
Input (Dense)	(None, 24)	tanh	216
Hidden 1 (Dense)	(None, 64)	relu	1600
Hidden 2 (Dense)	(None, 40)	tanh	2600
Output (Dense)	(None, 1)	linear	41
Total parameters:			4457 (17.41 KB)
Trainable parameters:			4457 (17.41 KB)
Non-trainable parameters:			0 (0.00 Byte)
Loss function:			MSE

Table 4.7: Torque ripple regression Neural Network layer summary

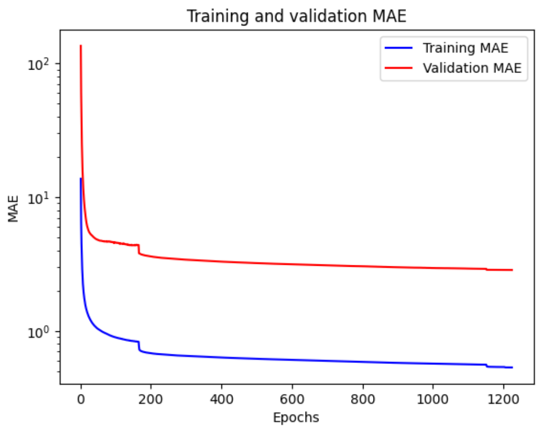
From Fig. 4.61 it is possible to make some considerations about the training process. In particular:

- The model is able to learn from the training set and to make predictions of the validation set without overfitting;
- Due to the shape of the curves, it is clear how some variability on the validation set is still present, much more than in torque prediction case;

- The total process required 1224 epochs and a total time of 1771 seconds;
- As already occurring for torque, also in this case a learning rate scheduler has been used to increase the speed of convergence ensuring to reach the best performances in a suitable time. In particular as shown in Fig. 4.62, 3 drops are performed, starting from $1e-3$ reaching $1e-6$. A larger learning rate is associated to a faster convergence but it also comes with the possibility of bouncing around the minimum. For this reason, a progressive decreasing of the hyperparameter value is adopted to avoid this effect, reaching the real (global or local) minimum. Also in this case, ADAM is used as optimizer;
- A batch size equal to 8 has been used.



(a) Torque ripple regression Neural Network - Training and Validation Loss



(b) Torque ripple regression Neural Network - training and validation MAE

Figure 4.61: Validation Curves - Torque ripple Neural Network

Performances on test set The prediction capabilities on test set are now assessed. The trained model is able to reach a $R^2 = 0.985$ as shown in Fig. 4.63, which is slightly improved with respect the results obtained by the SVR. In this case, the points are distributed closer to the red regression line, even if there are some mispredicted point. With the current setup, the Mean Squared Error is 1.71 while the Mean Absolute Error is 0.95, both evaluated on the test set. To associate the relative error, with respect to the true label, Fig. 4.64 is reported. In particular, it is possible to notice how the majority of the predictions are below a relative error of 20%. Few points are in the 20% - 60% range zone and just a point is close to almost 100%, producing a prediction that is double with respect the true value. This is anyway accepted due to the complexity beneath torque ripple prediction and the overall performances on the test set. To further stress this concept, Fig. 4.65 is used to give an idea about the number of samples predicted within a certain error. Almost all the test set is predicted with a relative error below 10%. This makes the model quite usable even if some predictions are coming with a really high relative error. Indeed the goal of the ML model is not to completely substitute the FEA analysis but just to provide a way to faster explore the design space, and later on validate the results with FEA, refining the design. The whole process has to be

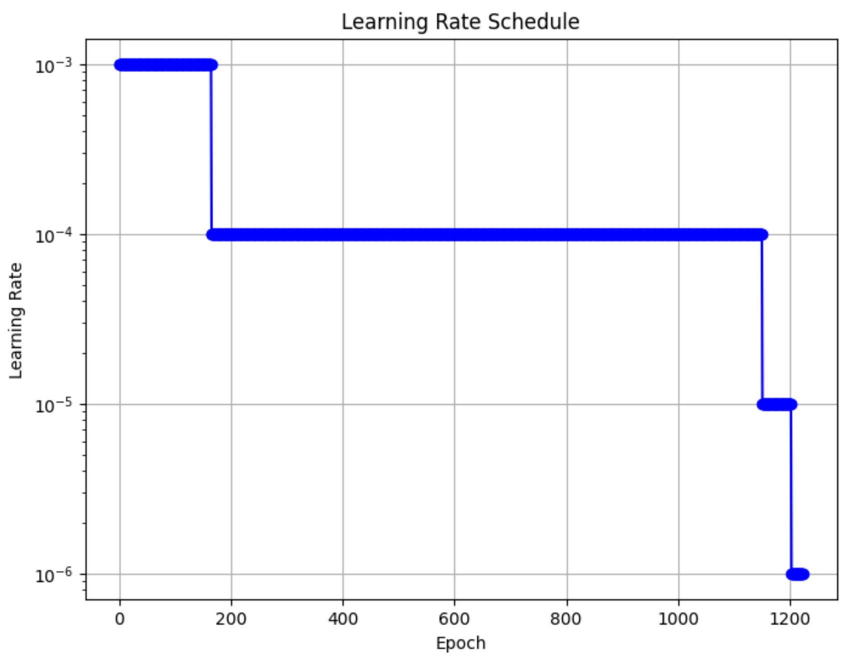


Figure 4.62: Torque ripple regression Neural Network - Learning Rate scheduler

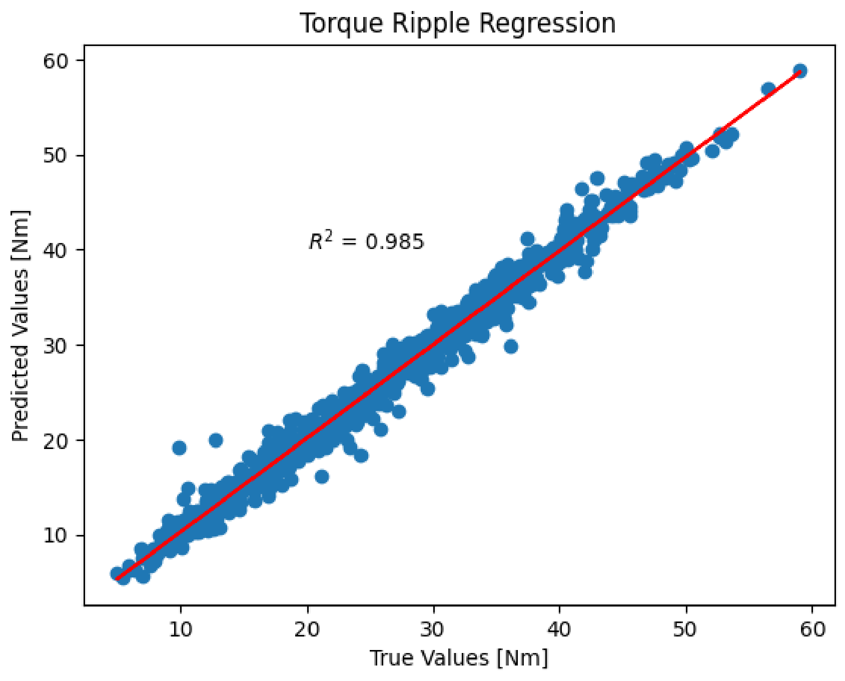


Figure 4.63: Torque ripple regression on test set - Neural Network

intended as a way to enhance the design capabilities and not as a way to achieve the best final combination of features without any further refinement. Lastly Fig. 4.66, reports the relative error for each sample of test set. A threshold equal to 25 % has been considered, to make a comparison with the SVR model. With this setup, just 5 configurations are above the limit. The location of these points in the Torque - Torque Ripple plane is reported in Fig. 4.67. It is interesting to try to understand why these points are mispredicted. In order to provide an explanation Fig. 4.68

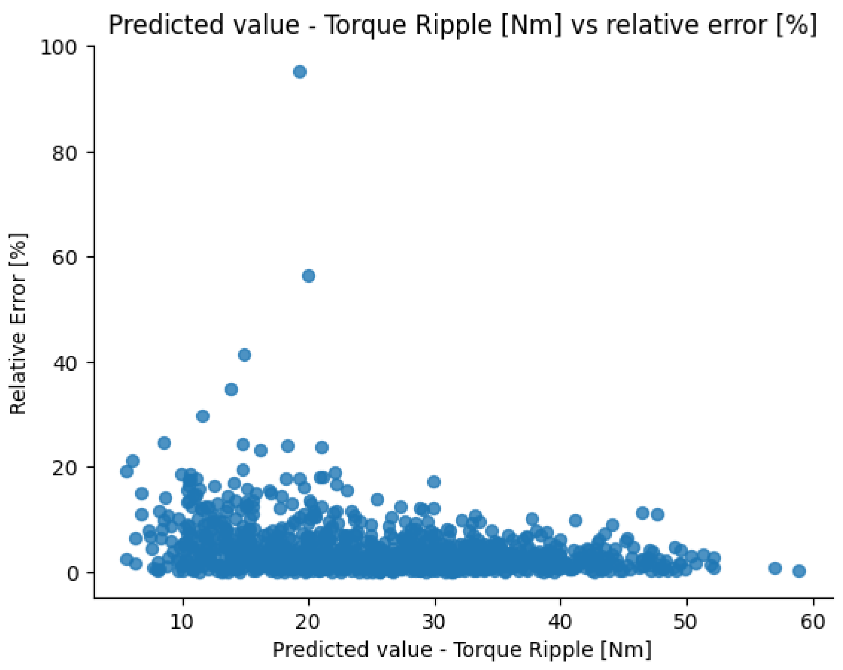


Figure 4.64: Torque prediction with relative error on test set - Neural Network

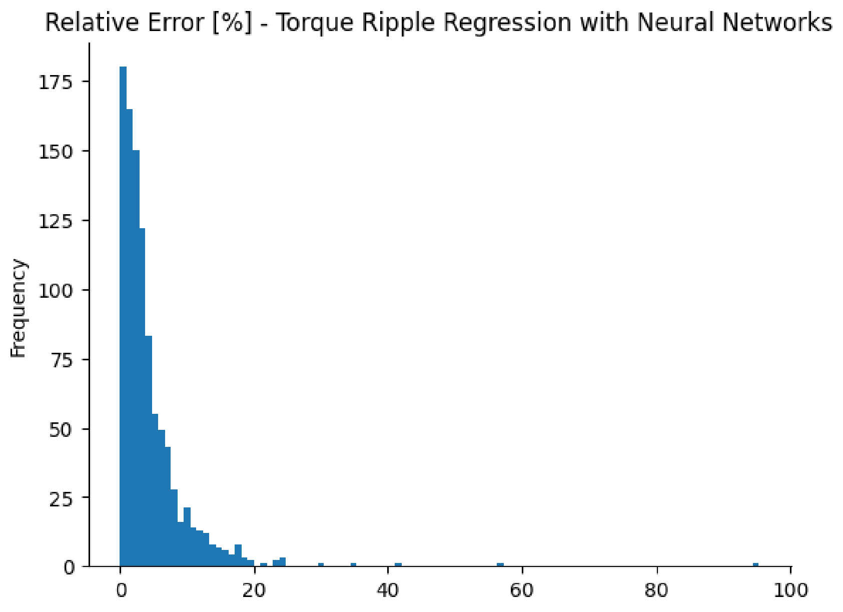


Figure 4.65: Distribution of relative error on test set - Torque ripple Neural Network

reports the distributions of every pre-processed feature variable (in purple) and the corresponding values of the configurations with a relative error higher than 25% (here referred as outliers). Each variable has a near-normal distribution, and it is evident that these outlier configurations belong to regions with the lowest data density. Those values are significantly deviating from the main data distribution and for this reason, the model is not able to predict their labels with a low relative error associated. Since they are just 5 points, the overall effect on the whole model accuracy is quite limited.

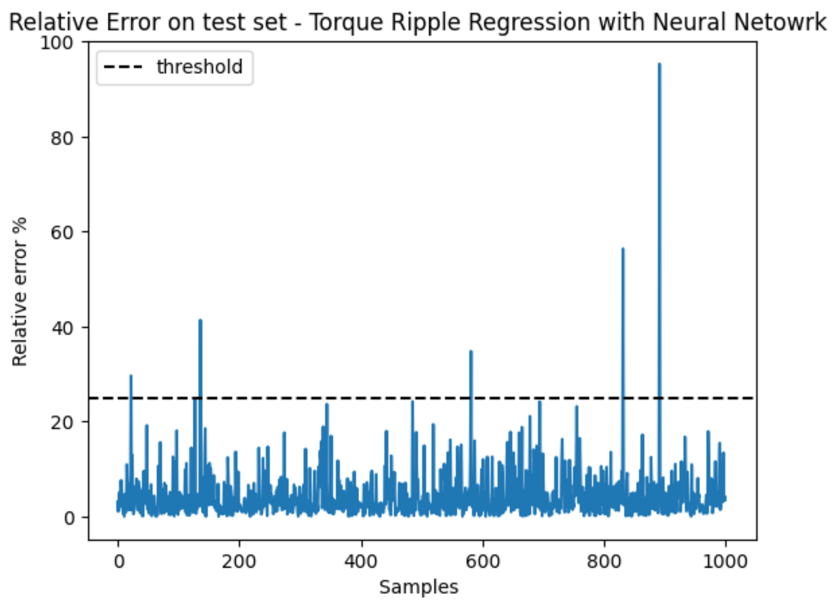


Figure 4.66: Samples with relative error on test set - Torque ripple Neural Network

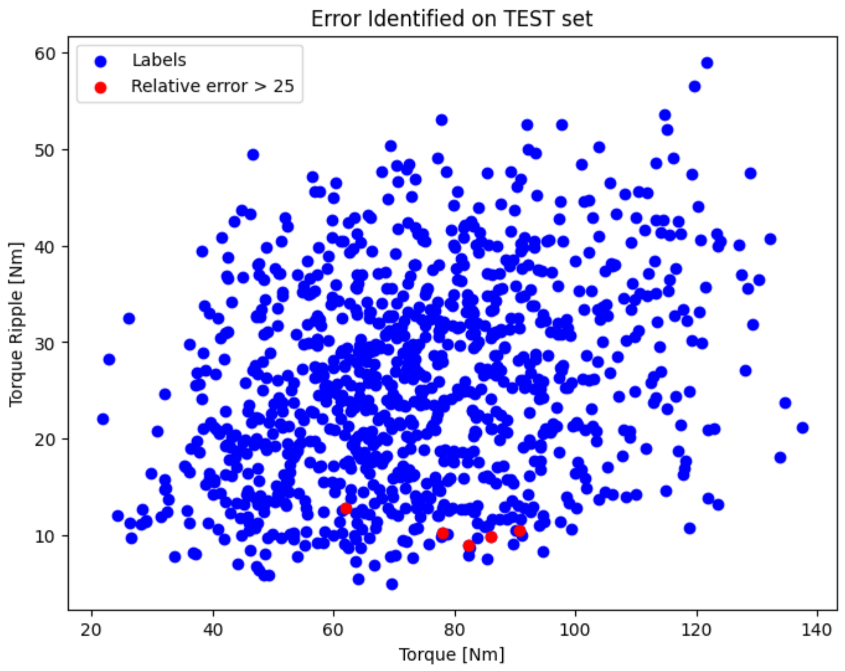


Figure 4.67: Samples with relative error on test set design space - Neural Network

Influence on Training size Also for the torque ripple model it is interesting to perform the sensitivity analysis to understand how the number of training samples is influencing the results. For this purpose Fig. 4.69 is reported. It is clear how even in the case of Neural Network, a similar trend to SVR model is achieved. In particular, focusing on the R^2 score, it is evident that there is a major advantages in training the model with a larger train size. In particular, the model is able to better fit the test set achieving a lower MSE. This confirms the fact that the complexity of the torque ripple prediction requires more samples with respect the torque case.

For what concerns the validation curve, Fig. 4.70, shows how a larger number of

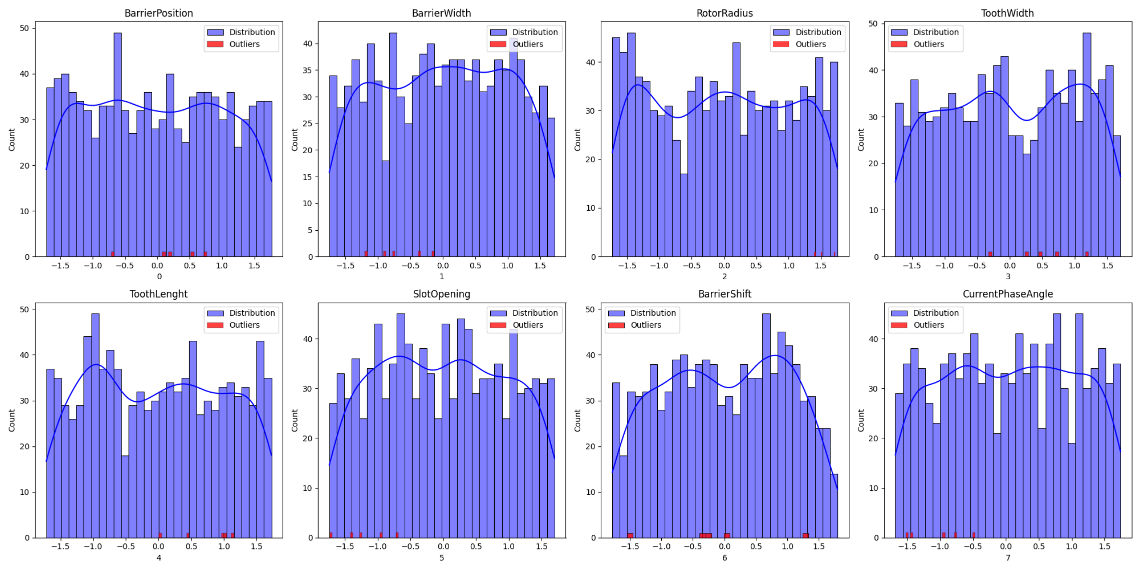


Figure 4.68: Distribution of outliers features on test set

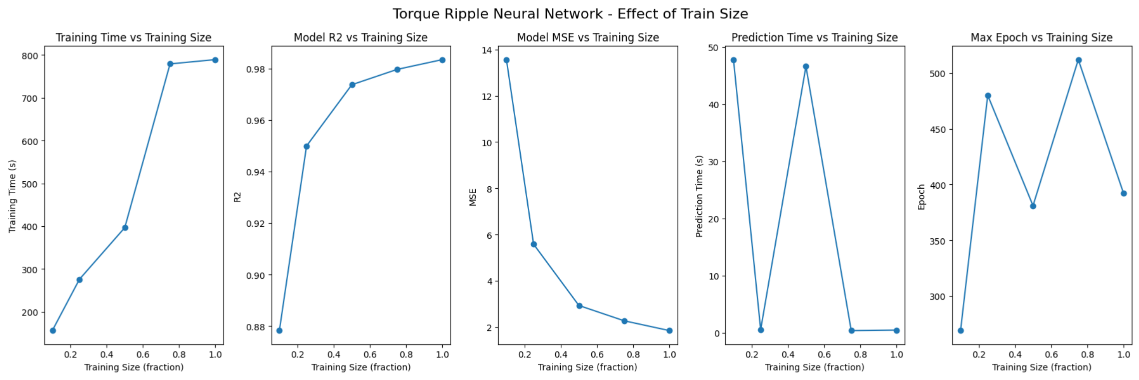


Figure 4.69: Effect of train size on torque ripple regression - Neural Network

training samples is able to improve the performance of the model, allowing to have a smaller gap between the validation loss and the training loss. For this reason, the full training subset is regarded as the best option.

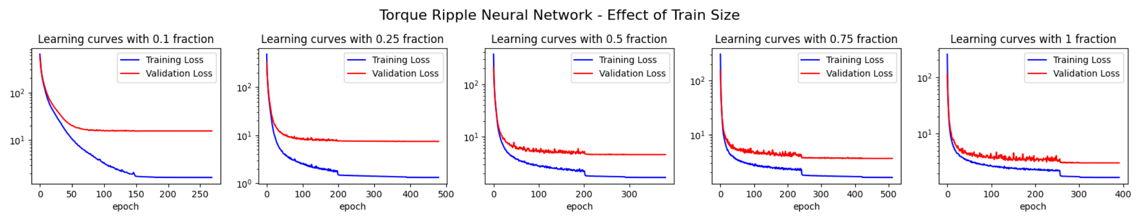


Figure 4.70: Effect of train size on torque ripple regression - Neural Network Validation Curve

Influence of Batch Size In this part, the effects of batch sizes on the training dynamics are presented. In particular 4 values are tested (8, 16, 32, 64). Looking at Fig. 4.71, as already occurring for the torque, also in this case, the best results are achieved considering 8 samples at the same time. Since the torque ripple is the the most critical output to predict for the purpose of this thesis, the compromise over a larger number of maximum epoch is accepted. Referring to Fig. 4.72, it is evident

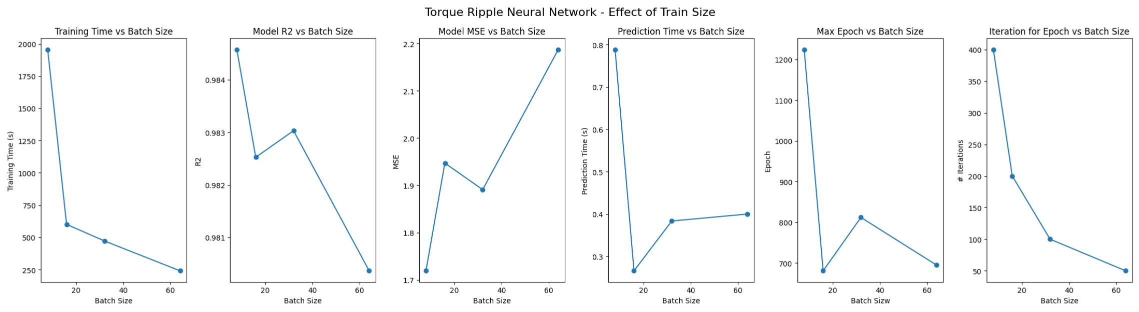


Figure 4.71: Effect of batch size on torque ripple regression - Neural Network

how the considerations done in case of torque batch size analysis are still valid. In particular, 8 as batch size allows for a faster convergence and for a longer training, ensuring therefore to achieve the best results.

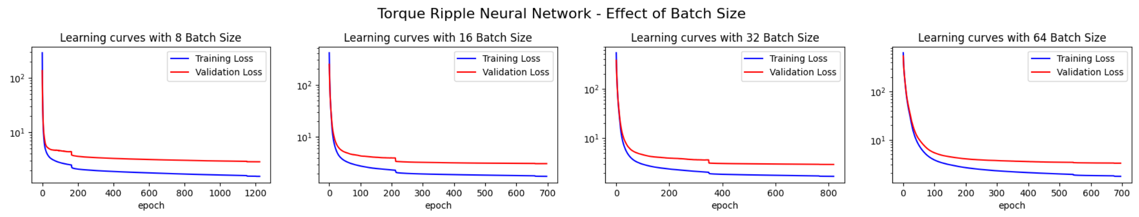


Figure 4.72: Effect of train size on torque ripple regression - Neural Network Validation Curve

4.4.5 Final Comparison among models

Model	Torque				Torque Ripple			
	R^2	MSE	MAE	Training Time (s)	R^2	MSE	MAE	Training time (s)
SVR	0.998	1.126	0.804	15.942	0.983	1.864	0.934	103.32
Neural Network	0.999	0.251	0.372	614	0.985	1.718	0.950	1771

Table 4.8: Performance Comparison of Machine Learning Models on test set

At this point all the models have been optimized and tested. A small recap about their prediction capabilities is given in Tab. 4.8. In particular, R^2, MSE, MAE on test set are used as metrics to compare the two models for the two different outputs of interest. Also training time is considered, as it gives direct insight about model complexity. It is possible to notice:

- **Torque**
  - R^2 is very similar for SVR and Neural Network, both really close to the unit value. Both models are able to explain the variance in the test data;

  - For what regards MSE, Neural Network is producing a much lower MSE (0.251) compared to the SVR (1.126). Neural Network is therefore generally able to avoid large prediction errors more effectively. This is confirmed also in Fig. 4.73;
  - Regarding MAE, the Neural Network has a lower MAE (0.372), which confirms that its predictions are generally closer to the actual torque values compared to SVR (0.804);
  - SVR is much faster at training time compared to Neural Network, suggesting that it is more computationally efficient for torque predictions.

- **Torque ripple**
  - R^2 is very similar for SVR and Neural Network, both slightly lower with respect the torque regression. This confirms that the torque ripple is more complex to predict starting from the geometrical configuration of the machine. The Neural Network (0.985) still outperforms SVR (0.983), but the difference is minimal;
  - Neural Network is achieving a lower MSE (1.718) versus SVR (1.864). This means that the first one on average is better at minimizing large errors at predicting torque ripple;
  - Considering MAE the Neural Network has a larger MAE (0.950) than SVR (0.934), and this values is influenced by the sample with very high relative error associated, as reported in Fig. 4.74;
  - SVR is much faster at training time compared to Neural Network.

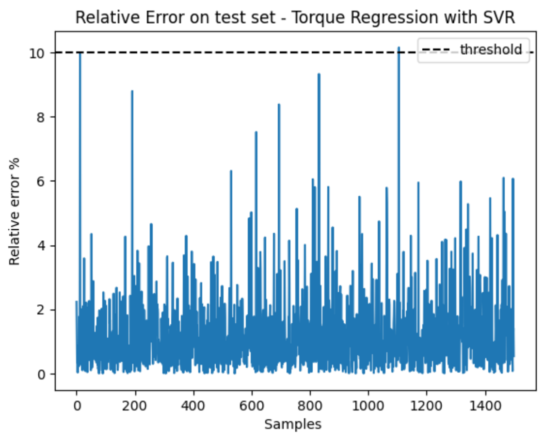
The main outcomes are:

- For what regards *torque*, Neural Network can minimize more effectively the risk of larger prediction errors (lower MSE). At the same time, it can offer an overall better average accuracy (lower MAE), generally making smaller errors. This is confirmed by Fig. 4.73, where the frequency and the magnitude of the spikes are lower. However, SVR is outperforming Neural Network in terms of training time.

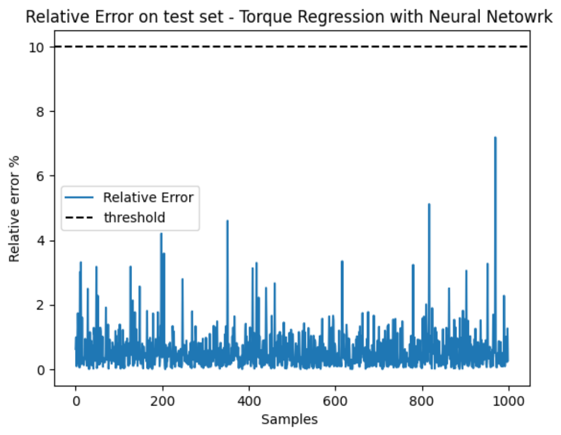
- For *torque ripple* prediction, SVR is offering more consistent predictions (due to the lower MAE) but Neural Network is able to slightly reduce the impact of larger errors (due to lower MSE). Looking at Fig. 4.74, it is clear how SVR has more frequent and less smoother relative error peaks compared to Neural Network. Similarly to the previous case, SVR is outperforming Neural Network in terms of training time.

Other valuable insight can be given looking at percentile values evaluated on the relative error between the true value and the label. In particular, the *i-th percentile* is corresponding to the values below which a given percentage of observations in the dataset falls. In particular, *i-th percentile* means that i% of the dataset is predicted with an error lower than to the corresponding value. Specifically, looking at Tab. 4.9, it is possible to say:

- Torque:

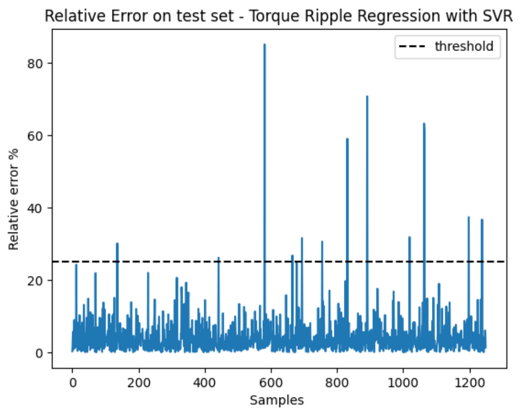


(a) Torque relative error on test set - SVR

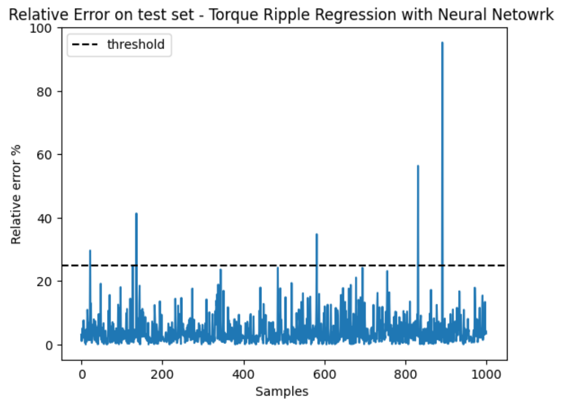


(b) Torque relative error on test set - NN

Figure 4.73: Final comparison Torque



(a) Torque ripple relative error on test set - SVR



(b) Torque ripple relative error on test set - NN

Figure 4.74: Final comparison - Torque ripple

Model	**Torque**					**Torque Ripple**				
	25 perc (%)	50 perc (%)	75 perc (%)	mean (%)	min (%)	25 perc (%)	50 perc (%)	75 perc (%)	mean (%)	min (%)
SVR	0.41	0.88	1.59	1.18	0.001	1.28	2.94	5.01	4.22	0.001
Neural Network	0.17	0.39	0.76	0.56	0.0001	1.36	2.90	5.54	4.37	0.001

Table 4.9: Error Performance Comparison of Machine Learning Models on test set

- 25th percentile: Neural Network is able to produce very low error;
- 50th percentile: considering the median, Neural Network is producing smaller errors;
- 75th percentile: In the upper end of error distribution, Neural Network is more able to control larger errors;
- mean: On average, Neural Network has almost half prediction errors compared to the SVR, indicating better overall performance;
- minimum error: both models are able to achieve near-perfect predictions with Neural Network achieving (0.0001% relative error).

• Torque ripple:

- 25th percentile: 25% of the errors are below 1.28% for the SVR and 1.36% for the Neural Network. This means that SVR is producing smaller errors more frequently;
- 50th percentile: considering the median, Neural Network is producing smaller errors;
- 75th percentile: In the upper end of error distribution, SVR is more able to control larger errors;
- mean: on average, SVR has slightly lower prediction errors compared to the Neural Network, indicating better overall performance;
- minimum error: both models are able to achieve near-perfect predictions (0.001% relative error).

These results are showing that both SVR and Neural Network can achieve similar results on the given dataset, providing a valuable methods to predict the main two outputs (torque and torque ripple), ensuring predictions within a certain error range. A key parameter for the decision of the Machine Learning model to adopt is also the training time. In particular SVR demonstrated its superiority in providing faster fit time on the train set. This is particular relevant if computational effort is a concern.

Chapter 5

Multi-objective optimization

This chapter is dedicated to the analysis of an optimization model implemented using DEAP library [48]. As already mentioned in the first part of this thesis, the exploration of the design space is highly limited by the available computational resources. To enhance the efficiency of exploring the objective space, surrogate models are used to predict the torque and the torque ripple inside the optimization framework. This is allowing to speed up the process of exploration of the objective space. To actually asses the feasibility of this process, a comparison with the results obtained with a real optimization process using FEA is reported. The final goal is to establish if the trained models are able to give valuable insight when dealing with an optimization process giving effective support to the classical optimization methodologies. It is important to underline that this process is not meant to substitute the traditional design framework, but it can be used to improve the exploration, allowing to use FEA just on a limited portion, identified with ML, of the design space.

In this thesis, two main regression machine models were developed (SVR and Neural Network). In Sec. 4.4.5, their performances are compared using different metrics. The main outcome is that the error among the models is comparable, while SVR has a huge advantage in terms of training time. Due to this reason, the following multi-objective problem is implemented using the SVR as evaluation model for each individual. Before reporting the results, a brief overview about Evolutionary algorithms (EAs) is given.

5.1 Evolutionary algorithms (EAs)

Evolutionary algorithms (EAs) can be regarded as a subset of *evolutionary computation (EC)*, that in general refers to a population - based meta - heuristic optimization algorithm. EAs are inspired to the biological evolution that focuses on mechanism as mutation, combination and selection. The purpose of this tools is to identify a set of possible solutions, starting from an initial random set of individuals. To determine the quality of a candidate, fitness functions are required. By minimizing (or maximizing) them, the algorithms are able to evolve the initial population, till the convergence to the final set of solutions [49]. Alg. 1 represents the mathematical formulation of the problem.

EAs can be applied to problems with either many design variables either many objectives. Both conditions are associated with a high complexity that is requiring

Algorithm 1 General Procedure of MOA [50]

```
Let $X$ be a set of possible solutions.
Find $x_0 \in X$ such that maximize/minimize $f$
(i.e. $f(x_0) = \max_{x \in X} f(x)$ or $f(x_0) = \min_{x \in X} f(x)$)
where $f : X \to \mathbb{R}$.
```

many computational effort to solve it. It is therefore suitable to solve multi-objective problems, where competing objectives and many design variables are present.
In the *evolutionary computation (EC)* domain, many algorithms are present, each of them with its own peculiarities [50]. The most common are: genetic algorithm (GA), genetic programming (GP), differential evolution (DE), the evolution strategy (ES), and evolutionary programming (EP). As a brief overview:

- **Genetic algorithm (GA)**: promotes evolution simulating natural selection using genetic operators like selection, mutation and cross-over;
- **Genetic Programming (GP)**: solutions are in forms of computer programs. Their fitness is determined by the ability of solving computational expensive programs;
- **Differential evolution (DE)**: mutation is used as main operator to perturb the population stimulating evolution of the individuals, focusing on continuous optimization;
- **Evolution strategy (ES)**: Focuses on optimizing real-valued parameters using mutation and selection, often applied to continuous optimization problems;
- **Evolutionary programming (EP)**: similar to GP but in this case the structure of the program is fixed while its numerical parameters are allowed to evolve.

Despite this different implementation of *EAs*, it is relevant to say that the *no free lunch theorem* is valid. Indeed, under the same condition, no evolutionary algorithm is fundamentally better than others [51].

In this thesis, Differential Evolution (DE) is chosen as main strategy since it can be easily implemented in python using DEAP library. Moreover, this evolutionary algorithm is already implemented in SyR- e and therefore it is possible to assess the quality of the surrogate model (SVR in this case) in the prediction of the best set of individuals by comparing it with the *best set so far*, evaluated using FEA.

5.1.1 Differential Evolution

Differential Evolution is a widely adopted *evolutionary algorithm (EAs)* for solving multi-objective problems. Unlike the other population - based methods, it generates new offspring by recombining previous solutions, following specific conditions. Moreover, the current solution is replaced if the new offspring outperforms it. The benefit of this approach is that it is possible to explore the design space and to obtain the best decision variables combination, ensuring to respect the constraints of the problem. Differential Evolution has moreover demonstrated very good performances dealing with non linear - problems, providing a simple implementation of the problem [52].

As other EAs, also Differential Evolution is producing new offsprings using three genetic operators like mutation, cross-over, and selection. Fig. 5.1 reports the logical sequence of this operations in the Differential Evolution framework.

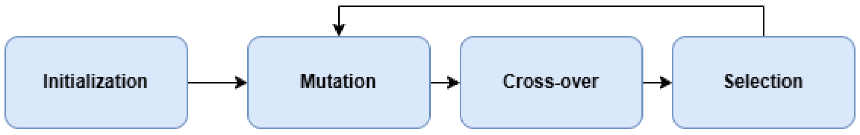


Figure 5.1: Differential Evolution Scheme

Before introducing the experimental setup that has been used, a small overview about each operator and its mechanism is reported.

Initialization After having decided the *population size NP*, initialization is used to populate the initial population by defining NP random configurations. For a N-dimensional problem, each individual is a vector of dimensions [1 x N]. It is worth mentioning that in order to ensure that each feature is correctly scaled, it is important to establish a lower bound and an upper bound for each dimension. By doing so, the problem is correctly constrained ensuring to have an effective design space exploration. Generally, random uniform is chosen as sampling method to populate the population.

Mutation The first genetic operator is mutation. It is used to modify the genes of the populations, according to the scale factor F that ranges between [0, 1]. Different mutation mechanism are present in literature, as reported in [52]. For the development of this thesis, *DE/current-to-best/2* is used. In particular, the expression is reported in Eq. 5.1.

$$\mathrm{Y}_i = \mathrm{X}_i + F \cdot (\mathrm{X}_{best} - \mathrm{X}_i) + F \cdot (\mathrm{X}_{r1} - \mathrm{X}_{r2}) \tag{5.1}$$

where:

- Y is the mutant vector;
- X is the current individual (target);
- best is the best individual of the current population;
- $\mathrm{X}_{r1}, \mathrm{X}_{r2}$ are two randomly selected individuals;
- F is the scale factor.

This current implementation is ensuring the the mutant is evolving toward the best solution. To avoid a too fast convergence, promoting therefore a good exploration, the mutant is also influenced by other two random individuals. Choosing the right F provides a good trade-off between exploration and exploitation.

Cross-over To further enhance diversity and variability, DE also implements cross-over. In particular, this genetic operator is used to combine the genes of the mutant and the target, following Eq. 5.2, which represents the *uniform cross-over* equation. The new vector is called offspring.

$$Z_{i,j}^{t} = \begin{cases} Y_{i,j}^{t} & \text{if } rand_{ij}[0,1] \leq CR \text{ or } j = k \\ X_{i,j}^{t} & \text{Otherwise} \end{cases} \tag{5.2}$$

Specifically:

- Z_i: is the offspring vector;
- CR: is the cross over rate that ranges between [0, 1]. It is used to control the cross over by comparing it with a random number;

Selection At each generation, the population size is fixed to NP. During the evolution, the creation of offspring is enlarging the initial population, overcoming the maximum number. In order to reduce the size, the last operation is selection. In particular, this step is used to cut each generation's population, in order to be consistent with NP. The mathematical expression adopted in Differential Evolution is reported in Eq. 5.3.

$$X_{i}^{t+1} = \begin{cases} Z_{i}^{t} & \text{if } f(Z_i^t) \leq f(X_i^t) \\ X_{i}^{t} & \text{Otherwise} \end{cases} \tag{5.3}$$

Where:

- X_i^{t+1}: is the individual belonging to the next generation;
- $f(Z_i^t), f(X_i^t)$: are the fitness's values of the offspring and of the target vector.

This means that the offspring is replacing the target vector if it has a better fitness value. This is ensuring the correct evolution toward the best set.

Fitness The fitness function is a fundamental component in any EAs and it used to evaluate the quality of a candidate during the evolution process. It is used as evaluation metrics to drive the process toward the optimal set. For the purpose of this thesis, torque has to be maximized while torque ripple has to be minimized. Fitness is strictly connected to an evaluation function. As already mentioned in Chap. 3, SyR- e and FEMM are used together to evaluate the torque and torque ripple of each geometrical configuration, obtaining the dataset (Fig. 3.10). This process is computationally intensive and required many hours to be completed. By using ML models trained on the dataset, it may be possible to improve the speed of convergence toward the optimal set, using the SVR to predict the fitness of the configurations. Due to the presence of error between the real outputs and the predicted one, it is important to assess the quality of the optimal set by comparing it with the one evaluated by FEMM.

Exploration Exploration is linked to the ability of the EAs to discover new and different solutions, avoiding to focus too much on certain design space areas. This is particular important since it allows to prevent premature convergence toward a possible local optimal solution. On the other hand, excessive exploration may lead to inefficiency, since it may require a lot of effort on searching in a poor design area.

Exploitation A complementary concept to exploration in optimization process is exploitation. The goal is to refine and improve solutions inside promising areas, narrowing therefore the design space. It is used thus to fine-tune the solutions, allowing to define the optimal solutions by making small adjustments. A too aggressive exploitation may lead to be trapped in local optima.

Exploration and Exploitation are thus two opposite concepts that are driving the evolution in a optimization process. It is particularly relevant to balance them, in order to find high-quality solutions inside the design space.

5.1.2 Experimental Setup

After the brief theoretical description of the main concept behind Differential Evolution, it is possible to introduce the implementation of the algorithm and the main parameters that have been used for this thesis. The logical sequence is reported in Fig. 5.2. The Algorithm is composed by 3 different blocks:

- **Initialization**: An initial population composed by 20 individuals is created, using random uniform distribution. Each individual is thus characterized by 8 features (the geometrical parameters) and 2 labels (torque and torque ripple). Since the evaluation is based on the SVR, it is mandatory to be consistent with the dataset used during training. Therefore each features has to respect the upper bound and the lower bound of the dataset (Fig. 3.12. Moreover, each geometrical variable has to be pre-processed in order to have 0 mean value and unitary standard deviation. This is particular important since the performances of the two SVR models are known inside this range (more details in Chap. 4).
- **Evolution loop**: After the definition of the maximum number of generations of the optimization process, mutation, cross-over, and selection are performed. Specifically:
  - **mutation**: is implemented following the *DE/current-to-best/2* logic of Eq. 5.1. The probability of mutation is controlled by the *mutation rate* τ_1. This value has been set equal to 0.1 and it is used to impose the *scale factor F*. The mathematical formulation is shown in Eq. 5.4.

$$scalefactor = \begin{cases} \text{random}(), & \text{if random}() < \tau_1 \\ F, & \text{otherwise} \end{cases} \tag{5.4}$$

    In particular, there is a probability of 10% of having a random scale factor while in the other 90% of the cases, F is fixed to 0.75. This value is promoting exploration of the design space. On the other hand, the introduction of a random F, is fostering exploitation. Overall, this approach is allowing to have enough flexibility ensuring to have a good balance among exploration and exploitation. It is also important to check always the bounds of the features of each mutant, in order to ensure consistent prediction from the SVR.

- **cross-over**: it is used to create a new individual (called *trial*) based on the combination of a *mutant* and *target*. This mechanism is controlled by *cross over rate* and reported in Eq. 5.5.

$$\text{trial} = \begin{cases} \text{mutant}[i], & \text{if random}() < \text{crossover\_rate} \\ \text{target}[i], & \text{otherwise} \end{cases}, \quad \forall i \in [0, n-1] \tag{5.5}$$

  Specifically, cross over rate is controlling the influence that the mutant have on the new trial. A higher value means that the trial has inherited more characteristics from the mutant. This process is repeated for each gene of the individual (n is the number of variables, 8 in this case). For the purpose of this thesis, cross over rate equal to 0.3 has been used. This value is encouraging exploitation, as the algorithm is more likely to refine solutions (*target*) rather than exploring new areas. This value has been chosen as trade-off between speed of convergence and accuracy of the final set.

- **selection**: as already mentioned, at each generation the population size should be fixed and equal to NP. During the evolution process, mutation and cross-over are increasing the number of individuals above the limit, requiring therefore a selection stage. To promote diversity and better accuracy, *crowding distance* is used to prevent convergence to a single area promoting solutions that are spread in the design space [48]. Eq. 5.6 reports the formulation.

$$\text{crowding\_distance}(i) = \sum_{m=1}^{M} \left(\frac{f_m^{\text{next}}(i) - f_m^{\text{prev}}(i)}{f_m^{\text{max}} - f_m^{\text{min}}} \right) \tag{5.6}$$

  Where:

  * i: is the index of the individuals;
  * m: is the index of the objectives;
  * $f_m^{\text{next}}(i)$: is the objective value of the next neighbor in the sorted order;
  * $f_m^{\text{prev}}(i)$: is the objective value of the previous neighbor in the sorted order;
  * f_m^{max} and f_m^{min}: are the maximum and minimum values of the m-th objective in the population.

  This can be used as an additional metric for the final sorting, allowing to obtain distributed solutions.

- **Final output**: Once all the generations are completed, the desired output is represented by the *Pareto Front* that represents the non-dominated solutions identified by the algorithm. Each point is a geometrical configuration, composed by the best combination of parameters found for the corresponding values of torque and torque ripple. This means that it is possible to retrieve the normalized geometrical features of each point. Then these values can be converted back to the de-normalized values by using the mean and the standard deviation of the training set, as done in Sec. 4.2.2. In the end, *Pareto set* is obtained.

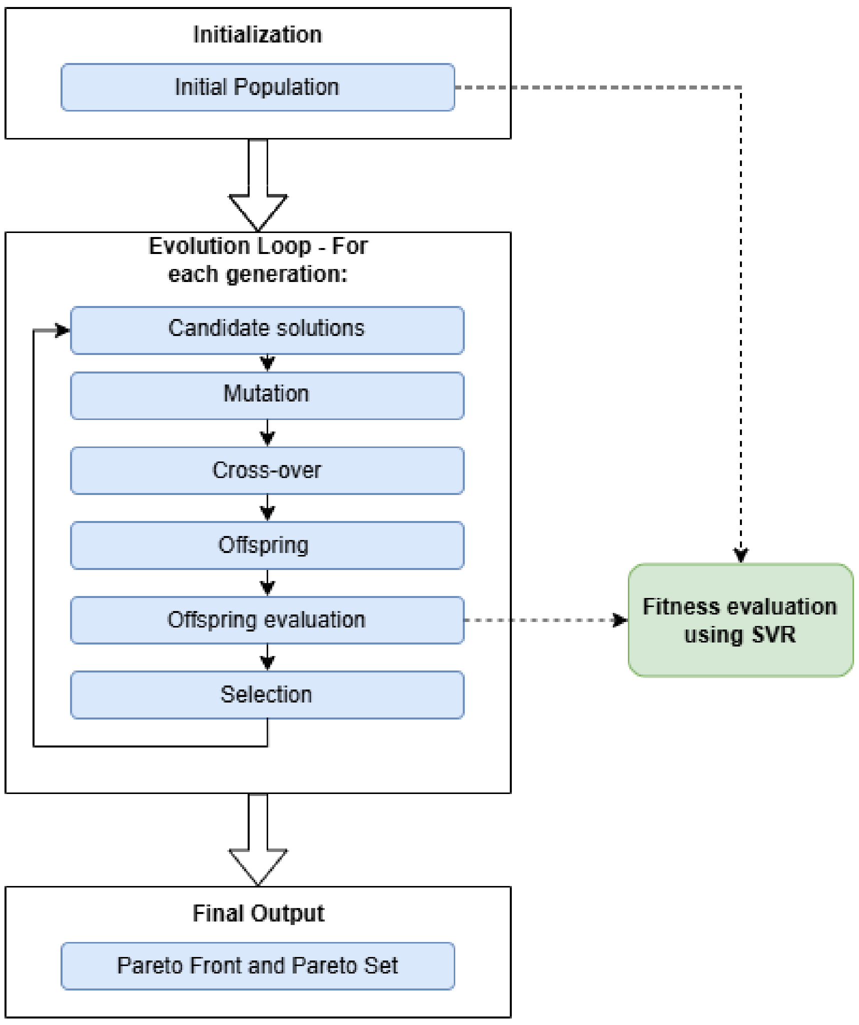


Figure 5.2: Multi-objective algorithm

Experimental Results

Finally, the results are now presented. First, the outcomes from the Differential Evolution (DE) algorithm, in combination with the Support Vector Regression (SVR), are discussed. Then, the real Pareto Front obtained on the full dataset is introduced. This allows to compare the two different best set, to understand what is the quality of the optimization process driven by the SVR. In the end, Pymoo performance index are used as a comparison metrics among the two different Pareto Fronts.

Differential Evolution with SVR After having defined the size of the population (NP = 20, in this case), it is necessary to decide the number of generations.

This parameter is quite important as it is linked to exploration and exploitation concepts (Sec. 5.1.1). Since the objectives are the maximization of the torque and the minimization of the torque ripple, those two are chosen as metrics to evaluate the convergence of the algorithm. In particular, for each generation, the average torque and the average torque ripple of the entire population are compared. Fig. 5.3 illustrates the behaviour of these two variables over the generations. At low generation numbers, it is clear that the predominant effect is due to the randomness of the chosen individuals belonging to the population. As the number of generations increases, Differential Evolution gradually provides solutions that align with the objectives. After several trial-and-error tests, the maximum number of generations have been set equal to 20, since it guarantees high average torque and low average torque ripple.

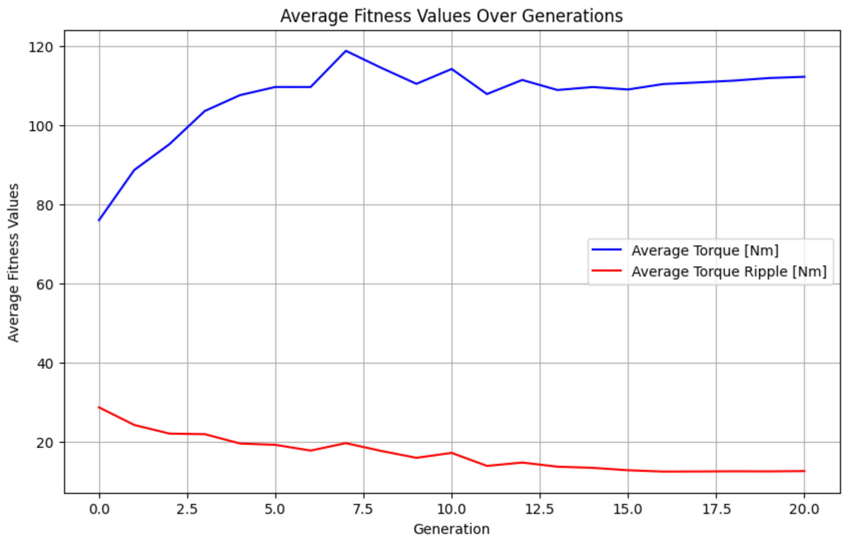


Figure 5.3: Average torque and torque ripple evolution

At the end of the evolution loop, the non-dominated solutions are identified. Specifically, being a bi-objective problems, it it possible to represent them in the Torque - Torque ripple plane, allowing for easier visualization. A non-dominated solution refers to a configuration where any improvement in one objective would lead to a deterioration in the other. This means that for any fixed value of one objective, the corresponding non-dominated solution provides the best achievable value for the other objective. Fig. 5.4, reports the evaluated solutions. Several key observations can be made:

- The set of solutions is spread across the objective plane. This means that the Differential Evolution setup has provided a good trade-off among exploration and exploitation;
- One point is associated with almost 0 Nm of torque ripple. Although this solution is not feasible, it is important to note that, with reference at Fig. 4.31, SVR exhibits its largest error in the lower torque ripple region;
- A total of 20 configurations belongs to the Pareto Front;
- The whole optimization process lasted 1.06 seconds.

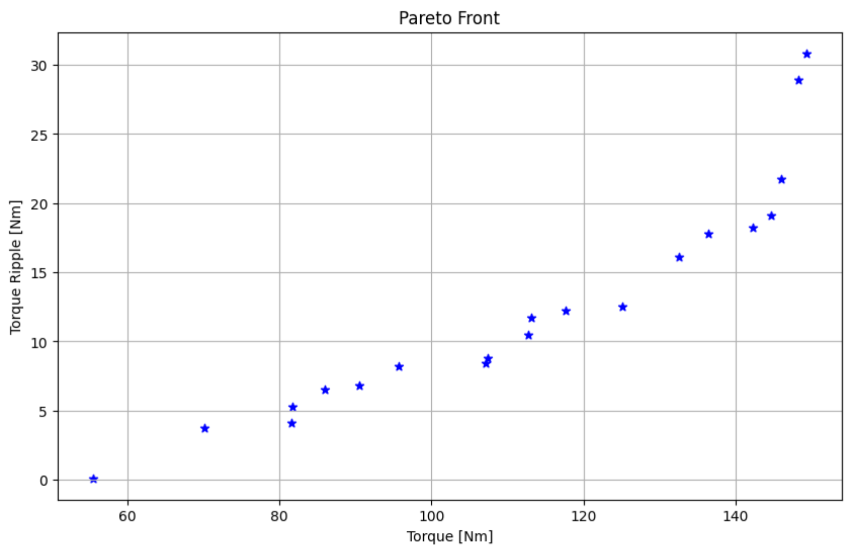
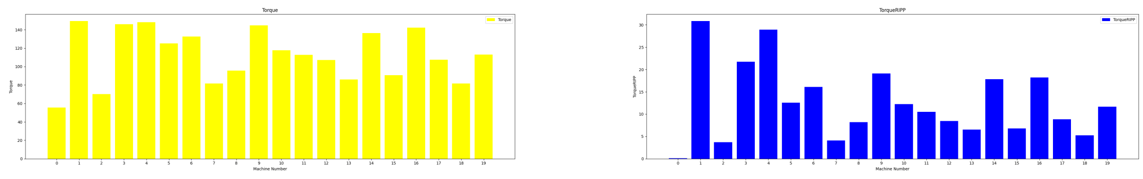


Figure 5.4: Pareto Front obtained with SVR

An alternative representation, similar to the one obtained at the end of the optimization process on SyR-e, is reported in Fig. 5.5. In this way, it is possible to associate each configuration with the corresponding machine characterized by a specific geometry.

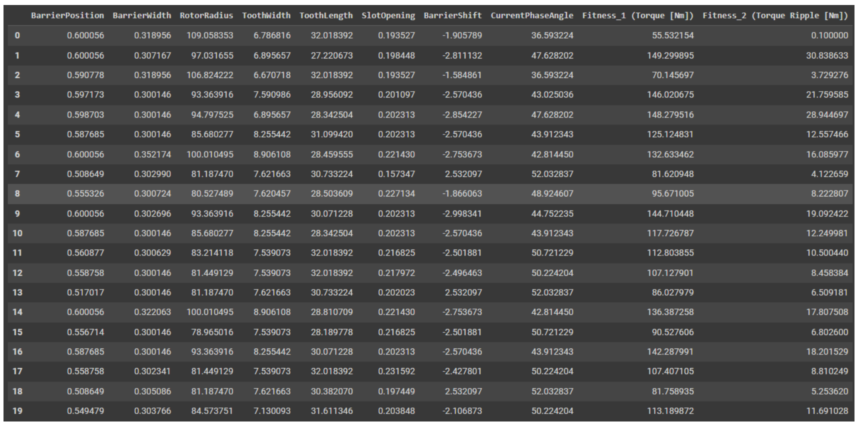


(a) Torque bar plot (b) Torque ripple bar plot

Figure 5.5: Bar plot - SVR Pareto Front

Additionally, it is possible to retrieve the geometrical characteristics of each machine. By doing so, each configuration is associated with a geometry and the corresponding output, evaluated with the SVR. The results are reported in Fig. 5.6. Those values are already representing the de-normalized variables, obtained using the mean and the standard deviation from the training set.

Pareto Front obtained on the dataset To assess the accuracy of the best set obtained by the SVR, it is necessary to compare it with the Pareto Front generated using FEA. Although, a full optimization process has not been performed using SyR- e, the high number of simulations performed to create the dataset allows to identify the best set composed by the non-dominated solutions. This task has been performed using Pymoo's non-dominated sorting [53]. The results are shown in Fig. 5.7. In particular, 17 configurations are identified: blue points represent dominated solutions, while green points denote the Pareto Front. This visualization also helps to understand the definition of non-dominated solution. Since those points are evaluated on the dataset, they are referred as *real Pareto*. Key observations include:



	BarrierPosition	BarrierWidth	RotorRadius	ToothWidth	ToothLength	SlotOpening	BarrierShift	CurrentPhaseAngle	Fitness_1 (Torque [Nm])	Fitness_2 (Torque Ripple [Nm])
0	0.600056	0.318956	109.058353	6.786816	32.018392	0.193527	-1.905789	36.593224	55.532154	0.100000
1	0.600056	0.307167	97.031655	6.895657	27.220673	0.198448	-2.811132	47.628202	149.299895	30.838633
2	0.590778	0.318956	106.824222	6.670718	32.018392	0.193527	-1.584861	36.593224	70.145697	3.729276
3	0.597173	0.300146	93.363916	7.590986	28.956092	0.201097	-2.570436	43.025036	146.020675	21.759585
4	0.598703	0.300146	94.797525	6.895657	28.342504	0.202313	-2.854227	47.628202	148.279516	28.944697
5	0.587685	0.300146	85.680277	8.255442	31.099420	0.202313	-2.570436	43.912343	125.124831	12.557466
6	0.600056	0.352174	100.010495	8.906108	28.459555	0.221430	-2.753673	42.814450	132.633462	16.085977
7	0.508649	0.302990	81.187470	7.621663	30.733224	0.157347	2.532097	52.032837	81.620948	4.122659
8	0.555326	0.300724	80.527489	7.620457	28.503609	0.227134	-1.866063	48.924607	95.671005	8.222807
9	0.600056	0.302696	93.363916	8.255442	30.071228	0.202313	-2.998341	44.752235	144.710448	19.092422
10	0.587685	0.300146	85.680277	8.255442	28.342504	0.202313	-2.570436	43.912343	117.726787	12.249981
11	0.560877	0.300629	83.214118	7.539073	32.018392	0.216825	-2.501881	50.721229	112.803855	10.500440
12	0.558758	0.300146	81.449129	7.539073	32.018392	0.217972	-2.496463	50.224204	107.127901	8.458384
13	0.517017	0.300146	81.187470	7.621663	30.733224	0.202023	2.532097	52.032837	86.027979	6.509181
14	0.600056	0.322063	100.010495	8.906108	28.810709	0.221430	-2.753673	42.814450	136.387258	17.807508
15	0.556714	0.300146	78.965016	7.539073	28.189778	0.216825	-2.501881	50.721229	90.527606	6.802600
16	0.587685	0.300146	93.363916	8.255442	30.071228	0.202313	-2.570436	43.912343	142.287991	18.201529
17	0.558758	0.302341	81.449129	7.539073	32.018392	0.231592	-2.427801	50.224204	107.407105	8.810249
18	0.508649	0.305086	81.187470	7.621663	30.382070	0.197449	2.532097	52.032837	81.758935	5.253620
19	0.549479	0.303766	84.573751	7.130093	31.611346	0.203848	-2.106873	50.224204	113.189872	11.691028

Figure 5.6: Parametric geometry and fitness of each configuration

- For a fixed torque value, the green points represents the solutions with the lowest torque ripple;
- Considering for example the blue points in the area around [20, 10] Nm, they do not belong to Pareto Front, as following the green curve, there are configurations with better torque and lower torque ripple.

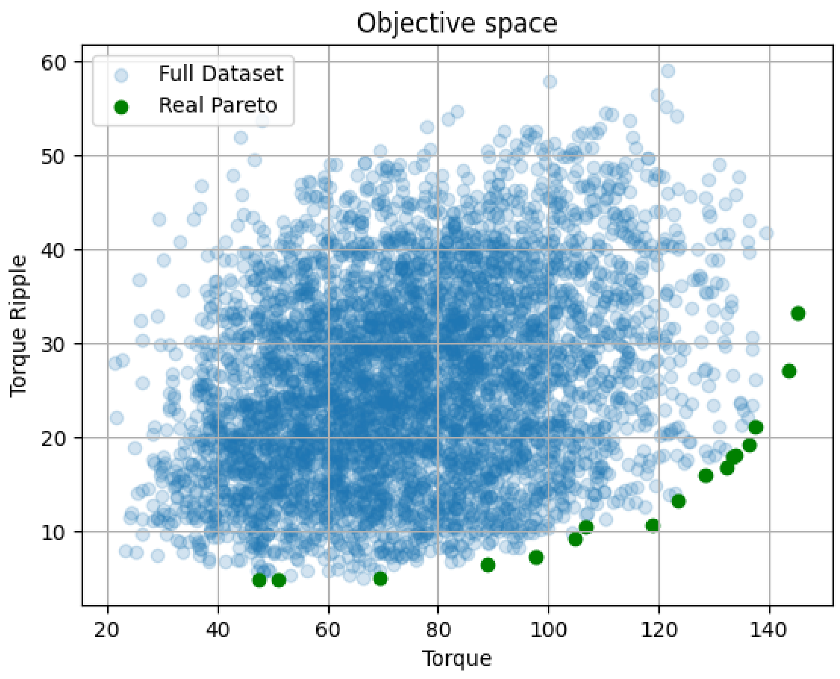


Figure 5.7: Pareto Front on the dataset

Comparison At this point it is possible to compare the results, by superimposing the two Pareto Fronts. Fig. 5.8 reports the outcomes. In particular, it is possible to notice:

- Both fronts are well-distributed in the objective space, providing a similar coverage. This indicates that the Differential Evolution setup with SVR achieves a good trade - off among exploration and exploitation.

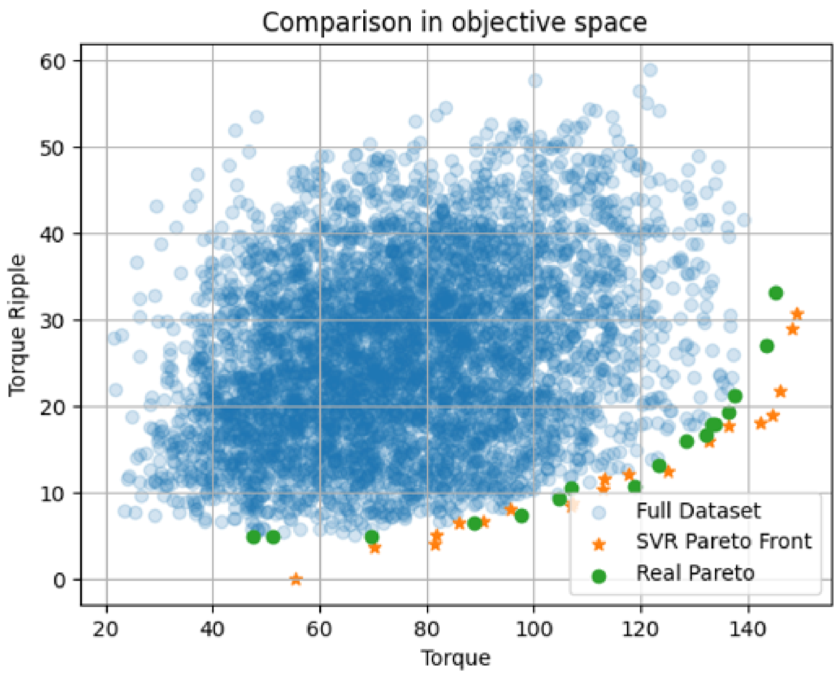


Figure 5.8: Pareto Fronts comparison in the objective space

- The torque range between [70, 130] Nm is well replicated by the SVR Pareto Front, with several points closely matching the real Pareto Front;
- In the lower torque ripple regions, the SVR-based Pareto Front deviates significantly from the real Pareto Front. This is due to the SVR model's difficulty in accurately predicting torque ripple, particularly where ripple values are small, as suggested by Fig. 4.33.

Moreover, a comparison among the two different Pareto sets is reported in Fig. 5.9. The idea is to understand if the SVR model is able to provide some insights about the geometries of the best solutions, allowing to better identify the boundaries of each feature. This allows to reduce the computational effort required by the FEA optimization, by focusing only on the most promising configurations.

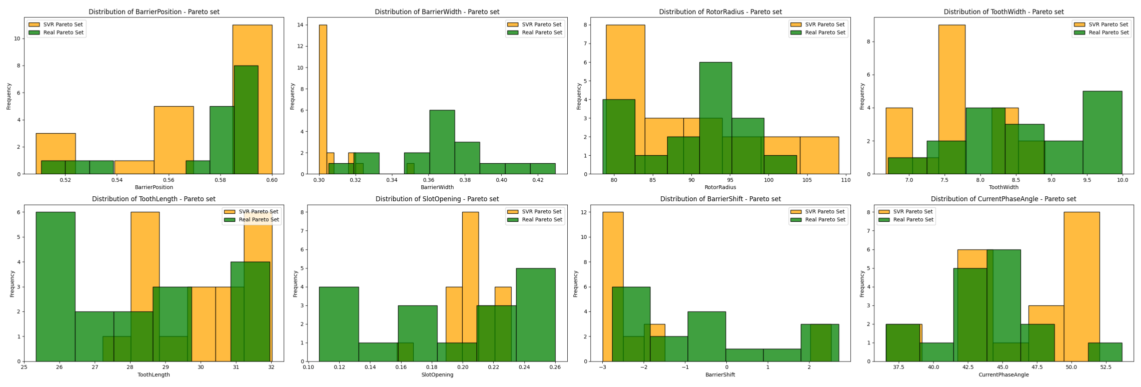


Figure 5.9: Pareto set comparison

Lastly, Pymoo Performance indicators [53] are reported, to assess the quality of the SVR Pareto Front compared to the real one giving quantitative insights. Three main metrics are used:

- **Generational Distance (GD)**: measures how far the obtained solutions are from the real Pareto Front. It is evaluated as the average Euclidean distance between each solution in the evaluated set (i.e., SVR) and the nearest solution in the real Pareto Front (i.e. the dataset front), as shown in Eq. 5.7. It gives, therefore, indications about the convergence of the two fronts.

$$GD = \left(\frac{1}{n} \sum_{i=1}^{n} d_i^p \right)^{\frac{1}{p}} \tag{5.7}$$

  Being a distance, it gives indication on how the evaluated solutions are close to the real ones. Lower values are more desirable.

- **Inverted Generational Distance (IGD)**: inverts the generational distance and measures the distance from any point in the real front to the closest point in the evaluated one. The mathematical expression is reported in Eq. 5.8.

$$IGD = \left(\frac{1}{n_{real}} \sum_{i=1}^{n_{real}} d_i^p \right)^{\frac{1}{p}} \tag{5.8}$$

  IGD helps in understanding how well the evaluated Pareto Front covers the true Pareto Front. Also in this case, lower values are indicating better coverage.

- **Hypervolume (HV)**: is the volume of the objective space dominated by the obtained Pareto Front solutions, bounded by a reference point (usually worse than the worst solution). It captures both the quality and diversity of solutions. Generally a larger HV means better diversity and coverage. It is not strictly correlated to the quality of the evaluated Pareto Front with respect the real one.

Performance indicator values are summarized in Tab. 5.1. The calculated perfor-

Metric	Value
Generational Distance (GD)	4.017
Inverted Generational Distance (IGD)	3.141
Hypervolume (HV) (SVR Pareto Front)	9550
Hypervolume (HV) (real Pareto Front)	8661

Table 5.1: Pymoo performance Indicators

mance indicators provides valuable insight about the effectiveness of the SVR Pareto Front with respect the real one. Specifically:

- Generational Distance (GD) of 4.017 suggests that there is a moderate distance between the two fronts. This indicates that while the SVR is effective in generating solutions, there is a certain discrepancy in terms of convergence to the real front;

- The Inverted Generational Distance (IGD) value of 3.141 indicates that the approximation of the real Pareto Front still can be improved, highlighting potential discrepancies in the lower torque ripple regions, where the SVR's predictions may be less reliable;
- The Hypervolume (HV) metric shows that the SVR Pareto Front achieves a hypervolume of 9550, which is higher than the real Pareto Front's hypervolume of 8661. This indicates that the SVR Pareto Front covers a larger area in the objective space, suggesting it provides a broader set of solutions and a good trade-off between torque and torque ripple.

Validation of the results with FEA It is essential to evaluate the error associated with the configurations identified by the SVR model. Each geometrical design is tested using SyR-e to assess its performance in terms of torque and torque ripple. This validation allows to measure how effectively the surrogate model framework can be employed in the optimization process.
At the end of the machine learning-driven optimization, the identified configurations are tested in SyR-e to evaluate their true fitness. Figure 5.10 compares the torque and torque ripple outputs predicted by the SVR model (orange points) with the true values calculated using the classical FEA-based approach (red points). Some configurations are accurately predicted by the surrogate model, while others, particularly in the lower torque region, show significant error. This behavior is expected since the SVR model tends to exhibit larger prediction errors in the low-torque range, while predictions improve as torque increases. Fig. 5.11 and Fig. 5.12 are report-

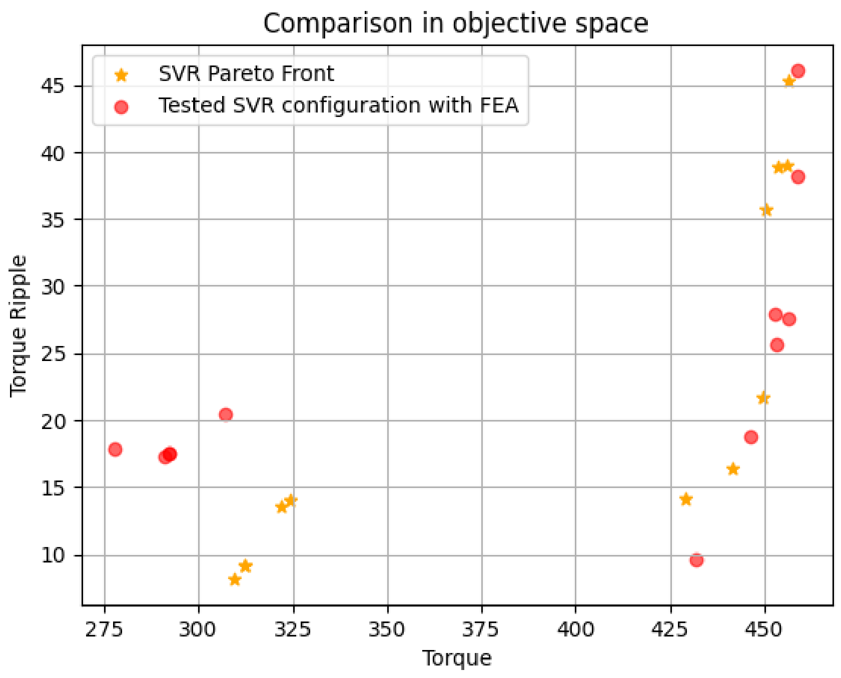


Figure 5.10: Configuration comparison - Pareto Front

ing the torque and torque ripple levels for each individual configuration (on the x-axis). It is evident how the low error associated to torque prediction, is allowing to closely match the FEA results. In contrast, torque ripple predictions exhibit a

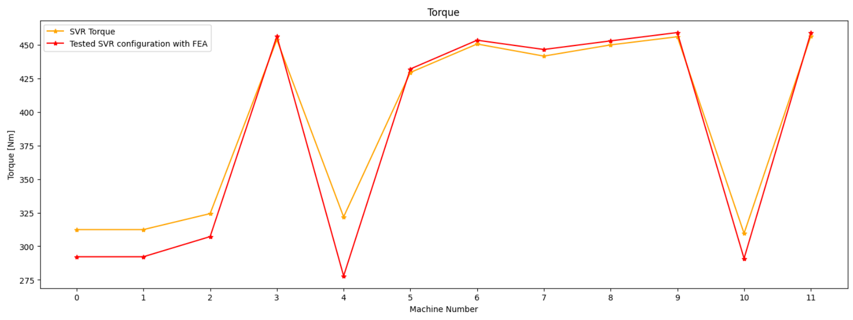


Figure 5.11: Configuration comparison - Torque

higher associated error, leading to solutions that are not always close to the true values. However, two configurations (9 and 11) are accurately predicted, suggesting promising results for further refinement of the surrogate model.

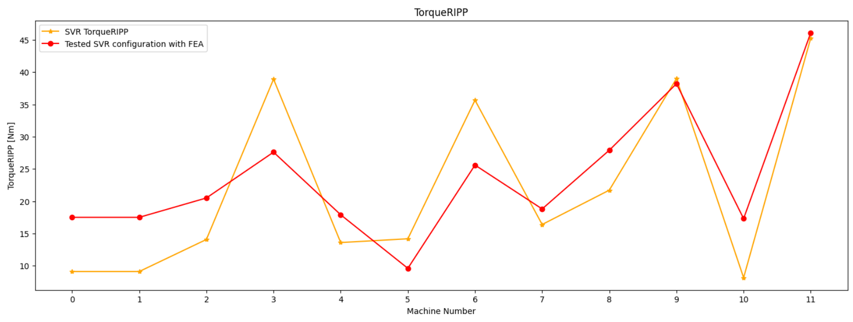


Figure 5.12: Configuration comparison - Torque Ripple

Overall it is clear how SVR model can yield to useful insights with respect the real Pareto Front. Pymoo's performance indicators are indicating how the ML model can be used to explore the design space effectively, especially where the prediction's accuracy is high. It is important to point out also that further refinements must be done to increase the accuracy in torque ripple predictions, that may lead to a better convergence of the two fronts. Anyway, it is important to emphasize the potential of using SVR in optimization processes for IPM motors, providing a valuable tool for a faster exploration of the design space.

Chapter 6

Conclusions and future works

The growing environmental concern has fostered the introduction in the automotive field of more sustainable powertrain architectures, specifically introducing electrical machines in place or together with Internal Combustion Engines. The search for a more sustainable mobility, requires an efficient electrical machine design to enhance performances and efficiency. In this scenario, Finite Element Analysis represents the state of art for the performances evaluation, especially in terms of torque and torque ripple. This tool is able to provide accurate solutions but it is very computationally expensive. This is limiting the exploration capabilities of the design space, slowing the design and optimization framework of the electrical machines. The introduction of Machine Learning in the design can be beneficial, providing insights about the solution in less time.

Machine Learning is a powerful tool that can be used to solve a wide range of problems, implementing models able to perform regression, classification and clustering. This thesis investigates the potential of Machine Learning in the design process of a Internal Permanent Magnet (IPM) motor. After an initial introduction about the motivations and the problems behind the actual design procedure, the theory behind Machine Learning training process is discussed with a particular focus on Support Vector Regression (SVR) and Neural Network.

The primary goal is to assess their effectiveness in predicting torque and torque ripple based on geometrical features of IPM configurations. The main outcome is that both SVR and Neural Network are valid models in predicting torque, achieving a R^2 close to the 1. However, predicting torque ripple is more challenging due to the highly non-linear relationship. In particular, more complex models are required, with an higher number of parameters. At the same time, the results are less accurate, providing a R^2 around 0.98 and predictions with larger relative error. Despite this, both models have demonstrated to be effective in predicting the two output of interest. Moreover, the big advantage is in terms of speed. Indeed, both SVR and Neural Network are able to provide the results in a much shorter time compared to classical FEA simulations, where multiple runs have to be performed in order to evaluate the performances. Comparing SVR and Neural Network, it is possible to assess that both models are producing similar MAE and MSE while the biggest difference is on the training time. In this case, SVR demonstrated to be much faster compared to Neural Network.

Since ML allows to produce the output, even if affected by errors, it is interesting to assess if those models can be effectively used in a multi-objective optimization process, implemented with Differential Evolution (DE). The goal is to obtain the best set, represented by the non-dominated solutions. In this case, due to similar performances but shorter training time, SVR is used in the evaluation phase, to obtain the fitness of each candidate. The preliminary experimental setup has demonstrated how ML can be effectively used to explore the design space in a much faster time compared to FEA. A key outcome of the analysis, further supported by Pymoo's performance indicators, is that SVR can provide valuable insights into the set of optimal configurations. However, it is important to note that SVR does introduce a certain degree of error, particularly in torque ripple predictions.

The task of predicting torque and torque ripple is particularly complex due to the intrinsic non-linearity in electrical machines. Despite the promising results, the models exhibit some reliability issues due to prediction errors, providing solutions that are not always reliable. Further work needs to be done focusing in identifying the best sampling method for creating a robust training dataset and the optimal size of the dataset, in order to maximize prediction accuracy. At the same time, the multi-optimization framework should be tested on other dataset, in order to assess the feasibility of the procedure.

Even if this research is just a preliminary setup, the gained insights may be of assistance to define a framework, relying on machine learning that can enhance the design and the optimization of the geometry of electrical machines, in order to promote e-Mobility and sustainable development in the automotive industry.

Bibliography

[1] Gro Harlem Brundtland, Mansour Khalid, et al. *Our common future.* Oxford University Press, Oxford, GB, 1987.

[2] Stephanie Shaw and Bill Van Heyst. «Nitrogen Oxide (NOx) emissions as an indicator for sustainability». In: *Environmental and Sustainability Indicators* 15 (2022), p. 100188. ISSN: 2665-9727. DOI: https://doi.org/10.1016/j.indic.2022.100188. URL: https://www.sciencedirect.com/science/article/pii/S2665972722000204.

[3] Vittorio Ravello. «E-powertrain components». Lecture Notes.

[4] IEA. «Global EV Outlook 2024». IEA, Paris. Licence: CC BY 4.0. 2024. URL: https://www.iea.org/reports/global-ev-outlook-2024.

[5] Zhi Cao et al. «An Overview of Electric Motors for Electric Vehicles». In: *2021 31st Australasian Universities Power Engineering Conference (AUPEC).* 2021, pp. 1–6. DOI: 10.1109/AUPEC52110.2021.9597739.

[6] Radu Bojoi. «Electrical Drives for eMobility». Lecture Notes.

[7] Seunghyeon Cho et al. «Optimal design to reduce torque ripple of IPM motor with radial based function meta-model considering design sensitivity analysis». In: *Journal of Mechanical Science and Technology* 33.8 (Aug. 2019), pp. 3955–3961. ISSN: 1976-3824. DOI: 10.1007/s12206-019-0740-0. URL: https://doi.org/10.1007/s12206-019-0740-0.

[8] Lavanya Balasubramanian et al. «Design and Optimization of Interior Permanent Magnet (IPM) Motor for Electric Vehicle Applications». In: *CES Transactions on Electrical Machines and Systems* 7.2 (2023), pp. 202–209. DOI: 10.30941/CESTEMS.2023.00021.

[9] Honda Motor Co., Ltd. *Two-Motor Hybrid System (Honda eHEV).* Accessed: 2024-09-18. 2024. URL: https://global.honda/en/tech/two_motor_hybrid_system_honda_eHEV/.

[10] None None. «FY2015 Electric Drive Technologies Annual Progress Report». In: (Feb. 2016). DOI: 10.2172/1245342. URL: https://www.osti.gov/biblio/1245342.

[11] *SyR-e: Synchronous Reluctance - evolution.* https://github.com/SyR-e/syre_public.

[12] *Finite Element Method Magnetics (FEMM).* Accessed: 2024-09-23. 2024. URL: https://www.femm.info/wiki/HomePage.

[13] Wikipedia contributors. *Latin hypercube sampling — Wikipedia, The Free Encyclopedia.* [Online; accessed 14-August-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Latin_hypercube_sampling&oldid=1216779135.

[14] Tatiana Tommasi. «Data analysis and Artificial Intelligence». Lecture Notes.

[15] Kiatdd. *Pearson correlation coefficient.* Own work, CC BY-SA 3.0. 2013. URL: https://commons.wikimedia.org/w/index.php?curid=37108966.

[16] MathWorks. *Machine Learning - MATLAB & Simulink Solution.* Accessed: 2024-8-10. 2024. URL: https://it.mathworks.com/solutions/machine-learning.html.

[17] Sam Ansari and Ali Bou Nassif. «A Comprehensive Study of Regression Analysis and the Existing Techniques». In: *2022 Advances in Science and Engineering Technology International Conferences (ASET).* 2022, pp. 1–10. DOI: 10.1109/ASET53988.2022.9734973.

[18] Jason Brownlee. *Regression Metrics for Machine Learning.* Accessed: 2024-8-10. 2020. URL: https://machinelearningmastery.com/regression-metrics-for-machine-learning/.

[19] Wikipedia contributors. *Coefficient of determination — Wikipedia, The Free Encyclopedia.* [Online; accessed 22-August-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Coefficient_of_determination&oldid=1241125450.

[20] Scikit - learn. «train_test_split». https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[21] Xue Ying. «An Overview of Overfitting and its Solutions». In: *Journal of Physics: Conference Series* 1168 (Feb. 2019), p. 022022. DOI: 10.1088/1742-6596/1168/2/022022.

[22] Educative Team. *Overfitting and Underfitting.* Accessed: 2024-09-30. 2023. URL: https://www.educative.io/answers/overfitting-and-underfitting.

[23] Safaa Azzakhnini. «Multimodal fusion based approaches for multi-sensor systems: from enhancing performance to ethical concerns». PhD thesis. Dec. 2020.

[24] Petro Liashchynskyi and Pavlo Liashchynskyi. *Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS.* 2019. arXiv: 1912.06059 [cs.LG]. URL: https://arxiv.org/abs/1912.06059.

[25] James Bergstra and Yoshua Bengio. «Random search for hyper-parameter optimization». In: *J. Mach. Learn. Res.* 13.null (Feb. 2012), pp. 281–305. ISSN: 1532-4435.

[26] Scikit - learn. «GridSearchCV». https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

[27] Scikit - learn. «RandomizedSearchCV». https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html.

[28] Scikit - learn. «Cross-validation: evaluating estimator performance». https://scikit-learn.org/stable/modules/cross_validation.html.

[29] Georgios Kiminos and Christos Gkaris. «Using Machine Learning for Text Classification to identify useful information in texts: A comparison of Naïve Bayes and Support Vector Machines to identify decisions in business meeting transcripts». PhD thesis. Nov. 2020. DOI: 10.13140/RG.2.2.19552.92166.

[30] Cha Zhang and Yunqian Ma, eds. *Ensemble Machine Learning. Methods and Applications*. Springer New York, NY, 2012. ISBN: 978-1-4419-9326-7. DOI: 10.1007/978-1-4419-9326-7. URL: https://doi.org/10.1007/978-1-4419-9326-7.

[31] Jason Brownlee. *A Tour of Ensemble Learning Algorithms*. Accessed: 2024-09-30. 2021. URL: https://machinelearningmastery.com/tour-of-ensemble-learning-algorithms/.

[32] Scikit - learn. «AdaBoostRegressor». https://scikitt-earn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html.

[33] Jason Brownlee. *Learning Curves for Diagnosing Machine Learning Model Performance*. Accessed: 2024-08-29. 2020. URL: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/.

[34] Keras. *Keras EarlyStopping Callback*. Accessed: 2024-08-29. 2023. URL: https://keras.io/api/callbacks/early_stopping/.

[35] Niousha Rasifaghihi. «From Theory to Practice: Implementing Support Vector Regression for Predictions in Python». https://medium.com/@niousha.rf/support-vector-regressor-theory-and-coding-exercise-in-python-ca6a7dfda927.

[36] Mariette Awad and Rahul Khanna. «Support Vector Regression». In: *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*. Berkeley, CA: Apress, 2015, pp. 67–80. ISBN: 978-1-4302-5990-9. DOI: 10.1007/978-1-4302-5990-9_4. URL: https://doi.org/10.1007/978-1-4302-5990-9_4.

[37] Scikit-learn Developers. *Support Vector Machines — Regression Example*. Accessed: 2024-10-09. 2023. URL: https://scikit-learn.org/stable/auto_examples/svm/plot_svm_regression.html.

[38] Hao Ma, Shuxin Shao, and Xin Huang. «In-depth analysis of SVM kernel learning and its components». In: *Neural Computing and Applications* 34 (2022), pp. 123–138.

[39] Xiaojian Ding et al. «Random radial basis function kernel-based support vector machine». In: *Journal of the Franklin Institute* 358.18 (2021), pp. 10121–10140. ISSN: 0016-0032. DOI: https://doi.org/10.1016/j.jfranklin.2021.10.005. URL: https://www.sciencedirect.com/science/article/pii/S0016003221006025.

[40] Mostafa Sabzekar and Seyed Mohammad Hossein Hasheminejad. «Robust regression using support vector regressions». In: *Chaos, Solitons Fractals* 144 (2021), p. 110738. ISSN: 0960-0779. DOI: https://doi.org/10.1016/j.chaos.2021.110738. URL: https://www.sciencedirect.com/science/article/pii/S0960077921000916.

[41] Matthew Stewart. «Introduction to Neural Networks». https://towardsdatascience.com/simple-introduction-to-neural-networks-ac1d7c3d7a2c.

[42] Ksenia Sorokina. «Image Classification with Convolutional Neural Networks». https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8.

[43] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. *Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark*. 2022. arXiv: 2109.14545 [cs.LG]. URL: https://arxiv.org/abs/2109.14545.

[44] David Shulman. *Optimization Methods in Deep Learning: A Comprehensive Overview*. 2023. arXiv: 2302.09566 [cs.LG]. URL: https://arxiv.org/abs/2302.09566.

[45] Wikipedia contributors. *Backpropagation — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Backpropagation&oldid=1238994293. [Online; accessed 28-August-2024]. 2024.

[46] keras. «Sequential Model». https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8.

[47] keras. «Hyperband tuner». https://keras.io/api/keras_tuner/tuners/hyperband/.

[48] DEAP. «DEAP library reference». https://deap.readthedocs.io/en/master/api/tools.html.

[49] Andrew N. Sloss and Steven Gustafson. *2019 Evolutionary Algorithms Review*. 2019. arXiv: 1906.08870 [cs.NE]. URL: https://arxiv.org/abs/1906.08870.

[50] Adam Slowik and Halina Kwasnicka. «Evolutionary algorithms and their applications to engineering problems». In: *Neural Computing and Applications* 32.15 (2020), pp. 12363–12379. DOI: 10.1007/s00521-020-04832-8. URL: https://doi.org/10.1007/s00521-020-04832-8.

[51] Wikipedia contributors. *Evolutionary algorithm — Wikipedia, The Free Encyclopedia*. [Online; accessed 23-September-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Evolutionary_algorithm&oldid=1247004801.

[52] Mohamad Faiz Ahmad et al. «Differential evolution: A recent review based on state-of-the-art works». In: *Alexandria Engineering Journal* 61.5 (2022), pp. 3831–3872. ISSN: 1110-0168. DOI: https://doi.org/10.1016/j.aej.2021.09.013. URL: https://www.sciencedirect.com/science/article/pii/S111001682100613X.

[53] J. Blank and K. Deb. «pymoo: Multi-Objective Optimization in Python». In: *IEEE Access* 8 (2020), pp. 89497–89509.