POLITECNICO DI TORINO

Master's Degree in Aerospace Engineering

Master's Degree Thesis

Data-Driven Analysis of Rotors' Performance Using Machine Learning

Supervisors

Prof. Domenic D'AMBROSIO

Prof. Manuel CARREÑO RUIZ

Candidate

Luca BUCCIONI

July 2025

Abstract

This thesis focuses on the use of machine learning for the design, optimization, and performance analysis of rotors.

The ML models were trained using a public dataset provided by the University of Illinois, containing both data related to rotor geometry and performance data under static conditions, in which the advance ratio J is zero, and dynamic conditions, in which $J > 0$. The training was carried out using MATLAB's Regression Learner, selecting the most suitable models based on prediction errors.

During the course of the work, two codes were developed, designed to be flexible and scalable. The first allows for the design of rotors starting from performance or geometric specifications provided as input; the user can choose whether to generate a Pareto front using a multi-objective genetic algorithm, or to directly obtain a single configuration through a standard genetic algorithm. The second code is dedicated to the optimization of existing rotors: starting from an initial configuration, the model modifies the geometry within user-defined tolerances in order to improve a single selected performance metric, namely the thrust coefficient (CT) or the power coefficient (CP).

In addition to design and optimization, the developed models are used for the prediction of the performance of new rotors, allowing for a significant time saving compared to traditional numerical simulations. The results obtained are compared with those derived from BEM (Blade Element Momentum) and CFD (Computational Fluid Dynamics) analyses in order to verify their consistency.

Overall, the work constitutes a basis for future extensions, such as the integration of neural networks and the use of larger datasets to improve the generalization capability of the models.

Alla mia famiglia
A chi mi ha visto crescere
e continua a vivere nei miei ricordi

Acknowledgements

Giunto alla fine di questo percorso mi sento in dovere di ringraziare tutte le persone con cui ho condiviso il mio tempo in questi anni e senza le quali non sarei la persona che sono oggi.

In primo luogo ringrazio Manuel per l'aiuto continuo che mi ha fornito nello sviluppo di questa tesi.

Ringrazio i miei genitori perché non mi hanno mai fatto mancare niente, mettendomi nelle condizioni migliori per crescere e migliorare.

Ringrazio Letizia per il costante supporto che mi ha dato in questi anni. Quando ero triste o pensavo di non farcela c'eri sempre per starmi vicino e dirmi che tutto sarebbe andato bene. Grazie.

Ringrazio tutti i ragazzi del campus. Con voi ho passato molto del mio tempo libero, anche se non abbastanza. Le sere passate a giocare rimarranno sempre dentro di me. In particolare ci terrei a ringraziare Alessandro per tutto quello che ha fatto per me in questi anni e per la mossa segreta che ci ha permesso di superare tanti esami. Una parola speciale anche ad Armando per essere stato un punto di rifermento nel mio percorso.

Ringrazio il mio gruppo di amici dell'università. Grazie a voi ho imparato tanto e parte dei miei buoni risultati è merito vostro. In particolare ringrazio Alessio per tutti gli aiuti che ci siamo dati a vicenda e che per me sono stati fondamentali e Giuseppe per essere stato il miglior compagno con cui seguire ogni lezione in questi anni.

Infine, ringrazio i miei amici di Rieti per avermi sostenuto in questo percorso nonostante la lontananza. Grazie per l'aiuto e il sostegno che mi avete dato anche nei momenti più difficili.

Table of Contents

List of Tables

List of Figures

Acronyms

BEM
: Blade Element Momentum

CFD
: Computational Fluid Dynamics

CP
: Power coefficient

CT
: Thrust coefficient

GA
: Genetic Algorithm

GPR
: Gaussian Process Regression

ML
: Machine Learning

RMSE
: Root Mean Squared Error

RPM
: Revolutions Per Minute

SVM
: Support Vector Machines

Chapter 1

Introduction

In recent years, Machine Learning (ML) has been emerging as a valuable complement to traditional aerodynamic analysis techniques. In this work, which focuses on the analysis of small two-bladed rotors, regression models trained on experimental data can estimate rotor performance in a fraction of time, drastically reducing the computational costs of CFD simulations and blade element methods (BEM). This speed makes it possible to examine many geometric configurations from the earliest design stages, accelerating the design–test–optimization cycle.
This thesis proposes a data-driven framework for the analysis, generation and optimization of small and medium scale aeronautical rotors. The main goal is to determine whether low-complexity ML models can deliver sufficiently accurate predictions while requiring far fewer resources than conventional approaches.
The work sets out to:

- generate new rotor geometries with respect to one or more objectives;
- optimise existing rotors by introducing limited variations in chord and angle of incidence;
- predict the thrust (CT) and power (CP) coefficients starting solely from the blade geometry, in both static and dynamic regimes;
- validate the predictions by comparing them with CFD and BEM results and, where available, experimental data.

In this way, analysis times are expected to shrink from hours or days (CFD) to just a few tenths of a second, while keeping the average errors on CT and CP within acceptable limits.

In summary, the thesis explores the possibility that low complexity ML models can accompany traditional methodologies, drastically reducing computation times without significantly compromising accuracy. The results obtained will constitute a basis for future developments in rotor design.

The following chapters present the fundamental steps carried out for training the ML models, performed using MATLAB's Regression Learner, followed by the analysis of the MATLAB codes developed to obtain the results listed above.

Chapter 2

Machine Learning

2.1 Introduction

This chapter provides an introductory overview of ML. It includes key concepts underlying ML, its main categories, the most common problems encountered during model training and strategies to mitigate them. The typical ML project pipeline, from data acquisition to model validation, is also described. These theoretical concepts supply the context needed to understand the techniques and choices adopted in the following chapters.

2.2 What is Machine Learning

ML is a subset of artificial intelligence that aims to build, from input data, algorithms capable of learning patterns and relationships, automatically improving their behaviour through experience. As Arthur Samuel stated in 1959:

"[Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed." [1]

Once ML models have been trained, they can be used to make predictions, perform analyses, classify information, and more.

2.3 Types of Machine Learning

The concept of ML encompasses numerous applications across different fields and this has led over the years to the development of various kinds of ML. In particular, the following ones are recognised:

- supervised ML;

- unsupervised ML;
- reinforcement ML.

2.3.1 Supervised Machine Learning

Supervised learning is the most widespread form of ML and relies on labelled data to train models capable of classifying or predicting outcomes. The algorithm learns from a set of examples in which every input datum is associated with the correct output. It is essential that the training data realistically represent the situations the model will encounter in the real world, including complex or borderline cases. During training, the model adjusts its parameters by minimising a loss function that measures the error between its predictions and the expected results. This process, often supported by validation techniques, progressively improves model accuracy. Supervised learning leverages classification and regression techniques to develop ML models. [2, 3, 4, 5]

2.3.2 Unsupervised Machine Learning

Unsupervised ML analyses unlabelled data, allowing the model to autonomously discover hidden structures, patterns, or relationships. Unlike supervised learning, no predefined outputs are provided: the algorithm explores the data without external guidance, searching for meaningful clusters or anomalies. One of its principal applications is clustering, which organises data into groups based on shared characteristics. This approach is particularly useful when working with large volumes of raw data that are difficult to analyse manually. Moreover, data preparation requires less effort, because labelling or providing expected results is unnecessary. [2, 3, 5]

2.3.3 Reinforcement Machine Learning

Reinforcement learning is a branch of ML in which an autonomous agent learns to perform optimal actions within an environment by interacting with it. Unlike supervised learning, which relies on labelled data, and unsupervised learning, which seeks hidden patterns, reinforcement learning is based on a reward mechanism: the agent receives positive or negative feedback in response to its actions and, through trial and error, refines its behaviour. The goal is to maximise cumulative reward over time, even at the cost of making sub-optimal choices in the short term. Reinforcement learning is particularly suited to tackling complex, sequential decision-making problems in dynamic environments. Data are not independent but form sequences of states, actions and rewards, making the process more akin to the learning observed in biological agents. [2, 3, 6]

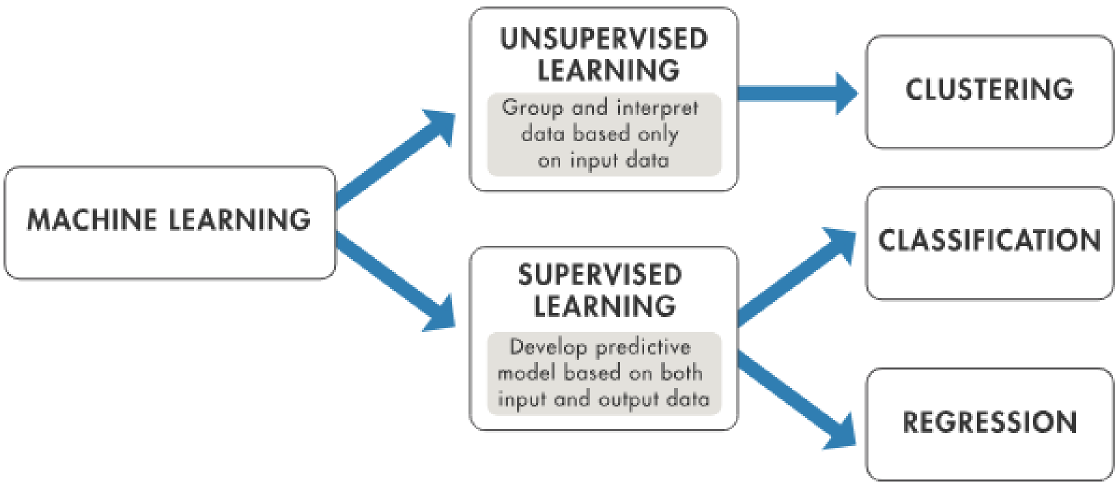

UNSUPERVISED LEARNING
Group and interpret data based only on input data
CLUSTERING
MACHINE LEARNING
CLASSIFICATION
SUPERVISED LEARNING
Develop predictive model based on both input and output data
REGRESSION

Figure 2.1: Comparison between supervised and unsupervised Machine Learning [7]

Throughout this thesis project, supervised ML models will be used; therefore, the analysis will focus on them.

2.4 Classification and Regression

As mentioned earlier, supervised learning can be divided into two main problem categories depending on the type of output the model must produce: classification and regression.

2.4.1 Classification

Classification makes it possible to predict discrete categories. An algorithm receives input data and assigns them to predefined classes (for example, determining whether an email is spam or legitimate, or performing automatic handwriting recognition).

2.4.2 Regression

Regression is employed when the expected result is a continuous variable, with the aim of estimating a functional relationship between independent and dependent variables.

In both cases, the model is trained on a labelled dataset in which both the inputs and the corresponding outputs are known. The algorithm learns a function that maps inputs to outputs in order to make predictions on previously unseen data.

Learning quality is assessed using error functions that measure the distance between the model's predictions and the expected results. [8, 7, 9]

In the present case, regression is required because rotor performance needs to be analysed.

2.5 Overfitting and Underfitting

Overfitting and underfitting are two issues that may arise when training ML models, as both result in poor performance in regression or classification. The two concepts are briefly introduced below.

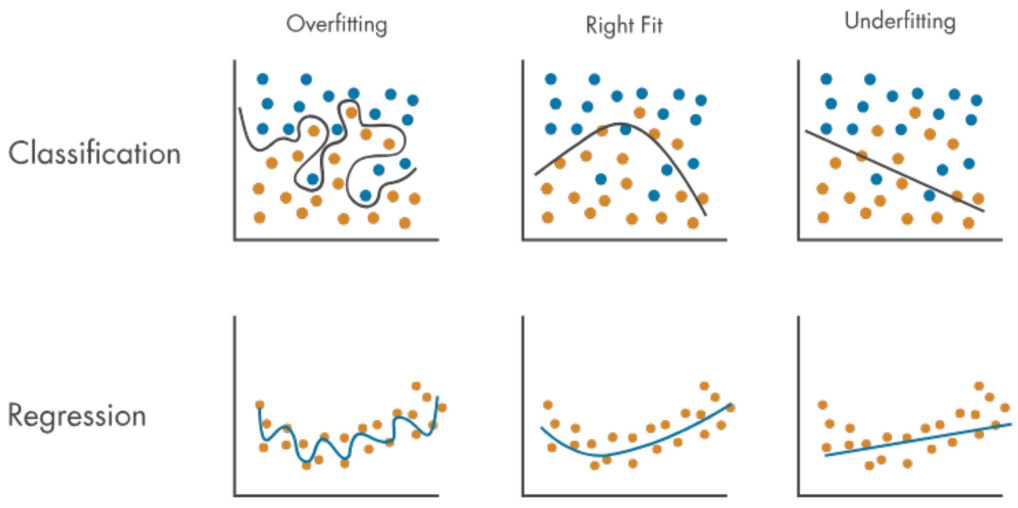


Figure 2.2: Overfitting and Underfitting [10]

2.5.1 Overfitting

Overfitting occurs when the model fits the training data excessively and memorizes noise or irregularities, as illustrated in Figure 2.2. This leads to poor generalisation on new data. It is typically caused by:

- excessive model complexity (high variance);
- a small or noisy dataset;
- the presence of irrelevant features.

Overfitting can be detected when the training (validation) error is much smaller than the test error. [11]

2.5.2 Underfitting

Underfitting represents the other side of the coin: it occurs when the model is too simple to capture the complexity of the data. Again, the model cannot make accurate predictions on new data. The main causes that can lead a model to underfit are:

- a model that is too simple or has too few parameters (high bias);
- insufficient training;
- poorly representative data.

Unlike overfitting in this case, errors are high in both validation and test phases. [12]

The main features of overfitting and underfitting can be summarised as shown in Table 2.1.

Feature	**Overfitting**	**Underfitting**
Model complexity	Too complex	Too simple
Performance on training set	High	Low
Performance on test set	Low	Low
Generalisation	Poor	Poor

Table 2.1: Comparison between Overfitting and Underfitting

2.5.3 Avoiding Overfitting and Underfitting

Having ML models that exhibit either of the two problems described above represents an obstacle to the success of the intended project. Several approaches can improve model generalisation on new data; some of them are presented below.

k-fold Cross-Validation

Cross-validation is a validation technique that allows all data to be used both to train and to validate the model. Depending on the chosen number of folds k, the dataset is divided into k equal parts. Of these, k - 1 parts are used to train the model, while the remaining part is used for validation. This process is repeated k times, as shown in Figure 2.3, and the errors computed at each iteration are then averaged. Using this technique, the computational cost of training increases and, in some cases, if the model is heavy to train, it becomes unsustainable without sufficient computing power. [14]

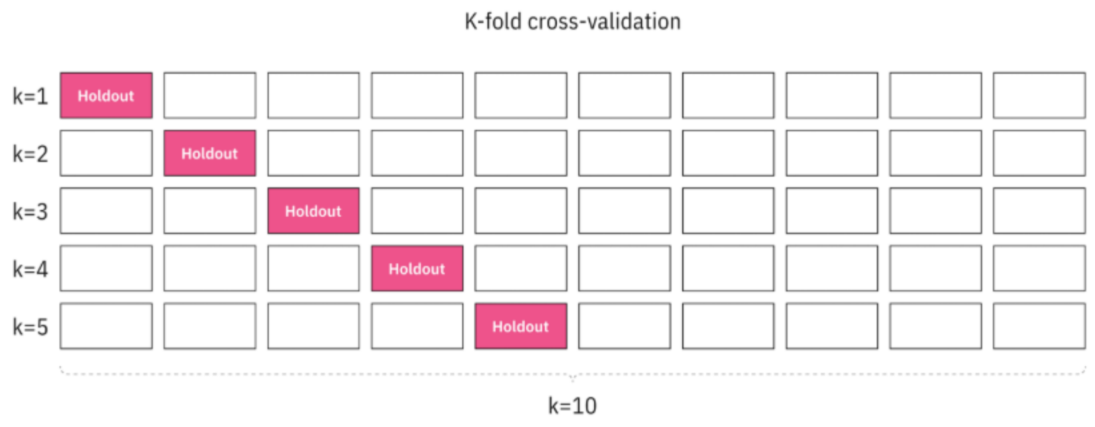


Figure 2.3: k-fold cross-validation [13]

Feature Selection

Feature selection is a technique used to reduce the likelihood of the model overfitting. It limits the information the model receives by identifying the variables that are most relevant for predicting the target variable. This makes it possible to remove from the dataset all unnecessary information that could generate noise and worsen model performance. Reducing the number of input variables allows the model to focus on relevant informative signals, limiting the learning of spurious correlations that do not replicate in future data. The most commonly used methods for feature selection are:

- Filter Methods;
- Wrapper Methods;
- Embedded Methods.

Other methods, such as the Genetic Algorithm (GA), can also be employed[1]. [15]

Parameter Tuning

Every ML model is characterised by settings (hyperparameters) that govern its operation. By properly adjusting hyperparameters, it is possible to improve the model's ability to generalise[2].

[1]This thesis will use this method for feature selection. The details of its operation will be explained in a later chapter.

[2]Hyperparameter optimisation can enhance a model's generalisation capability but, if not conducted properly, it risks leading to overfitting.

2.6 Machine Learning Pipeline

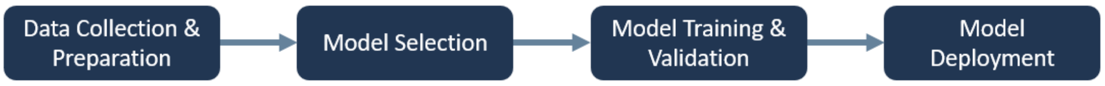


Figure 2.4: Machine Learning Pipeline

This section briefly describes the steps required to train and deploy a ML model. Specifically, the steps are:

- data collection, cleaning, and preparation;
- ML model selection;
- model training and validation;
- model deployment. [16]

2.6.1 Data Collection, Cleaning, and Preparation

Data Collection

The data collection phase is the essential starting point for training a ML model. Performing it accurately is crucial to guarantee the quality of the final model. Data collection consists of obtaining a sufficiently large[3] and representative set of raw data from reliable sources. The data must be consistent with the problem to be solved and they must contain all variables useful for the intended task.

Data Cleaning

Data cleaning involves identifying and correcting any anomalies, such as:

- missing or null values;
- duplicate values;
- outliers (extreme anomalous values);
- formatting or encoding errors.

Dirty or inconsistent data can compromise model training, introducing bias or leading to unreliable results.

[3]Having an insufficient amount of data could limit model performance or, in some cases, make training impossible.

Data Preparation

Finally, data preparation (preprocessing) transforms the data into a form suitable for the ML algorithm. All database entries must be in the same format. Typical operations in this phase are:

- normalising or standardising numerical variables;
- selecting relevant variables (feature selection);
- splitting data into inputs (x) and targets/outputs (y).

Proper data preparation improves model effectiveness, reduces the risk of overfitting and speeds up training times.

2.6.2 Machine Learning Model Selection

Choosing the model to use is one of the most critical steps in the ML process and depends on various factors, including:

- *Type of problem*: classification, regression, clustering, and so on;
- *Problem complexity*: linearity or non linearity of relationships between variables;
- *Quantity and quality of available data*;
- *Desired interpretability*;
- *Available time and computational resources.*

Numerous supervised ML models exist, each with specific characteristics, advantages, and limitations. The most commonly used are:

- **Linear regression**: simple but effective for linear data or data with few variables. It has low computational cost and it trains quickly even on large datasets.
- **Decision trees and Random Forests**: excellent for structured data and easy to interpret. Computational cost increases with tree depth and the number of trees in the forest, but it remains manageable for medium sized datasets.
- **Support Vector Machines (SVM)**: suitable for complex problems with clear margins between classes. Computational cost can become high, especially with non linear kernels and very large datasets.

- **K-Nearest Neighbors (KNN)**: simple to implement but computationally expensive on large datasets, as it requires computing distances to all training points for each prediction.
- **Neural networks (NN)**: effective on complex non linear problems, but less interpretable and more demanding computationally, particularly for deep models with many parameters to optimise.
- **Gaussian Process Regression (GPR)**: a powerful non parametric probabilistic model for regression problems. It provides both an expected value estimate and a measure of predictive uncertainty. However, computational cost grows cubically with the number of data points, limiting its use on large datasets.

The initial choice of a model is not the final one: it is often necessary to test several algorithms, compare their performance through validation methods and select the one that offers the best compromise among accuracy, generalisation, and complexity.

2.6.3 Model Training and Validation

Once the ML models to be trained have been chosen, the training phase can begin. At this point, it becomes necessary to identify a metric that helps determine which of the trained models might yield the best results.
To evaluate model performance and avoid issues such as overfitting, the dataset is divided into three subsets:

- Training set;
- Validation set;
- Test set.

Training Set

Training data are used only to train the models. These data enable the model to learn the relationships among the various variables, seeking to understand how their values or changes relate to the value or change of the target variable.

Validation Set

Validation data are unseen by the algorithm and they are used to check the model's behaviour in the presence of new data. So it is necessary to define performance metrics, such as:

- **RMSE (Root Mean Squared Error)**: RMSE measures the square root of the mean of squared errors.

$$RMSE = \sqrt{\frac{1}{n}\sum\left(y_{pred} - y_{true}\right)^2} \tag{2.1}$$

  where $y_{pred,i}$ is the model's predicted value for the i-th example, $y_{true,i}$ is the actual value, and n is the total number of examples. A lower RMSE indicates better model performance. RMSE penalises large errors more heavily and is useful when high precision is required.

- **MSE (Mean Squared Error)**: MSE measures the mean of squared errors between predicted and actual values.

$$MSE = \frac{1}{n}\sum\left(y_{pred} - y_{true}\right)^2 \tag{2.2}$$

  Like RMSE, MSE strongly penalises large errors.

- **MAE (Mean Absolute Error)**: MAE calculates the mean of absolute errors between predicted and actual values too.

$$MAE = \frac{1}{n}\sum\left|y_{pred} - y_{true}\right| \tag{2.3}$$

  Unlike RMSE, MAE is less sensitive to outliers.

- ***R*²**: measures how well the model explains data variability. It ranges between 0 and 1, where 1 indicates perfect predictive capability.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{2.4}$$

  - $SS_{res} = \sum\left(y_{true} - y_{pred}\right)^2$ is the residual sum of squares;
  - $SS_{tot} = \sum\left(y_{true} - y_{mean}\right)^2$ is the total sum of squares.

Using these metrics for each model makes it possible to choose the best one and to move to the next phase[4]. If validation set performance is poor, the model can be improved, for example by adjusting hyperparameters and repeating the training phase. This iterative process continues until satisfactory results are obtained. Some validation techniques are:

[4]The choice of metric must align with the model's objectives: for example, in domains where large errors are more penalising, RMSE is preferable to MAE.

- **Resubstitution**: evaluates the model on the same data used for training (training set equals validation set). It is very simple to implement but tends to give overly optimistic results and it is unsuitable for estimating generalisation ability.
- **Hold-out**: simple division of the dataset into training, validation, and test sets.
- **k-Fold Cross-Validation**: splits the dataset into k subsets so that the model is trained and validated k times, changing the validation fold each time.
- **Leave-One-Out Cross-Validation (LOOCV)**: a k-Fold variant where k equals the number of observations. Each example is used as the test case exactly once.

To avoid unrealistic estimates of model performance, *data leakage* (such as including the same rotor in both the training and validation sets) should be prevented, as it compromises assessment of the models' true predictive capability.

Test Set

Once the model achieves satisfactory results on the validation set, the test set can also be used. This set may be omitted, as it serves to evaluate and visualise the model's performance and behaviour.

In general, the dataset is split as follows[5] (or similarly): 70% train, 20% validation, and 10% test.

2.6.4 Model Deployment

After completing all the steps described, the chosen model is ready to be integrated into the code or the application for which it was designed.
The main deployment steps are:

- **Model saving**: after training, the model is saved in a format that allows reuse (for example, a `.mat` file in MATLAB).
- **Loading into the production environment**: the model is loaded into an operational environment where it can receive new input data to generate predictions.

[5]Considering Hold-out validation.

- **Input data preprocessing**: before new data are fed to the model, they must undergo the same type of preprocessing used during training (for example, normalisation or feature selection).
- **Prediction**: data are supplied to the model, which returns the desired prediction.

2.7 Conclusions

In conclusion, this chapter introduced the fundamentals of ML, highlighting the different learning types, common issues, and strategies for proper model design. The following chapters present the execution of the steps shown here for the case under study and delve deeper into the techniques and models employed.

Chapter 3

Data Collection

3.1 Introduction

It is essential to have data to train the ML models on, so that they can learn the relationships that exist between rotor characteristics and their performance. Several approaches could have been followed during the development of this thesis:

1. purchase and experimental analysis of various rotors under different operating conditions;
2. use of existing databases;
3. CFD analyses of several rotors.

The choice fell on the second option, because the third one would have required excessive time, while the first one would also have involved unsustainable expenses.

3.2 Database

The selected database is the one hosted on the University of Illinois website [17]. The site contains experimental data collected on various rotors from 2005 to 2022. Specifically, the database is divided as follows:

- volume 1: UIUC MS thesis by John Brandt and following tests (2005-2008);
- volume 2: UIUC PhD dissertation by Robert Deters and following tests (2009-2015);
- volume 3: publication and tests by Or Dantsker (2020);
- volume 4: publication and tests by Or Dantsker (2021-2022).

Together, the four volumes contain the experimental results of about 270 rotors. The data are subdivided as follows:

- volume 1: roughly 140 rotors used on small UAVs and model aircraft. The rotor families in this volume are:
  - Aeronaut;
  - APC;
  - Graupner;
  - GWS;
  - Kavon;
  - Kyosho;
  - Master Airscrew;
  - Rev Up;
  - Zingali.
- volume 2: around 70 small scale propellers. The rotor families in this volume are:
  - APC;
  - Crazyflie;
  - Da4002;
  - DA4022;
  - DA4052;
  - E-Flite;
  - GWS;
  - KP;
  - Micro Invent;
  - NR640;
  - Plantraco;
  - Union;
  - Vapor.
- volume 3: 40 Aero-Naut CAM carbon fiber folding propellers;
- volume 4: 17 APC Thin Electric 2-bladed.

Not all of these roughly 270 rotors could be used, because rotors with more than two blades and those lacking geometric data were discarded. Therefore the total number of rotors considered amounts to 162.

3.2.1 Geometry data

Within the database, rotor geometry data are provided as shown in Table 3.1.

r/R	**c/R**	**beta**
0.15	0.141	31.67
0.20	0.147	37.59
0.25	0.183	38.78
0.30	0.207	35.90
0.35	0.218	32.07
0.40	0.223	28.50
0.45	0.222	25.81
0.50	0.217	23.58
0.55	0.209	21.66
0.60	0.197	19.99
0.65	0.183	18.58
0.70	0.167	17.29
0.75	0.150	16.37
0.80	0.132	15.46
0.85	0.114	14.30
0.90	0.098	13.40
0.95	0.075	12.02
1.00	0.051	10.61

Table 3.1: Geometric data of the rotor APC Thin Electric 9 X 6

As the table shows, for each rotor only the distributions of normalised chord and profile incidence angle along the blade span are available.

From the rotor name it is also possible to deduce the diameter and the pitch; for instance, in the rotor APC Thin Electric 9 $\times$ 6, the first number indicates the diameter in inches, while the second represents the pitch in inches per revolution. For many rotors in the database, however, this geometric information is not available directly in this format; in particular, for many APC rotors the data were taken from the manufacturer's website [18], where they are given as in Table 3.2. It was necessary to extract the data from the columns `STATION`, `CHORD` and `TWIST` and interpolate them[1] at the stations listed in Table 3.1.

STATION (in)	CHORD (in)	PITCH (QUOTED)	PITCH (LE-TE)	PITCH (PRATHER)	SWEEP (in)	THICKNESS RATIO	TWIST (deg)	MAX-THICK (in)	CROSS-SECTION (in^2)	ZHIGH (in)	CGY (in)	CGZ (in)
1.0000	0.8667	6.0000	5.9969	5.4265	0.4034	0.1561	43.6646	0.1353	0.0749	0.3541	0.1212	0.1289
1.0500	0.8902	6.0000	6.0000	5.4638	0.4090	0.1515	42.2852	0.1349	0.0764	0.3554	0.1140	0.1315
1.1000	0.9105	6.0000	6.0000	5.4957	0.4141	0.1472	40.9618	0.1340	0.0775	0.3538	0.1075	0.1318
1.1500	0.9278	6.0000	6.0000	5.5262	0.4188	0.1430	39.7053	0.1327	0.0781	0.3492	0.1016	0.1297
1.2000	0.9435	6.0000	6.0000	5.5549	0.4231	0.1392	38.5119	0.1313	0.0784	0.3418	0.0960	0.1250
1.2500	0.9577	6.0000	6.0000	5.5820	0.4270	0.1355	37.3778	0.1298	0.0787	0.3333	0.0906	0.1194
1.3000	0.9706	6.0000	6.0000	5.6073	0.4304	0.1321	36.2996	0.1283	0.0788	0.3248	0.0853	0.1138
1.3551	0.9833	6.0000	6.0000	5.6332	0.4337	0.1287	35.1715	0.1265	0.0788	0.3155	0.0797	0.1077
1.4492	1.0014	6.0000	6.0000	5.6723	0.4382	0.1234	33.3824	0.1236	0.0786	0.2995	0.0706	0.0974
1.5483	1.0157	6.0000	6.0000	5.7063	0.4415	0.1188	31.6641	0.1207	0.0783	0.2827	0.0616	0.0866
1.6475	1.0254	6.0000	6.0000	5.7324	0.4433	0.1151	30.0976	0.1180	0.0780	0.2658	0.0532	0.0756
1.7467	1.0306	6.0000	6.0000	5.7503	0.4437	0.1124	28.6661	0.1158	0.0777	0.2490	0.0454	0.0647
1.8458	1.0317	6.0000	6.0000	5.7596	0.4427	0.1106	27.3545	0.1141	0.0776	0.2322	0.0383	0.0536
1.9450	1.0289	6.0000	6.0000	5.7611	0.4405	0.1096	26.1495	0.1128	0.0775	0.2155	0.0318	0.0424
2.0442	1.0223	6.0000	6.0000	5.7599	0.4369	0.1088	25.0396	0.1112	0.0771	0.1988	0.0258	0.0311
2.1433	1.0123	6.0000	6.0000	5.7585	0.4322	0.1080	24.0146	0.1094	0.0762	0.1821	0.0204	0.0199
2.2425	0.9991	6.0000	6.0000	5.7567	0.4264	0.1073	23.0659	0.1072	0.0749	0.1655	0.0155	0.0088
2.3417	0.9829	6.0000	6.0000	5.7533	0.4195	0.1066	22.1856	0.1048	0.0730	0.1490	0.0111	-0.0021
2.4408	0.9639	6.0000	6.0000	5.7472	0.4115	0.1059	21.3670	0.1021	0.0708	0.1327	0.0072	-0.0129
2.5400	0.9424	6.0000	6.0000	5.7404	0.4026	0.1053	20.6041	0.0992	0.0682	0.1164	0.0039	-0.0233
2.6392	0.9185	6.0000	6.0000	5.7336	0.3928	0.1047	19.8916	0.0962	0.0653	0.1003	0.0010	-0.0336
2.7383	0.8927	6.0000	6.0000	5.7271	0.3821	0.1042	19.2250	0.0930	0.0621	0.0842	-0.0014	-0.0436
2.8375	0.8649	6.0000	6.0000	5.7191	0.3706	0.1036	18.6001	0.0897	0.0586	0.0683	-0.0034	-0.0534
2.9367	0.8356	6.0000	6.0000	5.7113	0.3584	0.1032	18.0132	0.0862	0.0549	0.0526	-0.0049	-0.0630
3.0358	0.8050	6.0000	6.0000	5.7036	0.3455	0.1027	17.4611	0.0827	0.0511	0.0370	-0.0062	-0.0724
3.1350	0.7732	6.0000	6.0000	5.6964	0.3319	0.1023	16.9409	0.0791	0.0472	0.0215	-0.0073	-0.0816
3.2342	0.7405	6.0000	6.0000	5.6898	0.3178	0.1019	16.4499	0.0755	0.0433	0.0063	-0.0082	-0.0907
3.3333	0.7072	6.0000	6.0000	5.6834	0.3032	0.1015	15.9859	0.0718	0.0395	-0.0088	-0.0090	-0.0996
3.4325	0.6734	6.0000	6.0000	5.6774	0.2881	0.1012	15.5467	0.0682	0.0358	-0.0237	-0.0099	-0.1084
3.5317	0.6395	6.0000	6.0000	5.6718	0.2726	0.1009	15.1304	0.0646	0.0322	-0.0383	-0.0110	-0.1172
3.6308	0.6056	6.0000	6.0000	5.6667	0.2568	0.1007	14.7354	0.0610	0.0288	-0.0528	-0.0123	-0.1261
3.7300	0.5720	6.0000	6.0000	5.6617	0.2407	0.1005	14.3601	0.0575	0.0257	-0.0670	-0.0141	-0.1350
3.8292	0.5388	6.0000	6.0000	5.6571	0.2243	0.1003	14.0030	0.0540	0.0228	-0.0809	-0.0162	-0.1440
3.9283	0.5065	6.0000	6.0000	5.6531	0.2077	0.1001	13.6629	0.0507	0.0202	-0.0946	-0.0187	-0.1530
4.0275	0.4751	6.0000	6.0000	5.6482	0.1911	0.1000	13.3386	0.0475	0.0177	-0.1081	-0.0218	-0.1620
4.1267	0.4449	6.0000	6.0000	5.6253	0.1743	0.1000	13.0292	0.0445	0.0156	-0.1219	-0.0254	-0.1718
4.2258	0.4162	6.0000	6.0000	5.5730	0.1576	0.0999	12.7335	0.0416	0.0136	-0.1360	-0.0296	-0.1824
4.3235	0.3712	6.0000	6.0000	5.3711	0.1232	0.0999	12.4549	0.0371	0.0108	-0.1546	-0.0440	-0.1960
4.4105	0.2850	6.0000	6.0000	4.9817	0.0460	0.0999	12.2167	0.0285	0.0064	-0.1824	-0.0828	-0.2145
4.5000	0.0088	6.0000	6.0240	6.0240	-0.2185	0.1000	12.0274	0.0009	0.0000	-0.2529	0.0000	0.0000

Table 3.2: Geometric data of the rotor APC Thin Electric 9 X 6 — APC website [18]

3.2.2 Static data

For each analysed rotor, hover performance data are provided as shown in Table 3.3[2]. In the table, RPM represents the rotor speed in revolutions per minute, while CT (thrust coefficient) and CP (power coefficient) are calculated as follows:

$$CT = \frac{T}{\rho n^2 D^4} \tag{3.1}$$

$$CP = \frac{P}{\rho n^3 D^5} \tag{3.2}$$

where:

[1]A linear interpolation was used, and values outside the interval in the manufacturer's table were extrapolated.

[2]The RPM values available for individual rotors are not the same for all rotors.

- T: thrust in Newton;
- P: power in Watt;
- ρ: air density in kg/m^3;
- n: revolutions per second;
- D: diameter in metre.

RPM	**CT**	**CP**
2333	0.1096	0.0541
2620	0.1095	0.0528
2901	0.1117	0.0526
3145	0.1121	0.0526
3476	0.1132	0.0525
3779	0.1137	0.0522
4079	0.1139	0.0521
4353	0.1141	0.0518
4667	0.1142	0.0515
4963	0.1145	0.0514
5251	0.1151	0.0514
5537	0.1158	0.0515
5841	0.1159	0.0515
6108	0.1160	0.0513
6415	0.1165	0.0515
6717	0.1169	0.0516

Table 3.3: Static data of the APC Thin Electric 9 X 6 rotor

3.2.3 Dynamic data

The database also includes a large number of experimental data under dynamic conditions, i.e. with $J > 0$. The advance ratio J is defined as:

$$J = \frac{V}{nD} \tag{3.3}$$

where V is the vertical velocity of the rotor expressed in m/s. Static data clearly represent the special case in which $J = 0$. As mentioned, for each rotor several dynamic datasets are provided, varying both J and RPM. For a fixed rotational speed[3] the data are supplied as in Table 3.4.

J	**CT**	**CP**	η
0.158	0.1349	0.0728	0.292
0.197	0.1290	0.0718	0.355
0.239	0.1222	0.0702	0.417
0.275	0.1163	0.0688	0.466
0.325	0.1088	0.0670	0.528
0.361	0.1022	0.0651	0.566
0.400	0.0965	0.0638	0.606
0.442	0.0898	0.0618	0.642
0.478	0.0841	0.0600	0.670
0.521	0.0762	0.0569	0.697
0.554	0.0699	0.0545	0.711
0.600	0.0614	0.0510	0.722
0.648	0.0515	0.0464	0.718
0.677	0.0456	0.0439	0.704
0.721	0.0352	0.0388	0.654
0.759	0.0258	0.0340	0.577
0.806	0.0141	0.0285	0.399

Table 3.4: Dynamic data of the APC Thin Electric 9 X 6 rotor — RPM = 4016

The efficiency η can be calculated in two ways:

$$\eta = \frac{CT \cdot J}{CP} \tag{3.4}$$

$$\eta = \frac{T \cdot V}{P} \tag{3.5}$$

Since η can be obtained from the other data in the table, it will not be used to train the ML models.
Each rotor has data for roughly five or six RPM values, which greatly increases the amount of available data.

[3]The RPM value is indicated in the file name. For example, `apcsf_9x6_kt0980_4016.txt` indicates RPM = 4016.

3.3 Data merging

After downloading all the files in the database, it is necessary to create a matrix[4] that combines the data, so that they can be used to train the ML models. Because of this, a MATLAB script was written to perform the task automatically.

3.3.1 MATLAB script operation

To compare models trained with different databases, the script is designed so that it can be chosen whether to use only the static data or also the dynamic data. In this way the models can be trained using either a reduced database or the complete one.
The files downloaded from the database are saved in three separate folders:

- geometry data folder;
- static data folder;
- dynamic data folder.

Depending on the type of database to be used, the script extracts the data from the relevant folders and it builds the matrix for training the ML models.

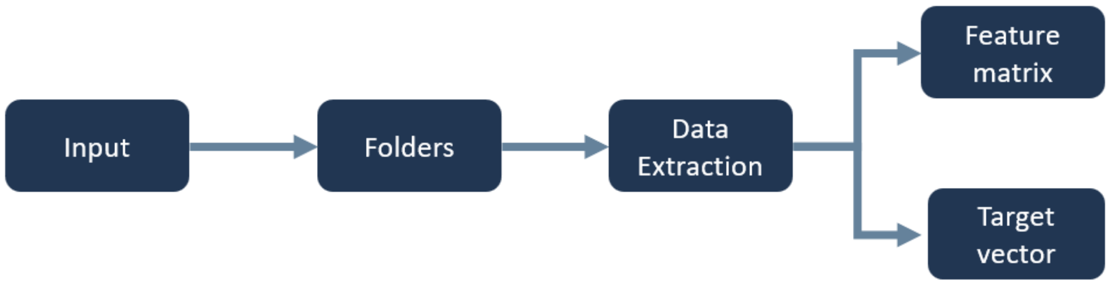


Figure 3.1: MATLAB code flow chart

Static database

The static database combines the rotor geometry data with only the hover data (static data folder). In the matrix, the data are arranged as follows:

- columns 1–18: c/R data;

[4]In the matrix each row represents a different rotor or operating condition.

- columns 19–36: β data in degrees.

The composition of the last columns depends on whether it is used a dimensional or non dimensional model:

- dimensional model:
  - column 37: angular velocity (Ω) in rad/s;
  - column 38: radius (R) in m.
- non dimensional model:
  - column 37: Mach number
$$M = \frac{\Omega \cdot R}{c_{SL}} \tag{3.6}$$
    where the speed of sound is taken as $c_{SL} = 340.29\frac{m}{s}$;
  - column 38: Reynolds number at 75 % span
$$Re_{75} = \frac{\rho_{SL} \cdot (\Omega \cdot R \cdot 0.75) \cdot C_{75}}{\mu_{SL}} \tag{3.7}$$
    with $\rho_{SL} = 1.225\frac{kg}{m^3}$ and $\mu_{SL} = 1.79 \cdot 10^{-5} Pa \cdot s$;

Since the database provides no information about the conditions under which the experimental data were acquired, sea level values of density and viscosity are used by default.
A total of 2611 different data are available, so the matrix is 2611 $\times$ 38.

Columns 1–18 c/R	Columns 19–36 β	Column 37 Angular velocity [rad/s]	Column 38 Radius [m]
Data 1			
Data 2			
…			
Data 2611			

Table 3.5: Data arrangement for the dimensional static case

Columns 1–18 c/R	Columns 19–36 β	Column 37 Mach	Column 38 Reynolds
Data 1			
Data 2			
…			
Data 2611			

Table 3.6: Data arrangement for the non dimensional static case

Dynamic database

The dynamic database includes, in addition to the static data, the dynamic data (dynamic data folder). The first 36 columns are arranged as in the static case, while the last columns are:

- dimensional model:
  - column 37: J;
  - column 38: angular velocity in rad/s;
  - column 39: radius in m.
- non dimensional model:
  - column 37: J;
  - column 38: Mach number;
  - column 39: Reynolds number at 75 % span.

As mentioned earlier, the available data increase greatly compared with the static database, and the resulting matrix is 18910 $\times$ 39.

Columns 1–18	Columns 19–36	Column 37	Column 38	Column 39
c/R	β	J	Angular velocity [rad/s]	Radius [m]
Data 1				
Data 2				
...				
Data 18910				

Table 3.7: Data arrangement for the dimensional dynamic case

Columns 1–18	Columns 19–36	Column 37	Column 38	Column 39
c/R	β	J	Mach	Reynolds
Data 1				
Data 2				
...				
Data 18910				

Table 3.8: Data arrangement for the non dimensional dynamic case

3.3.2 Output

The output vector is a column vector containing the experimental results, consisting of either the CT or CP values, depending on whether the model is trained to predict one or the other.

3.4 Conclusions

The rotor geometry data used to train the ML models include only the distributions of chord and β along the span. These data should be sufficient for a good prediction of rotor performance, but they cannot be fully accurate owing to the lack of information on:

- the airfoil type used;
- the leading edge coordinates of the airfoils along the span;
- the environmental conditions under which the data were acquired.

Indeed, informations on the airfoil (and leading edge coordinates) are available only for APC rotors. However, since there are not many rotor types in this family (three or four) and each type has the same airfoil, there are insufficient data to train the models.
In general, it should also be noted that the data provided by the database themselves have an inherent uncertainty both in the geometric measurements of the rotors and in the experimental results.

Chapter 4

Machine Learning Models

4.1 Introduction

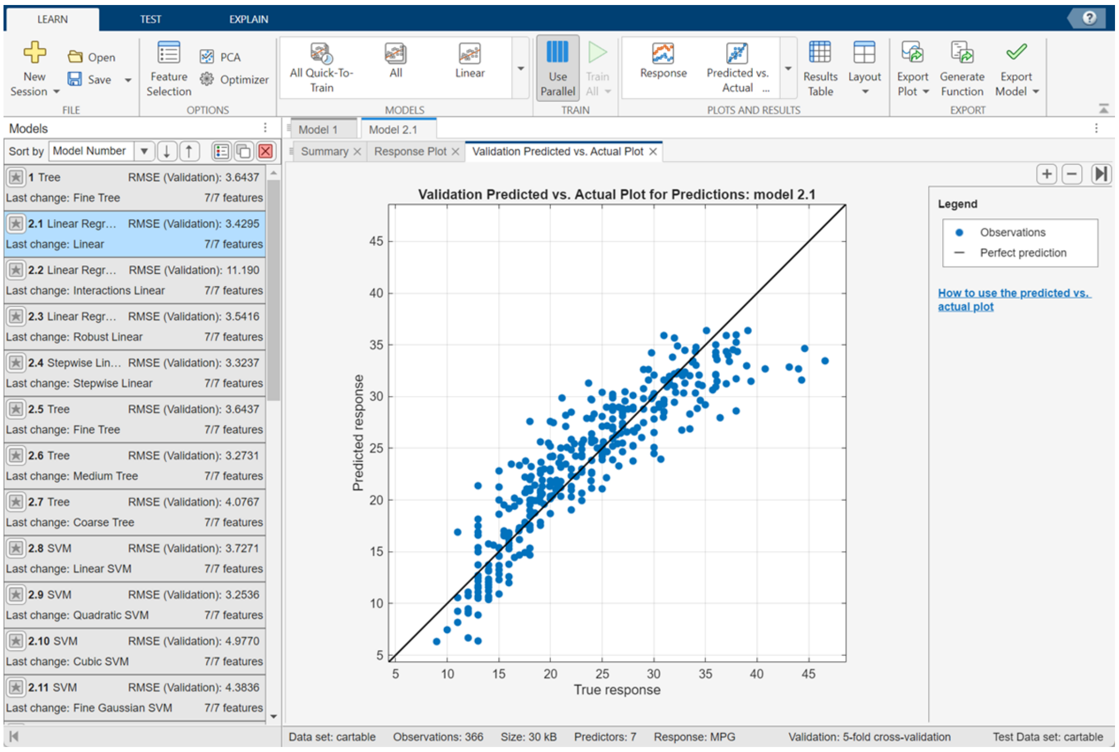
This chapter describes MATLAB's Regression Learner environment, which is used to train and compare various supervised regression models. The main algorithms available in the application are presented, with a focus on their characteristics, advantages and limitations. Finally, the criteria and methodology adopted for selecting the most suitable ML models for the development of this thesis are outlined.

4.2 Regression Learner

The Regression Learner is a MATLAB app [19] that enables supervised learning for data prediction using regression models. To train the models, both input and output data must be provided. It is possible to:

- explore the data, select variables, choose a validation method, train and optimise models;
- use automated training to find the best model among several options;
- compare models with result tables and plots;
- evaluate performance on test data.

Once the model has been trained, it can be exported to MATLAB. Unfortunately, the app currently lacks the option to perform group cross-validation. For this reason, as will be seen later, some analyses must be implemented manually in MATLAB code. The operating scheme of the application is shown in Figure 4.2.



1 Tree RMSE (Validation): 3.6437
Last change: Fine Tree 7/7 features
2.1 Linear Regr… RMSE (Validation): 3.4295
Last change: Linear 7/7 features
2.2 Linear Regr… RMSE (Validation): 11.190
Last change: Interactions Linear 7/7 features
2.3 Linear Regr… RMSE (Validation): 3.5416
Last change: Robust Linear 7/7 features
2.4 Stepwise Lin… RMSE (Validation): 3.3237
Last change: Stepwise Linear 7/7 features
2.5 Tree RMSE (Validation): 3.6437
Last change: Fine Tree 7/7 features
2.6 Tree RMSE (Validation): 3.2731
Last change: Medium Tree 7/7 features
2.7 Tree RMSE (Validation): 4.0767
Last change: Coarse Tree 7/7 features
2.8 SVM RMSE (Validation): 3.7271
Last change: Linear SVM 7/7 features
2.9 SVM RMSE (Validation): 3.2536
Last change: Quadratic SVM 7/7 features
2.10 SVM RMSE (Validation): 4.9770
Last change: Cubic SVM 7/7 features
2.11 SVM RMSE (Validation): 4.3836
Last change: Fine Gaussian SVM 7/7 features

Data set: cartable Observations: 366 Size: 30 kB Predictors: 7 Response: MPG Validation: 5-fold cross-validation Test Data set: cartable

Figure 4.1: Regression Learner screen [20]

As mentioned, the Regression Learner includes several ML models:

- Linear regression models;
- Regression Trees;
- Support Vector Machines;
- Efficiently Trained Linear Regression models;
- Gaussian Process Regression models;
- Kernel Approximation Regression models;
- Ensembles of Trees;
- Neural Networks.

A brief introduction to each of these models is provided below to analyse their main features.

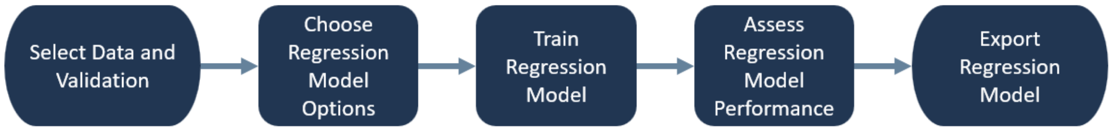

Select Data and Validation
Choose Regression Model Options
Train Regression Model
Assess Regression Model Performance
Export Regression Model

Figure 4.2: Models training flow chart in Regression Learner [19]

4.2.1 Linear regression models

Linear regression is one of the simplest and most widely used methods for supervised prediction [4]. The goal of these models is to estimate a linear function that approximates the relationship between the input variables and the output variable. The general form is:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + \varepsilon \tag{4.1}$$

where w_i are the coefficients to be estimated, x_i the input variables and ε an error term.
In MATLAB's Regression Learner several variants are available:

- Linear;
- Interactions Linear;
- Robust Linear;
- Stepwise Linear.

The advantages of these models are:

- high interpretability and ease of use;
- suitability as a baseline for initial comparisons;
- short training times.

On the other hand, the disadvantages are:

- poor ability to model non linear relationships;
- sensitivity to outliers.

Computational cost

The computational cost of linear regression is generally very low, even on large datasets. The more complex variants are *Stepwise* and *Robust*, yet they remain suitable for applications with limited computational resources.

4.2.2 Regression Trees

Regression Trees are predictive models based on tree structures that recursively split the feature space into regions increasingly homogeneous with respect to the target variable [5]. Each node of the tree represents a split condition on a feature, while the leaves contain the prediction.
In the Regression Learner various tree parameters can be set, such as the maximum depth or the minimum number of observations per node. The following variants are available:

- Fine Tree;
- Medium Tree;
- Coarse Tree.

The advantages are:

- ease of interpretation and visualisation;
- good handling of non linear data and categorical variables;
- little sensitivity to preprocessing.

On the other hand, the disadvantages are:

- tendency to overfit if not properly pruned;
- model instability: small data variations can produce very different trees.

Computational cost

The computational cost of regression trees is moderate. Training time increases with the tree depth and the number of observations, but tree models are quick to train and evaluate.

4.2.3 Support Vector Machines

Support Vector Machine (SVM) regression models aim to find a function that deviates from the true targets by less than a margin ε, penalising only deviations that exceed this threshold [4].
In MATLAB's Regression Learner the following variants are available:

- Linear SVM;
- Quadratic SVM;

- Cubic SVM;
- Fine Gaussian SVM;
- Medium Gaussian SVM;
- Coarse Gaussian SVM.

The advantages are:

- effectiveness in high dimensional spaces;
- robustness to noise thanks to the ε tolerance zone;
- ability to model non linear relationships through kernels.

On the other hand, the disadvantages are:

- high computational cost with large datasets;
- difficulty in tuning hyperparameters.

Computational cost

The computational cost of SVMs is high, especially with non linear kernels and large datasets. Training time grows faster than linearly with the number of samples.

4.2.4 Efficiently Trained Linear Regression models

The Efficiently Trained Linear Regression models category includes optimised versions of linear regression models, implemented to reduce computation time and memory requirements, particularly on large datasets.
These models use efficient numerical techniques to estimate parameters without altering the predictive model structure compared with classical regression.
In MATLAB's Regression Learner the following models are available:

- Efficient Linear Least Squares;
- Efficient Linear SVM.

The advantages are:

- improved computational efficiency;
- suitability for datasets with many observations or features.

The disadvantages are the same accuracy limitations as the ones for ordinary linear regression.

Computational cost

The computational cost of Efficiently Trained Linear Regression Models is low. As noted, they are designed to optimise computational performance using fast numerical solvers, making them ideal for large datasets where the standard version would be slow.

4.2.5 Gaussian Process Regression models

Gaussian Process Regression (GPR) is a non parametric method that assumes the data can be modelled as a multivariate normal distribution [4]. Each prediction point is associated with both an expected mean and an uncertainty estimate (variance).
In the Regression Learner it is possible to choose among:

- Rational Quadratic;
- Squared Exponential;
- Matern 5/2;
- Exponential.

The advantages are:

- provision of probabilistic estimates and confidence intervals;
- high flexibility in modelling non linear phenomena.

On the other hand, the disadvantages are:

- high computational cost ($\mathcal{O}(n^3)$);
- poor scalability to datasets with many thousands of observations.

Computational cost

The computational cost of GPR is very high. As mentioned, training complexity is $\mathcal{O}(n^3)$, where n is the number of observations. Consequently, GPR is poorly scalable and suitable only for relatively small datasets.

4.2.6 Kernel Approximation Regression models

Kernel Approximation Regression models approximate kernel behaviour through projection techniques into a finite space, thereby reducing computational costs compared with exact kernel methods [21].
In the Regression Learner it is possible to choose among:

- SVM Kernel;
- Least Squares Kernel Regression.

The advantages are:

- greater scalability than standard kernel models;
- performance similar to standard kernel models but with reduced computational costs on large datasets.

On the other hand, the disadvantages are:

- possible reduction in accuracy due to the approximation;
- need for an accurate approximation method.

Computational cost

The computational cost of Kernel Approximation Regression models is moderate. Approximation techniques drastically reduce the cost relative to classical kernel models (such as SVMs), making them suitable for large datasets.

4.2.7 Ensembles of Trees

Ensembles combine multiple tree based regression models to create a more robust and accurate predictive model [22].
In the Regression Learner the following are available:

- Boosted Trees;
- Bagged Trees.

The advantages are:

- high accuracy;
- reduction of variance (bagging) or bias (boosting).

On the other hand, the disadvantages are:

- reduced interpretability of individual trees;
- need for greater computational resources.

Computational cost

The computational cost of Ensembles of Trees is medium-high. Cost increases with the number of trees and the complexity of each tree. *Boosted* models require sequential training and they are slower than the *Bagged* ones.

4.2.8 Neural Networks

Neural Networks are models inspired by the functioning of the human brain, composed of neurons organised in layers [12].
In MATLAB's Regression Learner the following models are available:

- Narrow Neural Network;
- Medium Neural Network;
- Wide Neural Network;
- Bilayered Neural Network;
- Trilayered Neural Network.

The advantages are:

- high flexibility for modelling complex, non linear relationships;
- good performance on large datasets.

On the other hand, the disadvantages are:

- reduced interpretability;
- need for careful tuning and possible susceptibility to overfitting.

Computational cost

The computational cost of Neural Network models is high. Neural networks require significant training times, especially with many layers and neurons. Cost grows with the architectural complexity and the number of epochs[1].

The main characteristics of the various models are summarised in Table 4.1.

[1]The number of epochs is a critical hyperparameter because it determines how many times the model learns from the entire dataset.

Model	Computational cost	Interpretability	Accuracy
Linear Regression models	Low	High	Low–Medium
Regression Trees	Moderate	High	Medium
Support Vector Machines	High	Low	High
Efficiently Trained Linear models	Low	High	Medium
Gaussian Process Regression models	Very high	Low	High
Kernel Approximation Regression models	Moderate	Low	Medium–High
Ensembles of Trees	Medium–High	Medium	High
Neural Networks	High	Low	High

Table 4.1: Comparison of the regression models in Regression Learner

4.3 Selection of Machine Learning models

The choice of the ML models used in this thesis is based on a comparative analysis of the errors made on the test set[2] in order to identify those with the best predictive performance. For this comparison the previously created dimensional static database is used.[3] Each time, an increasingly larger percentage of data is excluded in order to be used as a test set and to evaluate the prediction errors of the various models.

4.4 Creation of the test set

The creation of the test set was implemented manually in MATLAB, even though the Regression Learner offers the option to automatically leave part of the data as a test set. The reason for this choice lies in the fact that the Regression Learner draws data from the database to create the test set randomly. This approach is not suitable here because in the database each rotor appears several times, with only the last columns concerning operating conditions differing. This would imply training the model on observations of certain rotors that are simultaneously present in the test set, generating data leakage.
Indeed, as observed in Figure 4.3, the predicted values are practically perfect even though a 10 % test set is considered and despite all the simplifications introduced. Consequently this approach cannot be used.
As schematically shown in Figure 4.4, a percentage of rotors was manually extracted from the database, in order to be used as a test set. This approach is correct

[2]It is not possible to evaluate the models using validation because, as mentioned, group cross-validation cannot be performed in the Regression Learner.

[3]The static database is chosen because some models take a long time to train when the dynamic database is used.

because the models are certainly trained on a dataset that has nothing in common with the test data, thus simulating the condition under which these models are called upon to work, namely predicting the performance of previously unseen rotors.

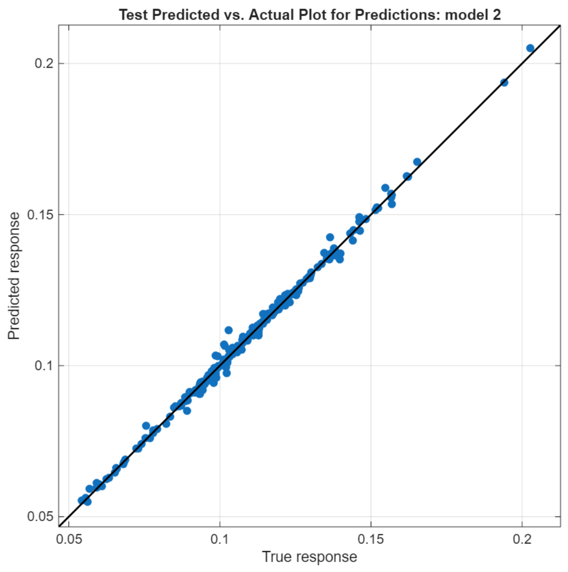


Figure 4.3: GPR Exponential with a 10 % test set

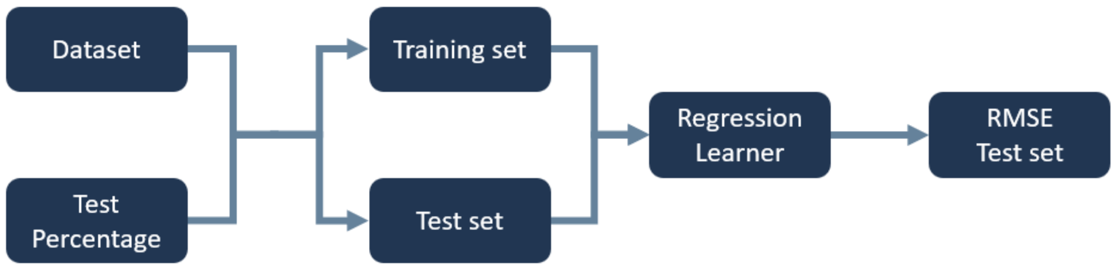


Figure 4.4: Models test flow chart

4.5 Results

Figures 4.5 and 4.6 show the results obtained by varying the percentage of database data used as the test set from one to ten percent and by analysing the RMSE error (Equation (2.1)) of the various ML models.

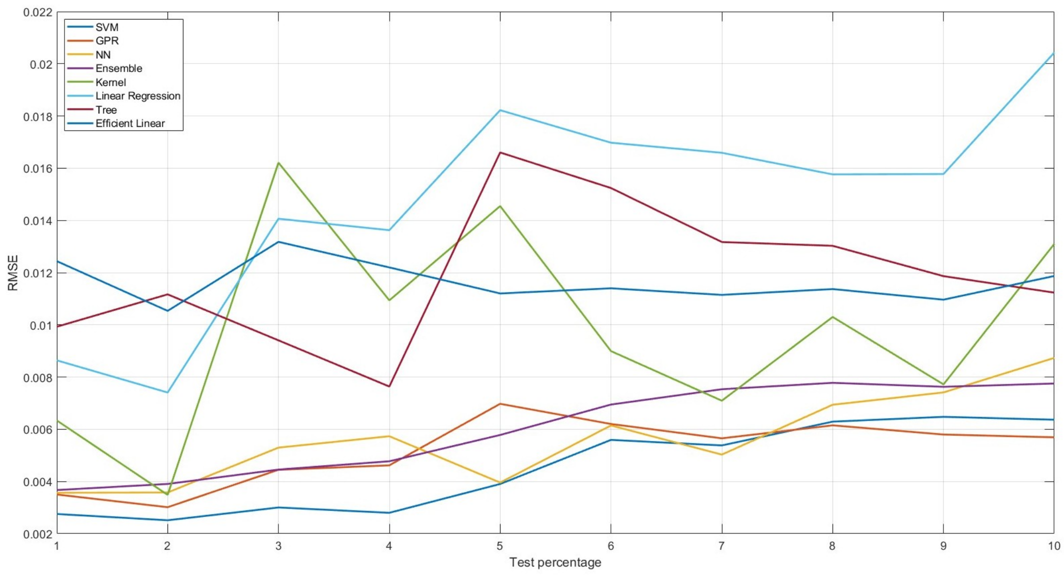


Figure 4.5: RMSE trend versus test set percentage for CT

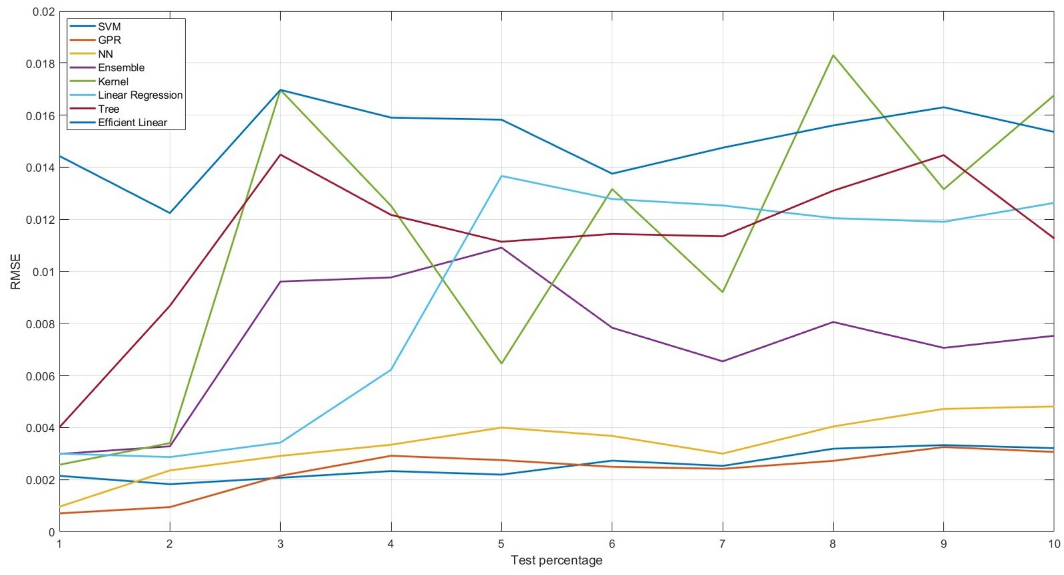


Figure 4.6: RMSE trend versus test set percentage for CP

As can be seen, the tests show lower and more consistent errors for both CT and CP with the Gaussian Process Regression and the Support Vector Machine. In particular, the models that have the lowest average errors on the test set are:

- Exponential GPR;
- Medium SVM.

In the thesis these two models will be implemented so that their performance can be compared.

4.6 Conclusions

This chapter introduced the MATLAB Regression Learner application, used to train the ML models. After an overview of the main algorithms available, a comparative evaluation was carried out based on prediction error on carefully selected test data to avoid data leakage.
The results showed that the Exponential GPR and Medium SVM models are the most performant in terms of accuracy and robustness. Consequently, these models will be implemented in the subsequent phases of the thesis to predict rotors performance.

Chapter 5

Feature selection

5.1 Introduction

When dealing with high dimensional datasets such as the one analysed in this thesis, the choice of features plays a fundamental role in building ML models that are effective and generalisable. In fact, the presence of redundant or scarcely significant features can not only burden the model, but also compromise its ability to correctly learn the relations present in the data. In this chapter the method used to automatically select the most relevant features is illustrated, according to the predictive performance obtained.
The chosen approach is based on the use of a Genetic Algorithm (GA), an evolutionary optimisation technique that is able to efficiently explore complex search spaces. After a concise discussion of the reasons that led to the adoption of feature selection, there is a detailed description of the phases of the algorithm, the imposed constraints, the fitness function and the configuration options employed. Afterwards the results obtained with different models and databases are presented, and finally the automated implementation of the procedure in the MATLAB environment is illustrated.

5.2 Reasons for Using Feature Selection

In Chapter 3 the format in which the data are stored in the database was presented. The motivation for using feature selection lies in the possibility that the model may yield better results by considering only a limited number of features. So the goal is to verify whether the model can improve its performance without exploiting all the data available in the database, thus increasing its capacity for generalisation. As explained in that chapter, depending on the use of whether the static or dynamic database, and the dimensional or non dimensional version, the data differ, however those concerning geometry are arranged in the same way. Consequently, for each database and for each model, the combination of geometric features that allows the smallest errors will be identified[1].

5.3 Feature Selection Algorithm

As discussed in Chapter 2, several approaches exist for performing feature selection. However, because of the complexity of the present problem, which is characterised by high dimensionality and possible non linear interactions among variables, it was decided to adopt a GA for the feature selection stage. This choice is motivated by the ability of evolutionary algorithms to efficiently explore large and complex search spaces, to avoid becoming trapped in local minima and to offer good solutions even when the objective function is non differentiable or discontinuous. [23, 24]

5.3.1 Genetic Algorithm

GA are a class of optimisation methods inspired by the principles of natural selection and biological evolution [25]. In the context of feature selection, every individual of the population represents a possible combination of variables coded as a binary chromosome: each gene of the chromosome indicates the presence (1) or absence (0) of a specific feature.
The main phases of the GA are the following:

- *Initialisation.* A starting population composed of randomly generated individuals is created.
- *Evaluation (Fitness function).* Each individual is evaluated by means of a fitness function, for instance the predictive performance of a model trained only with the features selected by that chromosome. [15]
- *Selection.* The fittest individuals (those with lower fitness values) are chosen for the next generation.
- *Elitism.* A certain number of the best individuals of the current generation (the elite) are copied unchanged into the next generation, which ensures that the best solutions are not lost.
- *Crossover.* The remaining individuals of the new population (the children) are generated by crossing the genetic information of two parents. Crossover

[1]Actually, as will be explained later, in the case of GPR the dynamic case is not considered.

combines the features of the parents to explore new regions of the solution space.

- *Mutation.* With a certain probability some genes in the children mutate, switching from 0 to 1 or from 1 to 0. This introduces genetic variation and allows the algorithm to avoid local minima.
- *Iteration.* The process of evaluation, selection, crossover and mutation is repeated for a fixed number of generations or until predefined stopping criteria are reached.

The described approach is particularly effective in high dimensional contexts, where exhaustive methods[2] would be too computationally expensive. [26]

5.3.2 MATLAB Implementation of the Genetic Algorithm

Implementing the GA in MATLAB involves a few key steps:

- definition of constraints;
- definition of the fitness function;
- choice of the options.

Constraints

When performing feature selection, constraints were introduced to allow interpolation of the results. Specifically, in order to prevent the feature selection procedure from choosing too few features to interpolate the data along the blade span[3], the presence of features was forced at the following stations, both for normalised chord and β:

15%, 20%, 30%, 45%, 60%, 70%, 75%, 80%, 90%, 100%

As a further limitation, the GA is compelled to choose both the normalised chord and β for a given station. It means that, if one elects to select the features at 55 % span, both normalised chord and β for that station must necessarily be selected. By proceeding in this way, the code is slightly simplified and, in principle, no excessive worsening of the results is expected. After introducing these limitations, the number of possible combinations is markedly reduced, going from 2^{36} possibilities to 2^{8}.

[2]Exhaustive methods test every possible feature combination.

[3]In a later chapter, these features will be used not only to predict CT and CP, but also to generate rotors that satisfy certain characteristics.

The choice of features to be locked was made on the basis of rotor physics and geometry. In fact, for a generic rotor, the largest variations of incidence angle and chord occur near the root of the blade, whereas the most important region for lift generation lies around 75 % of the span. For this reason those areas of the rotor are densified.
It would have been possible to make other choices, which probably would have led to different results from those reported later[4].

Fitness function

The fitness function is the function containing the metrics that enable the GA to evaluate individuals. In the present case individuals are assessed according to how the ML model predicts CT and CP for the rotors in the database when only some features are considered. To do that, group cross-validation is employed, which is a variant of cross-validation used when the data are divided into groups and when it is requested that data belonging to the same group do not end up in both the training and the validation sets. In this case the database contains many rows that differ only by a few values, since the same rotor appears under different operating conditions. If standard cross-validation were applied, the training and validation data would contain the same rotors, and this would give results that are too optimistic. As stated in Chapter 3, 162 different rotors are present in the database and they have to be divided into groups whose number can be estimated with Sturges' rule[5]:

$$k = 1 + \log_2(n) \tag{5.1}$$

Applied to the present case, the rule gives $k = 8$. Since using such a large number of groups would require too much time to train a single model when GPR is considered, applying it within the GA would become unsustainable[6], hence $k = 3$ was set manually[7].
Therefore the code operates as follows:

1. it selects the database columns chosen by the GA;
2. it trains the ML model with group cross-validation for both CT and CP;

[4]The feature selection code was designed to be easily modified, so that if it is required to change the constraints described above, it can be done very simply.

[5]Sturges' rule is an empirical guideline for determining the optimal number of classes (or intervals) when building a histogram.

[6]Considering an ordinary laptop. If sufficient computational resources are available, it can be used $k = 8$.

[7]The code nevertheless allows Sturges' rule to be used.

3. it calculates the RMSE (Equation (2.1)) for both CT and CP;
4. it sums the two RMSEs after normalising them by the average value of CT and CP in the database output:
$$err_{tot} = \frac{err_{CT}}{mean(output_{CT})} + \frac{err_{CP}}{mean(output_{CP})} \tag{5.2}$$

The best individuals are those with smallest errors.

Options

The GA allows a wide number of options to be specified so that it can be adapted to various situations. The options used are listed below, along with the reasons why they were chosen [27]:

- **CreationFcn = `@gacreationuniformint`**. Function that creates the initial population.

  MATLAB uses this function by default whenever the optimisation problem includes integer constraints. The routine imposes an artificial bound on unbounded components, generates individuals uniformly at random within the specified limits, and then enforces the integer constraints.

- **CrossoverFcn = `@crossoverlaplace`**. Function used by the algorithm to create crossover children.

  This function is also used by MATLAB by default when the problem contains integer constraints. The operator produces children by combining the parents through a Laplace distribution while preserving the integer nature of the solutions. One of the two following formulas is chosen randomly:

  $$xOverKid = p_1 + bl \cdot abs(p_1 - p_2)$$
  $$xOverKid = p_2 + bl \cdot abs(p_1 - p_2)$$

  where:

  – p_1 and p_2 are the two parents;
  – bl is a random number drawn from a Laplace distribution;
  – $xOverKid$ is the newly generated child.

- **SelectionFcn = `@selectiontournament`**. Function that selects the parents for crossover and mutation children.

  Tournament selection chooses the parents by randomly extracting a group of individuals and then taking the best among them. In this work the default tournament size of 4 is used.

- **MutationFcn = `@mutationpower`**. Function that produces mutation children.

  MATLAB applies this power–mutation operator by default when integer constraints are present. For each component $x(i)$ of the parent, the i-th component of the child is computed as

  $$mutationChild(i) = x(i) - s \cdot (x(i) - lb(i)) \quad if \;\; t < r$$

  $$mutationChild(i) = x(i) + s \cdot (ub(i) - x(i)) \quad if \;\; t \geq r$$

  where:

  - t is the scaled distance of $x(i)$ from the lower bound $lb(i)$;
  - s is a random variable drawn from a power distribution;
  - r is a random variable drawn from a uniform distribution.

  The function also handles the case $lb(i) = ub(i)$, in which the child's i-th component is simply set equal to that common bound.

- **PopulationSize = 20**. Size of the population.

  Because the total number of admissible combinations is small in the present case, a low population size is sufficient. Should the constraints be relaxed, the number of individuals per generation can be increased.

- **EliteCount = 1**. Number of individuals in the current generation that are guaranteed to survive to the next one (the elite).

  A value of one is chosen so that only the best individual is preserved intact, whereas all the others are used to generate offspring, ensuring genetic diversity.

- **Generations = 400**. Maximum number of iterations before the algorithm stops.

  A rather high limit is set to make sure the algorithm has enough time to reach a good solution. In practice this limit is never reached[8].

- **Display = `'diagnose'`**. Level of output displayed. The `'diagnose'` level is chosen to obtain an overall view of what the GA is doing.

[8]With only 2^8 possible combinations and a population of 20 individuals, just 14 generations are theoretically required. More generations appear in the plots because the default GA tolerance was not tightened.

- **PlotFcns = `@gaplotbestf`**. Function that plots data produced by the algorithm.

  This routine displays both the best fitness and the mean fitness for each generation.

- **OutputFcns**. Function called by the GA at every iteration.

  A custom MATLAB function is implemented so that, once the GA has finished, a graph shows how the number of features selected for the best individual evolves from one generation to the next. By examining that graph it can be observed the variation in the feature count of the best solution at each iteration.

- **UseParallel = true**. This option allows the computation to use the computer's CPU cores in parallel, speeding up the calculations.

- **InitialPopulationMatrix**. The possibility of specifying an initial population has been added. If a certain feature combination is thought to be the best, the GA can be started from that solution[9]. If no initial population is supplied, the algorithm creates it randomly via `@gacreationuniformint`, as mentioned earlier.

The GA configuration and execution script is reported below.

Listing 5.1: GA configuration and execution

```
%% === CONFIGURAZIONE GA ===
n_stazioni = 18;
num_features = n_stazioni;
lb = zeros(1, num_features);
ub = ones(1, num_features);
IntCon = 1:num_features;

% Forza alcune stazioni se vuoi
lb(stazioni_geometria_da_tenere) = 1;

%% Definizione della funzione di fitness per GA
fitnessFcn = @(subset) fitnessFunction_nuova(subset, dati, output_CT,
    output_CP, c, scelta_modello);

%% Configurazione Algoritmo Genetico
if isempty(Initialpopulation)
    options = optimoptions('ga', ...
```

[9]If the number of individuals in the initial population is smaller than the specified population size, `@gacreationuniformint` is used to generate the remaining individuals.

```
            'PopulationSize', num_population, ...
            'MaxGenerations', num_generations, ...
            'PlotFcn','gaplotbestf', ...
            'Display', 'diagnose', ...
            'OutputFcn', @gaoutfun, ...
            'EliteCount', Elite, ...
            'UseParallel', true);
else
    options = optimoptions('ga', ...
            'PopulationSize', num_population, ...
            'MaxGenerations', num_generations, ...
            'PlotFcn','gaplotbestf', ...
            'Display', 'diagnose', ...
            'OutputFcn', @gaoutfun, ...
            'EliteCount', Elite, ...
            'UseParallel', true, ...
            'InitialPopulationMatrix', Initialpopulation);
end

%% Esecuzione GA per selezione feature
[selected_features, best_fitness] = ga(fitnessFcn, num_features, [],
    [], [], [], lb, ub, [], IntCon, options);
```

The code of the fitness function and the `@gaoutfun` routine is provided in Appendices A.1 and A.2.
The MATLAB workflow is summarised in Figure 5.1.

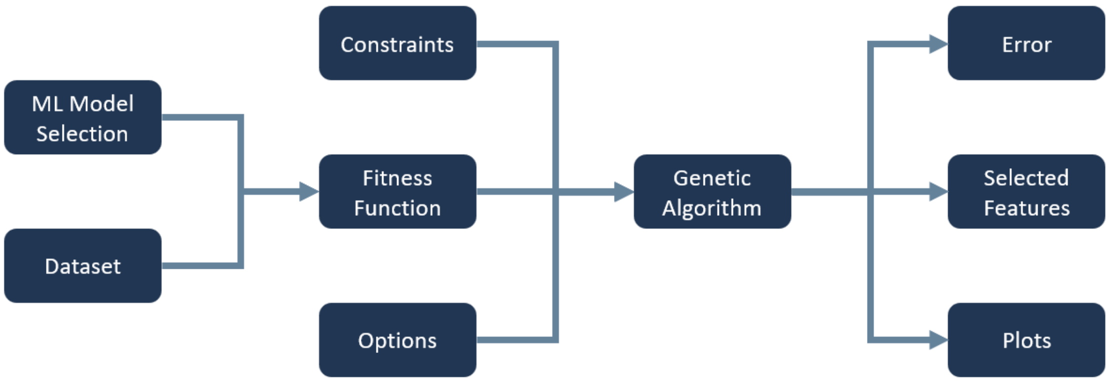


Figure 5.1: MATLAB code flow chart

5.4 Results

For each analysed case the following are shown:

- the plot of the number of features of the best individual versus generations;
- the `@gaplotbestf` plot;
- the selected features;
- the fitness error of the best solution computed with Equation (5.2).

5.4.1 GPR

As mentioned earlier, for GPR the analyses are performed only with the static database, because the model is very heavy to train and its computational cost grows with $O(n^3)$[10]. So the feature subset found for the static database is used also for the dynamic database.

Dimensional database

Stations:

15%, 20%, 30%, 45%, 55%, 60%, 65%, 70%, 75%, 80%, 90%, 95%, 100%

Fitness error: 0.3350.

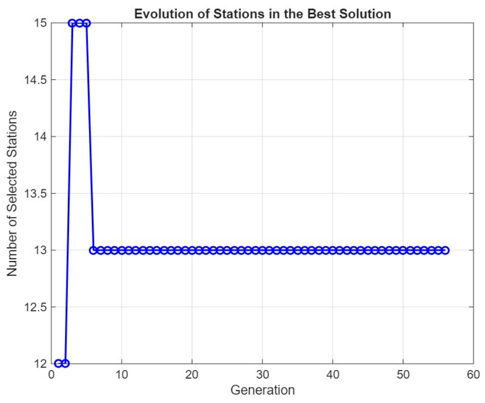


Figure 5.2: Number of features — dimensional GPR

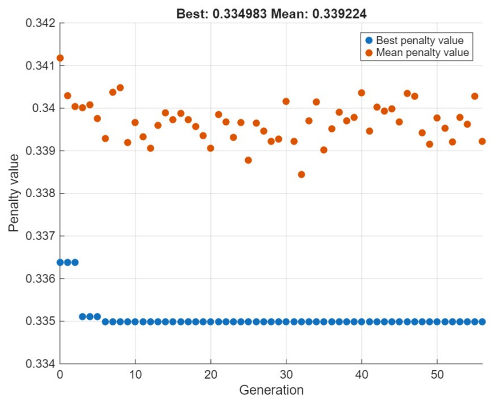


Figure 5.3: `@gaplotbestf` — dimensional GPR

[10]Using a laptop the feature selection with the static database ($n = 2611$) and a population of 20 individuals takes several hours. With the dynamic database ($n = 18910$) the execution time becomes unsustainable.

Non dimensional database

Stations:

15%, 20%, 30%, 45%, 55%, 60%, 65%, 70%, 75%, 80%, 90%, 95%, 100%

Fitness error: 0.3328.

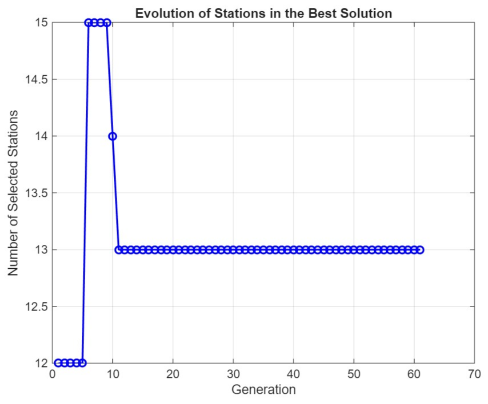


Figure 5.4: Number of features — non dimensional GPR

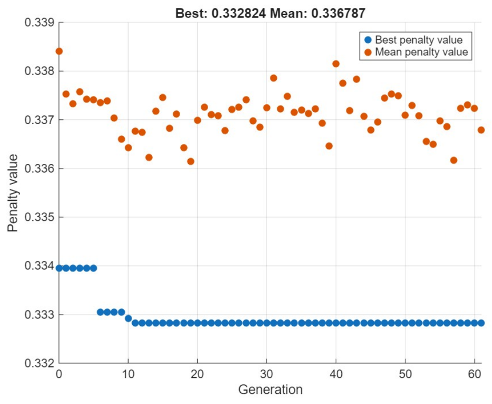


Figure 5.5: `@gaplotbestf` — non dimensional GPR

Both databases converge to the same combination of features. A similar behaviour will be observed when comparing dimensional and non dimensional SVM.

5.4.2 SVM

Static database

- **Dimensional database**

  Stations:

  15%, 20%, 30%, 45%, 60%, 70%, 75%, 80%, 90%, 100%

  Fitness error: 0.4443.

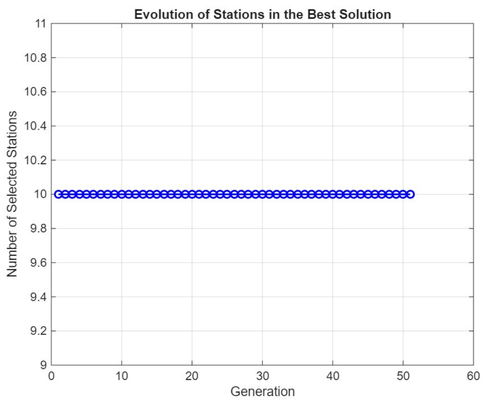


Figure 5.6: Number of features — static dimensional SVM

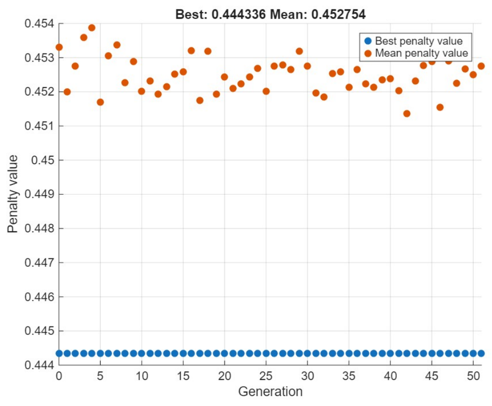


Figure 5.7: `@gaplotbestf` — static dimensional SVM

- **Non dimensional database**

  Stations:

  15%, 20%, 30%, 45%, 60%, 70%, 75%, 80%, 90%, 100%

  Fitness error: 0.4345.

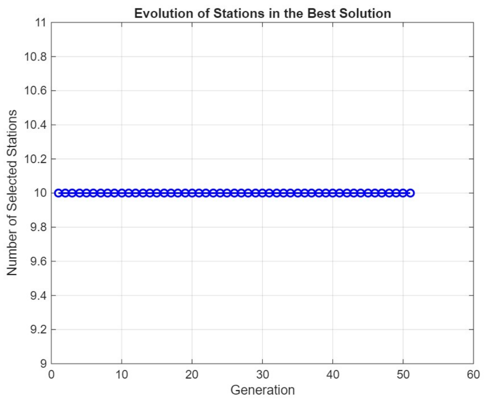


Figure 5.8: Number of features — static non dimensional SVM

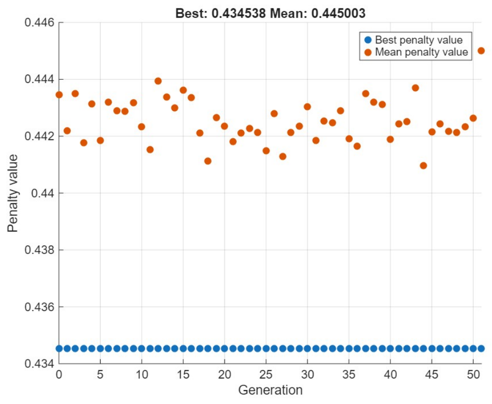


Figure 5.9: `@gaplotbestf` — static non dimensional SVM

Dynamic database

- **Dimensional database**

  Stations:

  15%, 20%, 30%, 45%, 60%, 70%, 75%, 80%, 90%, 95%, 100%

  Fitness error: 0.7659.

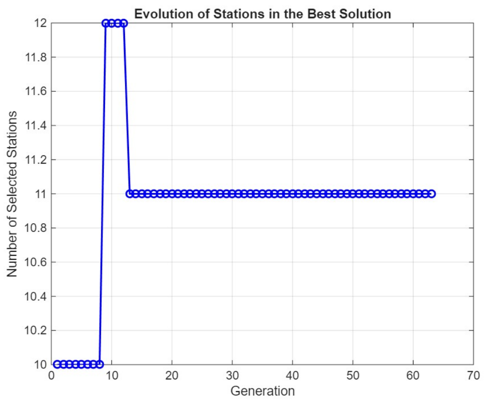


Figure 5.10: Number of features — dynamic dimensional SVM

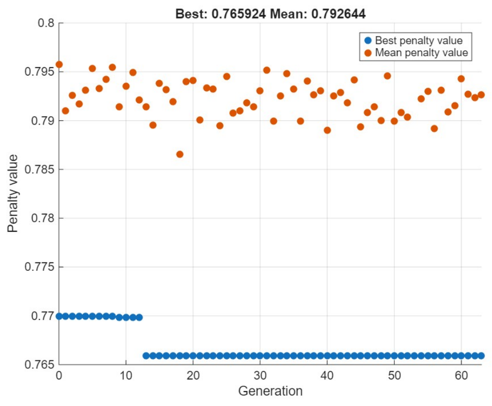


Figure 5.11: `@gaplotbestf` — dynamic dimensional SVM

- **Non dimensional database**

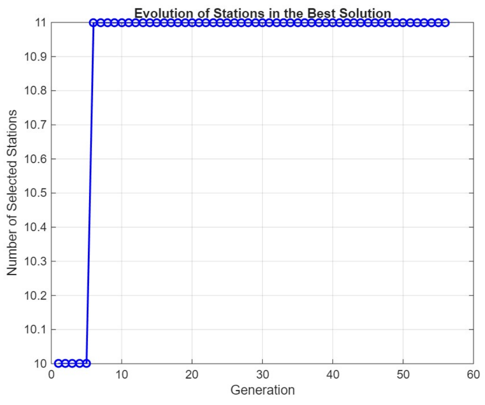


Figure 5.12: Number of features — dynamic non dimensional SVM

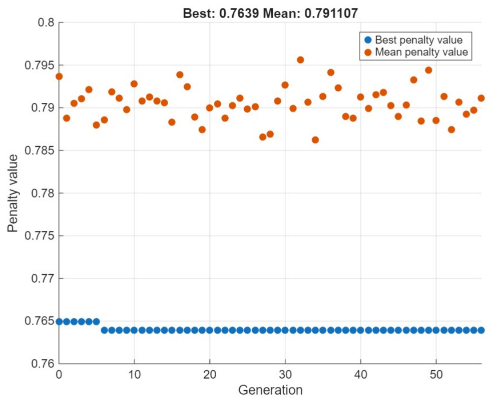


Figure 5.13: `@gaplotbestf` — dynamic non dimensional SVM

Stations:

15%, 20%, 30%, 45%, 60%, 70%, 75%, 80%, 90%, 95%, 100%

Fitness error: 0.7639.

The results obtained are summarized in Table 5.1.

Model	**Number of stations**	**Fitness error**
Dimensional GPR	13	0.3350
Non dimensional GPR	13	0.3328
Static dimensional SVM	10	0.4443
Static non dimensional SVM	10	0.4345
Dynamic dimensional SVM	11	0.7659
Dynamic non dimensional SVM	11	0.7639

Table 5.1: Comparison of ML models

As anticipated, dimensional and non dimensional cases yield identical feature subsets, whereas static and dynamic cases differ: dynamic databases converge on subsets with more stations[11].
When the GPR and the SVM are compared under identical conditions (static database), it can be seen that the former employs a larger number of features. In addition, the SVM exhibits greater errors than the GPR, and they increase when the dynamic database is used. This behaviour stems from the fact that the dynamic database is much larger than the static one. Therefore, with the same number of folds in the grouped cross-validation, the test is carried out on a far greater number of rotors.

5.5 MATLAB Implementation of the Results

Once the optimal feature combinations for each analysed case had been identified, a dedicated MATLAB script was developed to manage feature selection automatically. It contains a structured logic based on `if` statements, enabling the correct combination of features to be selected according to the user's choices.
Specifically, the code takes as input the desired ML model (GPR or SVM) and the type of database to be used (static or dynamic, dimensional or non dimensional). On the basis of this information, the system automatically retrieves the feature

[11]Shown for SVM; GPR could not be tested with the dynamic database.

combination previously selected by the GA for that particular scenario, thereby making the process fully automated. This approach ensures consistency in applying the feature selection results to later stages of the work, such as model training or rotor generation.
The structure of the code also makes it easy to extend the framework in case new models or analysis scenarios will be added in the future.
The MATLAB code that implements the automatic feature selection logic is provided in Appendix A.3.

5.6 Conclusions

This chapter presented the procedure adopted for automatic feature selection, with the goal of improving the predictive capability of the employed ML models. The chosen technique was the genetic algorithm (GA), selected despite the reduced number of possible combinations imposed by the constraints. Although an exhaustive search would have been theoretically feasible in this case, the GA was adopted to develop a generalisable and flexible approach, applicable even to more complex scenarios where the combination space is too large for a brute force search.
Using the GA makes it possible to explore the space of candidate configurations efficiently, to avoid the sub-optimal solutions typical of simpler approaches, and to keep the method robust when the objective function is non linear, discontinuous or non differentiable.
From the comparison between models it emerges that the GPR, although it requires a larger number of features, achieves the lowest fitness error. By contrast, the SVM model selects fewer features at the cost of higher error.
Moving from the static to the dynamic database, the number of features selected for the SVM increases. For lack of time, the same verification was not performed for the GPR; consequently, the feature set adopted for this model may not be optimal and the reported performance should be regarded as conservative.
Another noteworthy observation concerns the comparison between dimensional and non dimensional databases: overall the results are very similar in the two configurations; however, the non dimensional database yields slightly lower fitness errors, suggesting that removing dimensional information can enhance model generalisation.
Finally, the results have been integrated into MATLAB by means of a script that automatically selects the most appropriate features according to the user's choices. This structure is easily extendable and makes the entire workflow scalable for future developments, new models, or new datasets.

Chapter 6

Rotors Generation

6.1 Introduction

This chapter includes the description of the functioning of a MATLAB developed code, which allows the generation of rotors based on user inputs, using rotors from the database as a starting point.
The final goal is to create a tool that, given specific operating conditions selected by the user, can rapidly generate theoretically optimal rotors for the required performance specifications. However, considering the possibility that the ML models predict rotor performance inaccurately, it becomes necessary to validate the obtained results through further numerical analyses to verify the performance of the generated rotor.
The code has two main functionalities:

1. construction of the Pareto front;

2. generation of a single rotor.

In the following sections both functionalities are described in depth, along with the various options selectable by the user. Finally, the results obtained are presented and they are validated through BEM and CFD analyses.

6.2 Input

The code is structured to offer users a wide range of configurability through a series of input parameters. The available options allow control of both the objective of the analysis and the methods by which it is carried out. The code used for input management is reported in Appendix B.1.
The selectable options are listed and described below:

- **Type of analysis** (`'pareto'` or `'diretta'`): represents the main choice to be made.

  Selecting `'pareto'`, the code constructs the Pareto front. Selecting `'diretta'`, it generates a single rotor with the desired characteristics. Both modes are detailed in the following paragraphs.

- **Type of database** (`'statico'` or `'dinamico'`): allows selection of the database on which the predictive model is based.

  The `'statico'` database does not consider variations of the advance ratio J, while the `'dinamico'` database includes them.

- **Type of model** (`'dimensionale'` or `'non dimensionale'`): allows choosing models trained with dimensional or nondimensional data.

- **ML algorithm** (`'GPR'` or `'SVM'`): selects the type of predictive model.

  `'GPR'` corresponds to Gaussian Process Regression with Exponential kernel, while `'SVM'` refers to the Medium-type Support Vector Machine model.

- **Visualization of graphs** (`'si'` or `'no'`): enables or disables the visualization of graphs showing trends of CT, CP, and efficiency η (Formula (3.4)), as a function of advance ratio J[1].

  This option is only available with the `'dinamico'` database since the `'statico'` database does not provide data varying with J.

- **Local optimization** (`'si'` or `'no'`): allows application of a local optimization phase to refine the solution found by the GA.

  In `'diretta'` mode, it is possible to choose between the algorithms `@fmincon` and `@patternsearch`; in `'pareto'` mode, `@fgoalattain` is used instead. Note that activating optimization significantly increases execution time[2].

- **Maximum Mach number at the tip**: specifies the maximum allowed value for the Mach number at the rotor tip. It prevents the code from generating unrealistic solutions that would require excessively high rotational speeds to achieve certain performance levels.

- **Performance targets** (CT, CP): desired thrust and power coefficient targets can be defined.

[1]The range of variation of J can be freely defined.

[2]In the `'pareto'` mode, optimization with many points on the front may make the analysis computationally too heavy.

In `'pareto'` mode, at most one target must be specified, while in `'diretta'` mode both of them can be set.

- **Geometric and operational targets**: optionally, users can define geometric parameters (rotor radius) and operating conditions (angular velocity in rad/s and advance ratio J) to be respected during rotor generation.
- **Tolerances**: indicate the allowed precision relative to the defined targets.

  In `'diretta'` mode, CT and CP tolerances are expressed as percentages, whereas in `'pareto'` mode, they are absolute values.
- **Genetic Algorithm (GA) parameters**: the user must specify population size and maximum generation number.
- **NACA profile selection**: allows selection of the specific four digit NACA airfoil profile used for rotor construction[3].
- **CT and CP weight** (`peso_CT_CP`): a parameter active only in the `'diretta'` mode, designed to compel the GA to seek improved solutions[4].

6.3 Analysis: `'pareto'`

In this paragraph the detailed operation of the code dedicated to constructing the Pareto front is explained.

6.3.1 Pareto front

The Pareto front is the set of optimal solutions in a multi-objective optimization problem. It comprises all non dominated points; that is, those solutions for which there exists no alternative that is simultaneously better in all objectives considered. A solution can belong to the Pareto front even if it does not dominate others, provided that it is not dominated by any. [28]

6.3.2 Implementation in MATLAB

The construction of the Pareto front is based on using the multi-objective GA via the MATLAB function `@gamultiobj`. The operation of this function is similar to the GA explained in Chapter 5. Therefore, it is necessary to define a fitness function and a constraint function.

[3]Additional details on its usage are provided in the following sections.

[4]Increasing its numerical value forces the GA to move closer to the Pareto front.

Fitness function

The fitness function used is reported in Appendix B.2. Its task is to evaluate the rotors' performance by computing CT and CP. In addition to estimating these two parameters, infinite penalties are assigned to rotors that have tip Mach numbers exceeding the imposed constraint and also to rotors having negative CP[5].

Since the GA might generate unrealistic geometries (especially with small populations), a constraint is imposed on variations between successive stations along the blade span, based on an analysis of the rotors in the database. Therefore, a function that analyses all rotors in the database was created.

Constraint function

The constraint function analyses the database rotors to determine the admissible variations between one station of the rotor and the next one. In particular, it examines how the normalized chord and the β vary for all rotors in the database and extracts an interval of variation. This process is carried out for each interval between two rotor stations. Once the maximum and the minimum variation values for each station are extracted, they are implemented in the multi-objective GA as non linear inequality constraints. In this way, the GA is allowed to generate rotors closer to real ones even using few individuals per generation. The two functions used to perform this procedure are reported in Appendix B.3.

Options

As done for feature selection in Chapter 5, it is possible to specify the operating options for the multi-objective GA. All selected options are reported below and some of them are explained[6] [27]:

- **CreationFcn = `@gacreationuniformint`**.
- **CrossoverFcn = `@crossoverintermediate`**.

  Used by default in the presence of linear constraints, it generates offspring by computing a weighted average between the two parents.

  The child is generated from the two parents *parent1* and *parent2* according to the following formula:

$$child = parent1 + rand \cdot Ratio \cdot (parent2 - parent1)$$

[5]This second option was added to prevent the Pareto front from developing in regions of no practical interest.

[6]Many of them were already introduced in Chapter 5.

where *Ratio* represents a weight.

- **SelectionFcn = `@selectiontournament`**.
- **MutationFcn = `@mutationadaptfeasible`**.

  Default option in the presence of non integer constraints, it randomly generates directions that adapt based on the success or the failure of the previous generation.
- **Display = 'diagnose'**.
- **PlotFcn = `@gaplotpareto`**.

  Plots the Pareto front for the objective functions.
- **UseParallel = false**.

  Parallelization is not used because various tests show that its use, in this specific case, slows down code execution[7].
- **MaxGenerations**.

  The maximum number of generations that the multi-objective GA can perform is set by the user.
- **PopulationSize**.

  The population size is also set by the user, bearing in mind that larger populations can yield potentially better results but require more execution time.
- **InitialPopulationMatrix**.

  As the initial population, the data present in the database are used. If the number of chosen individuals is smaller than the database size, only a part of it is used; if it is larger, additional individuals are generated randomly using the function `@gacreationuniformint`.
- **HybridFcn**.

  As previously mentioned, it is possible to use the function `@fgoalattain` to optimize the Pareto front. This is the only function implemented natively in MATLAB for the multi-objective GA.

[7]This effect occurs because the function evaluation is very fast and the overhead of starting, synchronizing and copying data between workers outweighs any savings from parallel execution.

Below there is the code used to configure and run the multi-objective GA with all implemented options.

Listing 6.1: Script for configuring and running the multi-objective GA

```matlab
% Definisco la popolazione iniziale dal database
initialPopulation = dati;

% Definizione della funzione di fitness per GA
fitnessFcn = @(x) stima_CT_CP(x, Modello_ML_CT, Modello_ML_CP,
    densita_aria_SL, viscosita_dinamica_aria_SL, velocita_del_suono_SL
    , mach_massimo_al_tip, scelta_dim_nondim, colonna_075);

% Definisco la funzione vincolo con parametri. Questa funzione serve
    a limitare il GA per non fargli generare geometrie troppo strane
nonlcon_fun = @(x) nonlcon_funzione_massimi(x, massimi_corda,
    massimi_beta);

if strcmp(ottimizzazione_locale_GA_multiobiettivo, 'si')
    opts_fgoalattain = optimoptions('fgoalattain', 'Display', 'iter')
    ;
    options = optimoptions('gamultiobj', ...
        'Display','diagnose', ...
        'PlotFcn','gaplotpareto', ...
        'UseParallel',false, ...
        'MaxGenerations',numero_generazioni, ...
        'PopulationSize',grandezza_popolazione, ...
        'InitialPopulationMatrix',initialPopulation, ...
        'HybridFcn', {@fgoalattain, opts_fgoalattain});
elseif strcmp(ottimizzazione_locale_GA_multiobiettivo, 'no')
    options = optimoptions('gamultiobj', ...
        'Display','diagnose', ...
        'PlotFcn','gaplotpareto', ...
        'UseParallel',false, ...
        'MaxGenerations',numero_generazioni, ...
        'PopulationSize',grandezza_popolazione, ...
        'InitialPopulationMatrix',initialPopulation);
end

% Faccio partire il GA multiobiettivo
[x_sol_pareto, fval] = gamultiobj(fitnessFcn, nVars, [], [], [], [],
    lb, ub, nonlcon_fun, options);
```

The lower (lb) and upper (ub) bounds are set to the minimum and maximum values present in the database for each column.
After the Pareto front has been constructed, considering the various inputs and constraints used, if a target on CT or CP has been imposed, it is possible to determine which rotor on the front best satisfies the target. The rotor's characteristics, together with the chosen NACA profile, are then automatically exported

to an Excel file containing a macro that generates the rotor in SolidWorks. In the same Pareto front plot, the database rotors that satisfy the imposed targets and tolerances are also shown. Specifically, the database rotor that meets the constraints and that is closest to the user's requested CT or CP is automatically selected[8]. The two rotors, database and Pareto front, are displayed on screen via a 3D plot to give an idea of their shapes[9]. Finally, if graph display has been enabled, the variation of CT, CP and η as functions of J is also shown.
Below there is an example of results generated by the dimensional dynamic GPR ML model.

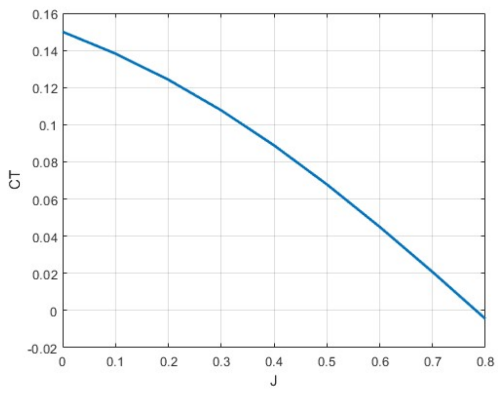


Figure 6.1: Trend of CT

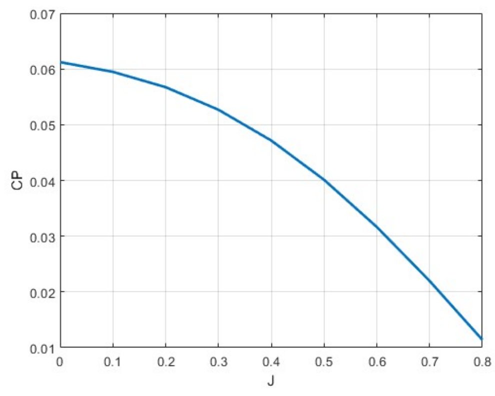


Figure 6.2: Trend of CP

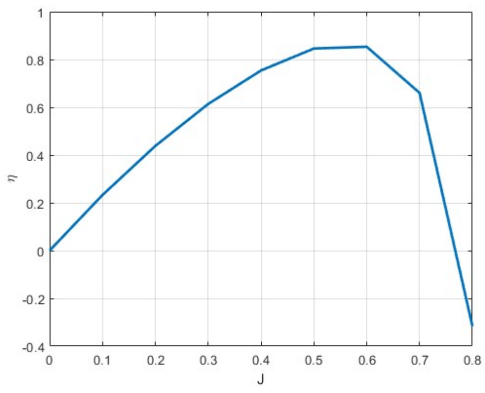


Figure 6.3: Trend of η

[8]Rotors closer to the Pareto front are favoured.

[9]The shape shown may differ slightly from that exported to SolidWorks due to different interpolation applied.

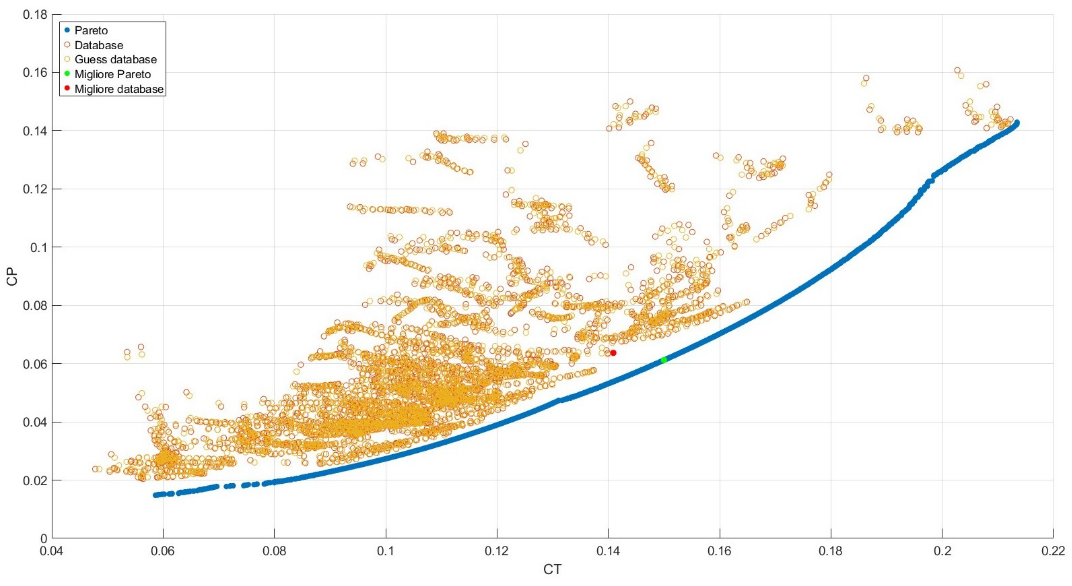


Figure 6.4: Pareto Front

6.4 Analysis: 'diretta'

This paragraph includes the explanation of the operation of the code dedicated to generating new rotors from user defined inputs.

6.4.1 Implementation in MATLAB

The code uses the GA to generate new rotors and for this reason it is necessary to define a fitness function.

Fitness function

The fitness function used is reported in Appendix B.4. The function evaluates how closely a rotor approaches the imposed CT and CP targets as follows:

- if a CT target is imposed, the relative error is calculated as:

$$valore = valore + \frac{|CT\_new\_rotor - target\_CT|}{target\_CT} \tag{6.1}$$

- if no CT target is imposed:

$$valore = valore - \frac{CT\_new\_rotor}{mean(output\_CT)} \cdot peso \tag{6.2}$$

- if a CP target is imposed:

$$valore = valore + \frac{|CP\_new\_rotor - target\_CP|}{target\_CP} \tag{6.3}$$

- if no CP target is imposed:

$$valore = valore + \frac{CP\_new\_rotor}{mean(output\_CP)} \cdot peso \tag{6.4}$$

Rotors with lower *valore* are better. Again, when the tip Mach number exceeds the maximum value, $valore = Inf$ is set, thus preventing the algorithm from returning a rotor that does not respect this constraint. *peso*, as previously explained, acts to push the solution toward the Pareto front by increasing *valore* and forcing the GA to seek better solutions before converging due to the tolerances.

Constraint function

The constraint function used, reported in Appendix B.5, is more complex than in the previous case. The reason is that, besides implementing the same constraints explained previously, it must also consider the CT and CP targets. Indeed, when at least one of the two targets is imposed, it must ensure that the generated rotor's value falls within the imposed tolerance:

- if target_CT imposed:

$$\frac{|CT\_new\_rotor - target\_CT|}{target\_CT} \leq tolerance\_CT \tag{6.5}$$

- if target_CP imposed:

$$\frac{|CP\_new\_rotor - target\_CP|}{target\_CP} \leq tolerance\_CP \tag{6.6}$$

Options

The selected options are the following ones (only those different from `'pareto'` are reported) [27]:

- **CrossoverFcn = `@crossoverscattered`**.

  Generates a random binary vector used to combine the genes of the two parents.

  For each position of the vector:

  - if the value is 1, the gene is taken from the first parent;
  - if the value is 0, the gene is taken from the second parent.

  The resulting child is thus a random combination of the parents' genes.

- **SelectionFcn** = **`@selectionstochunif`**.

  Arranges a line where each parent corresponds to a section of line proportional to its scaled fitness value.

  The algorithm moves along the line in steps of equal size. At each step, it selects the parent corresponding to the section on which it lands. The first step is a random uniform number less than the step size.

No PlotFcn is present and for HybridFcn it is possible to choose between `@fmincon` and `@patternsearch`.

Listing 6.2: Script for configuring and running the GA

```
% FUNZIONE OBIETTIVO
inverse_design = @(x) punteggio(x, target_CT, Modello_ML_CT, peso_CT,
     target_CP, Modello_ML_CP, peso_CP, media_output, densita_aria_SL,
     viscosita_dinamica_aria_SL, mach_massimo_al_tip,
    velocita_del_suono_SL, scelta_dim_nondim, colonna_075);
% Metto vincoli
nonlcon = @(x) vincoli(x, target_CT, target_CP, Modello_ML_CT,
    Modello_ML_CP, tolleranza_CT, tolleranza_CP, densita_aria_SL,
    viscosita_dinamica_aria_SL, velocita_del_suono_SL,
    scelta_dim_nondim, colonna_075, massimi_corda, massimi_beta);

% Fase Globale (GA): Popolazione iniziale basata sul database
% fprintf('--- FASE GLOBALE (GA) ---\n');

popolazione_iniziale = dati;

if strcmp(ottimizzazione_locale_GA, 'si')
    if strcmp(ottimizzazione_locale_GA_funzione, 'fmincon')
        funzione = @fmincon;
        opzioni_funzione = optimoptions('fmincon', 'Display', 'iter')
    ;
    elseif strcmp(ottimizzazione_locale_GA_funzione, 'patternsearch')
        funzione = @patternsearch;
        opzioni_funzione = optimoptions('patternsearch', 'Display', '
    diagnose', 'MeshTolerance', 1e-8, 'UseParallel', false);
    end
    options_ga = optimoptions('ga',...
        'Display', 'diagnose',...
        'PopulationSize', grandezza_popolazione,...
        'MaxGenerations', numero_generazioni,...
        'InitialPopulationMatrix', popolazione_iniziale, ...
        'EliteCount', Elite, ...
        'UseParallel', false, ...
        'HybridFcn', {funzione, opzioni_funzione});

elseif strcmp(ottimizzazione_locale_GA, 'no')
    options_ga = optimoptions('ga',...
        'Display', 'diagnose',...
        'PopulationSize', grandezza_popolazione,...
        'MaxGenerations', numero_generazioni,...
        'InitialPopulationMatrix', popolazione_iniziale, ...
        'EliteCount', Elite, ...
        'UseParallel', false);
end

% Faccio partire il GA
[sol_final, ~] = ga(inverse_design, nVars, [], [], [], [], lb, ub,
   nonlcon, options_ga);
```

Once again, as soon as the best rotor has been selected considering the various inputs and constraints, its characteristics and the chosen NACA profile are automatically exported to an Excel file containing a macro to generate the rotor in SolidWorks. The rotor is displayed on screen via a 3D plot to visualize its shape, and, if graph display has been activated, CT, CP and η versus J are plotted.

6.5 Blade Element Momentum

Within the code there is a function that uses BEM to estimate the effect of the airfoil profile on the rotor's performance.
The methodology by which the BEM was employed will not be reported here. For a detailed analysis of the method used, refer to [29].
The function runs simulations with NACA 0012 and NACA 4412 profiles, calculating CT and CP in both cases.
Measuring the delta of CT and CP and locating the value predicted by the algorithm within the interval, it is possible to determine the effect of the airfoil profile. Since most rotors in the database are APC and have a NACA 4412 profile for most of the span, it is assumed that the model's predictions anticipate the use of this profile.
The output shows both the value predicted by the model and the delta due to the possibility of choosing different profiles.

```
CT = 0.0926 - Maximum value 0.0926 - Minimum value 0.0806
CP = 0.0444 - Maximum value 0.0444 - Minimum value 0.0388
```

The operation of the MATLAB code can be schematized as in Figure 6.5.

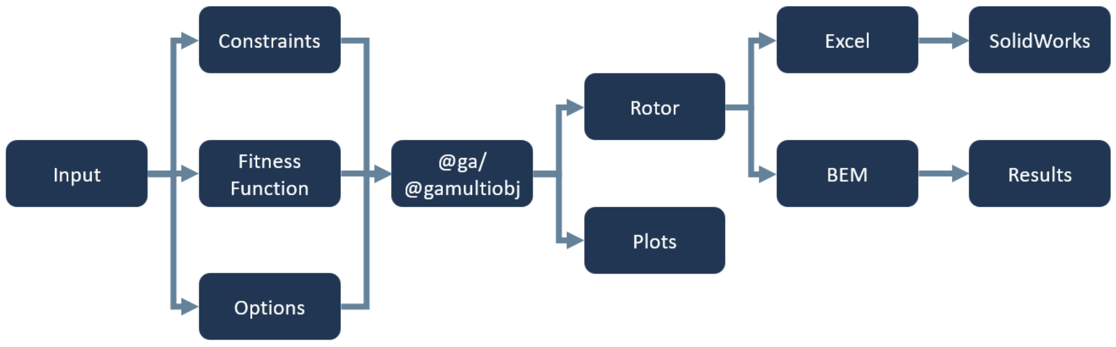


Figure 6.5: Flow chart of MATLAB Pareto code

6.6 Computational Fluid Dynamics

Similarly to the BEM analyses, the detailed description of the CFD methodology adopted will not be explored in depth in this thesis. For a comprehensive treatment of this methodological aspect, see [30].

6.7 Results

The results shown concern only the `'pareto'` analysis. A target CT of 0.15 and a $J = 0$ were set in order to use both the static and dynamic models. The other targets were not set, so the models are free to modify their values within the constraints.
Firstly the rotor geometries and operating conditions are reported and then there are CT and CP values compared with those obtained via BEM and CFD[10].

[10]The CFD analyses were performed only for some ML models.

6.7.1 GPR

Static

- **Dimensional**

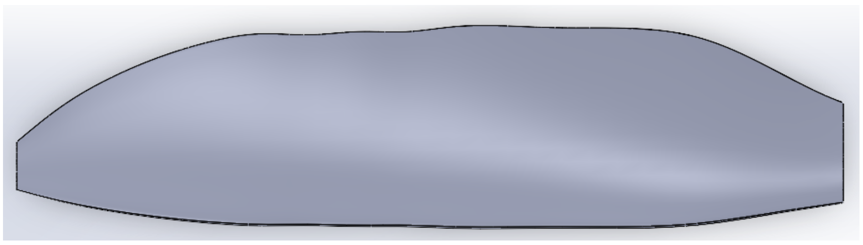


Figure 6.6: Rotor geometry - static dimensional GPR – top view

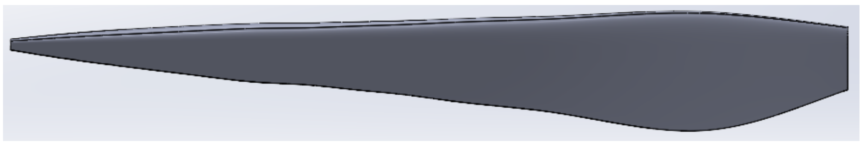


Figure 6.7: Rotor geometry - static dimensional GPR - front view

r/R	c/R	β [°]
0.1500	0.1200	30.9710
0.2000	0.1582	32.8080
0.3000	0.2252	31.3370
0.4500	0.2253	24.1470
0.5500	0.2208	20.5240
0.6000	0.2112	18.9050
0.6500	0.2093	17.1910
0.7000	0.2033	15.7370
0.7500	0.2037	14.6860
0.8000	0.1919	13.3630
0.9000	0.1419	11.3540
0.9500	0.1032	10.5440
1.0000	0.0389	9.3646

Table 6.1: Rotor geometry - static dimensional GPR

Radius = $0.2259m$.

Angular velocity = $450.21\frac{rad}{s}$.

- **Non dimensional**

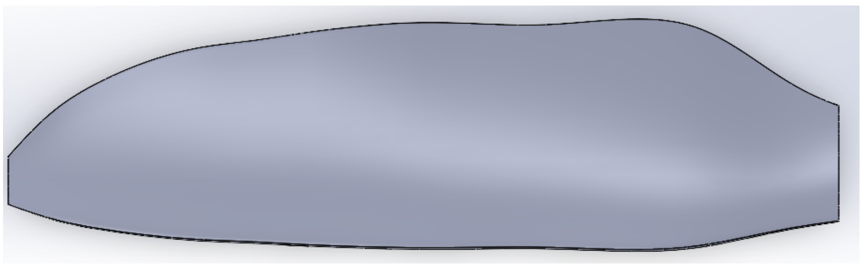


Figure 6.8: Rotor geometry - static non dimensional GPR - top view

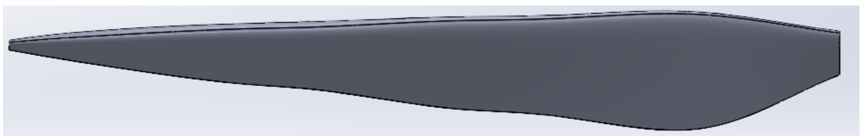


Figure 6.9: Rotor geometry - static non dimensional GPR - front view

r/R	**c/R**	$\boldsymbol{\beta}$ [°]
0.1500	0.1256	19.0880
0.2000	0.1657	25.3140
0.3000	0.2546	26.9340
0.4500	0.2481	22.6730
0.5500	0.2471	20.4600
0.6000	0.2434	18.5680
0.6500	0.2353	16.7020
0.7000	0.2251	15.2760
0.7500	0.2142	14.7570
0.8000	0.2043	13.5450
0.9000	0.1579	10.6450
0.9500	0.1168	9.2730
1.0000	0.0395	8.1564

Table 6.2: Rotor geometry - static non dimensional GPR

Radius = $0.1721m$.

Angular velocity = $355.01\frac{rad}{s}$.

Dynamic

- **Dimensional**

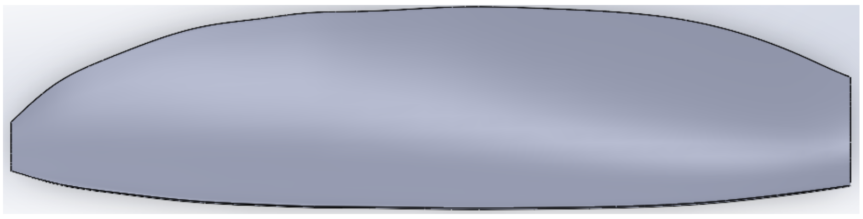


Figure 6.10: Rotor geometry - dynamic dimensional GPR - top view

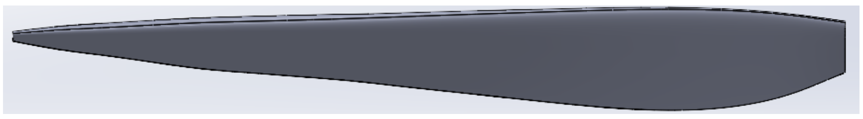


Figure 6.11: Rotor geometry - dynamic dimensional GPR - front view

r/R	**c/R**	$\boldsymbol{\beta}$ [°]
0.1500	0.1207	24.2320
0.2000	0.1582	28.6790
0.3000	0.2076	28.6540
0.4500	0.2208	23.9880
0.5500	0.2177	20.3400
0.6000	0.2124	18.8610
0.6500	0.2074	17.1660
0.7000	0.2040	15.8190
0.7500	0.1948	15.0400
0.8000	0.1850	14.1550
0.9000	0.1391	11.3490
0.9500	0.1069	10.1870
1.0000	0.0413	7.9292

Table 6.3: Rotor geometry - dynamic dimensional GPR

Radius $= 0.1730m$.

Angular velocity $= 589.66\frac{rad}{s}$.

- **Non dimensional**

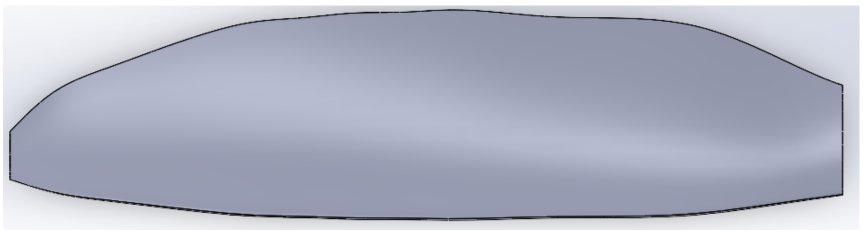


Figure 6.12: Rotor geometry - dynamic non dimensional GPR - top view

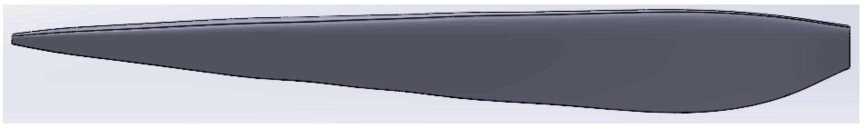


Figure 6.13: Rotor geometry - dynamic non dimensional GPR - front view

r/R	**c/R**	$\boldsymbol{\beta}$ [°]
0.1500	0.1187	19.3010
0.2000	0.1560	26.5140
0.3000	0.2161	27.1770
0.4500	0.2236	23.4040
0.5500	0.2284	20.3300
0.6000	0.2229	18.6750
0.6500	0.2199	17.2890
0.7000	0.2118	15.8380
0.7500	0.2051	15.0180
0.8000	0.1881	13.9650
0.9000	0.1388	11.6150
0.9500	0.1091	10.2170
1.0000	0.0407	8.1197

Table 6.4: Rotor geometry - dynamic non dimensional GPR

Radius = $0.1439m$.

Angular velocity = $471.75\frac{rad}{s}$.

6.7.2 SVM

Static

- **Dimensional**

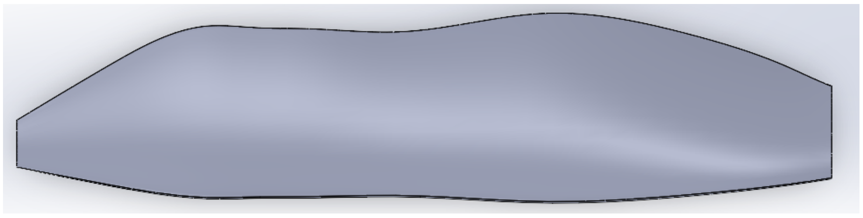


Figure 6.14: Rotor geometry - static dimensional SVM - top view

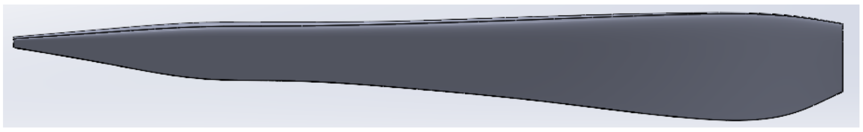


Figure 6.15: Rotor geometry - static dimensional SVM - front view

r/R	**c/R**	$\boldsymbol{\beta}$ [°]
0.1500	0.1183	35.1840
0.2000	0.1587	37.6220
0.3000	0.1990	31.7430
0.4500	0.2141	22.6160
0.6000	0.1833	20.1150
0.7000	0.1846	17.0800
0.7500	0.1854	16.5780
0.8000	0.1873	15.8570
0.9000	0.1322	13.6500
1.0000	0.0269	9.5379

Table 6.5: Rotor geometry - static dimensional SVM

Radius $= 0.2034m$.

Angular velocity $= 501.60\frac{rad}{s}$.

- **Non dimensional**

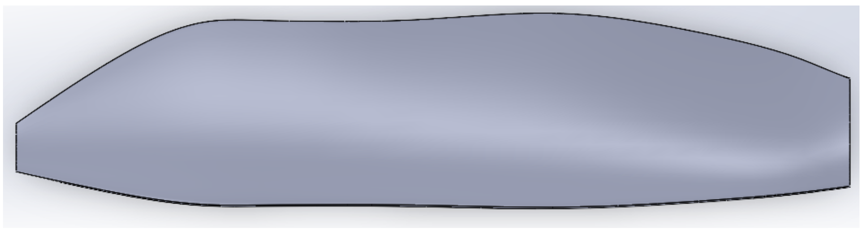


Figure 6.16: Rotor geometry - static non dimensional SVM - top view

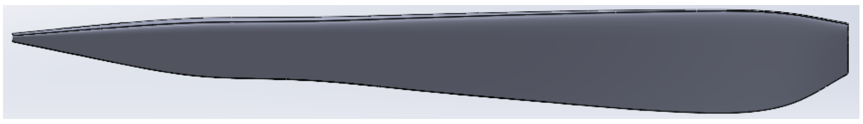


Figure 6.17: Rotor geometry - static non dimensional SVM - front view

r/R	**c/R**	$\boldsymbol{\beta}$ [°]
0.1500	0.1209	23.6580
0.2000	0.1611	31.7950
0.3000	0.1996	30.4320
0.4500	0.2173	23.7090
0.6000	0.2020	20.0830
0.7000	0.1968	16.9050
0.7500	0.1972	16.2120
0.8000	0.1960	15.6560
0.9000	0.1405	13.3060
1.0000	0.0351	6.0647

Table 6.6: Rotor geometry - static non dimensional SVM

Radius $= 0.1392m$.

Angular velocity $= 385.71\frac{rad}{s}$.

Dynamic

- **Dimensional**

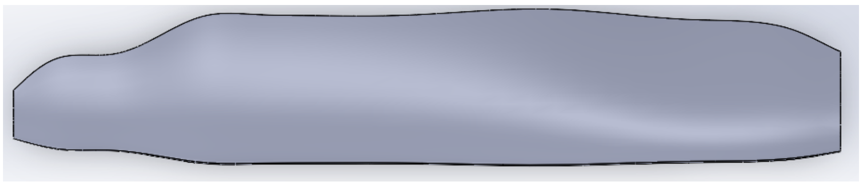


Figure 6.18: Rotor geometry - dynamic dimensional SVM - top view

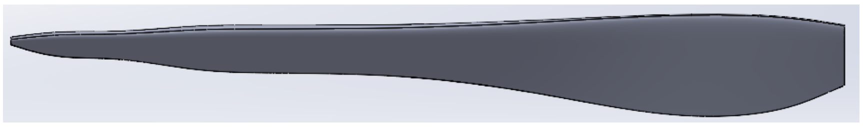


Figure 6.19: Rotor geometry - dynamic dimensional SVM - front view

r/R	**c/R**	$\boldsymbol{\beta}$ [°]
0.1500	0.1188	30.1540
0.2000	0.1586	32.9210
0.3000	0.1779	34.6560
0.4500	0.1769	24.5430
0.6000	0.1597	17.4110
0.7000	0.1582	15.8840
0.7500	0.1592	15.4680
0.8000	0.1577	14.5200
0.9000	0.1018	13.3620
0.9500	0.0978	12.1370
1.0000	0.0500	5.1973

Table 6.7: Rotor geometry - dynamic dimensional SVM

Radius = $0.1627m$.

Angular velocity = $626.45\frac{rad}{s}$.

- **Non dimensional**

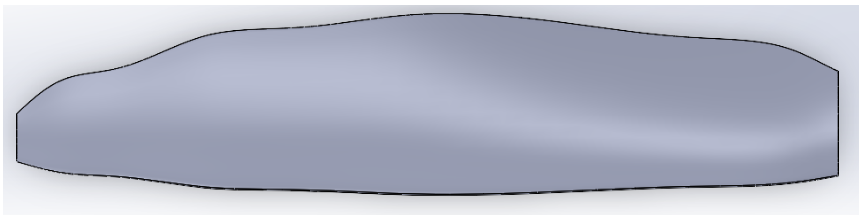


Figure 6.20: Rotor geometry - dynamic non dimensional SVM - top view

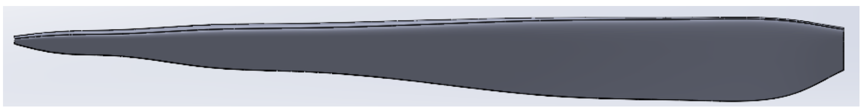


Figure 6.21: Rotor geometry - dynamic non dimensional SVM - front view

r/R	**c/R**	$\boldsymbol{\beta}$ [°]
0.1500	0.1154	20.2460
0.2000	0.1562	27.2750
0.3000	0.1747	27.7460
0.4500	0.1930	22.8990
0.6000	0.1926	16.1250
0.7000	0.1745	14.0710
0.7500	0.1707	13.6480
0.8000	0.1627	12.9370
0.9000	0.1128	10.4260
0.9500	0.0994	10.3060
1.0000	0.0457	5.0804

Table 6.8: Rotor geometry - dynamic non dimensional SVM

Radius = $0.1391m$.

Angular velocity = $571.29\frac{rad}{s}$.

The CT and CP values of the generated rotors are reported in Table 6.9.

ML model	ML				BEM		CFD	
	CT	CT_0012	CP	CP_0012	CT	CP	CT	CP
Stat. dim. GPR	0.15	0.1123	0.0578	0.0437	0.1335	0.0492	-	-
Stat. non dim. GPR	0.15	0.1091	0.0604	0.0442	0.1389	0.0541	-	-
Dyn. dim. GPR	0.15	0.1111	0.0612	0.0472	0.1330	0.0489	-	-
Dyn. non dim. GPR	0.15	0.1107	0.0625	0.0480	0.1314	0.0514	0.1203	0.0509
Stat. dim. SVM	0.15	0.1159	0.0491	0.0367	0.1316	0.0490	-	-
Stat. non dim. SVM	0.15	0.1123	0.0545	0.042	0.1251	0.0527	-	-
Dyn. dim. SVM	0.15	0.1186	0.0482	0.0384	0.1100	0.0393	-	-
Dyn. non dim. SVM	0.15	0.1173	0.0540	0.0426	0.1086	0.0393	0.1010	0.0396

Table 6.9: Comparison of ML, BEM, and CFD values

The results show that for both CT and CP the lowest errors, with respect to BEM and CFD values, are obtained by the GPR models.

Considering only these models, the percentage errors on CT with respect to BEM range between 7% and 14%, while relative to CFD the error, for the single case considered, is about 25%.

For CP, the errors with respect to BEM vary between 10% and 25%, whereas with respect to CFD they are about 23%.

Examining Figure 6.4, it can be noted that the CT value, at which the analyses were performed, has a lower data density in the database, which could affect the model's ability to generate rotors for that CT. To verify this, a simulation was carried out with a target CT of 0.11[11] and the results are shown below.

6.7.3 Analysis at CT = 0.11

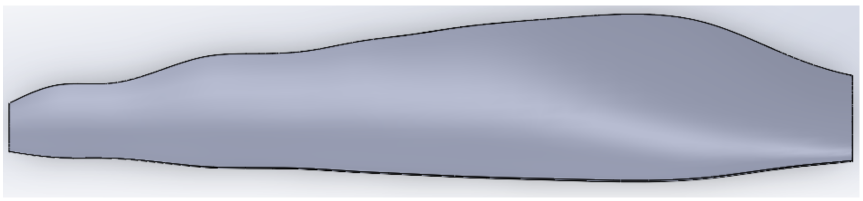


Figure 6.22: Rotor geometry - dynamic non dimensional GPR - top view

[11]The analysis was performed using the dynamic non dimensional GPR model.

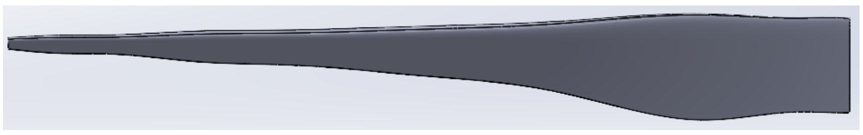


Figure 6.23: Rotor geometry - dynamic non dimensional GPR - front view

r/R	**c/R**	**β [°]**
0.1500	0.1279	47.5170
0.2000	0.1411	41.4190
0.3000	0.1894	33.3550
0.4500	0.1758	21.7730
0.5500	0.1582	17.7840
0.6000	0.1471	16.9480
0.6500	0.1368	15.4490
0.7000	0.1236	14.7620
0.7500	0.1196	14.1620
0.8000	0.1157	13.4660
0.9000	0.0773	11.6300
0.9500	0.0754	11.2890
1.0000	0.0023	10.425

Table 6.10: Rotor geometry - dynamic non dimensional GPR

Radius $= 0.2302m$.
Angular velocity $= 433.95\frac{rad}{s}$.

ML model	**ML**				**BEM**		**CFD**	
	CT	**CT_0012**	**CP**	**CP_0012**	**CT**	**CP**	**CT**	**CP**
Dyn. non dim. GPR	0.11	0.0831	0.0338	0.0257	0.0941	0.0303	0.0917	0.0336

Table 6.11: Comparison of ML, BEM, and CFD values

In this case the percentage errors on CT are approximately 17% for BEM and 20% for CFD. Considering the latter as reference, this represents a slight improvement. For CP, the percentage errors are 12% with BEM and 0.6% with CFD, indicating a significant improvement.

6.8 Conclusions

In this chapter a MATLAB code capable of constructing a Pareto front and generating rotors based on user-provided inputs, using ML models, was presented. The code is highly configurable, so it allows the user to customize numerous geometric and performance parameters.
Two operating modes were analysed in detail:

1. the `'pareto'` mode;
2. the `'diretta'` mode.

In both modes a GA is used with constraints derived from the database.
The analysis of the results highlighted that the GPR models provide more reliable predictions compared to the SVM models. However, a certain discrepancy emerged between the model-predicted values and those obtained via BEM and CFD simulations. The results seem to improve when considering performance targets with higher rotor density in the database but remain not sufficiently reliable for fully autonomous rotor generation, especially when precise performance is desired.
To fully exploit the potential of this code, it is necessary to further improve the predictive capability of the models. This can be achieved, for example, through the use of deep neural networks (Deep Learning) or by increasing the quantity and the quality of training data, in order to expand the database representativeness and reduce generalization error.

Chapter 7

Rotors Optimization

7.1 Introduction

This chapter explains the functioning of a MATLAB code developed to optimize rotors. The code offers two main operating modes, which are described below:

1. `‘costruzione’` mode;
2. `‘pareto’` mode.

To optimize the rotors, the code employs a GA to generate new geometries and trained ML models, with the aim of evaluating their performance. The code makes it possible to modify a rotor in order to increase the CT while keeping the CP constant, or to decrease the CP while holding the CT constant, thereby improving efficiency.
The T-Motor 15 X 5 CF rotor will be optimized, and the results will be compared with those obtained via BEM and CFD analyses.

7.2 T-Motor 15 X 5 CF Rotor Geometry

The geometry of the T-Motor 15 X 5 CF rotor is given in Table 7.1. Of the six columns only *Station*, *Pitch* and *Chord* are used, as these are the only data accepted by the trained models. The values along the blade span must be interpolated[1] at the stations required by the various ML models for their analyses, and the chord must be normalised with respect to the radius. Consequently, the geometry matrix employed is the one shown in Table 7.2.

[1]Linear interpolation was adopted in this case.

Station (in)	r/R	Pitch (deg)	Chord (in)	Xqc (in)	Zqc (in)
1.240	0.165	17.171	0.982	-0.169	0.028
1.550	0.207	17.284	1.101	-0.177	0.038
1.860	0.248	21.264	1.349	-0.148	0.025
2.170	0.289	19.863	1.449	-0.141	0.026
2.480	0.331	17.925	1.498	-0.135	0.027
2.790	0.372	16.355	1.511	-0.132	0.025
3.100	0.413	15.090	1.497	-0.132	0.025
3.410	0.455	13.921	1.467	-0.138	0.025
3.720	0.496	12.928	1.427	-0.145	0.028
4.030	0.537	12.168	1.379	-0.153	0.031
4.340	0.579	11.468	1.329	-0.159	0.031
5.890	0.785	9.385	1.008	-0.175	0.039
6.510	0.868	8.527	0.860	-0.173	0.047
6.820	0.909	8.128	0.764	-0.164	0.059
7.130	0.951	7.833	0.633	-0.136	0.079
7.285	0.971	7.629	0.530	-0.102	0.092
7.440	0.992	7.987	0.310	0.011	0.100

Table 7.1: T Motor 15 X 5 Carbon Fiber geometry [31]

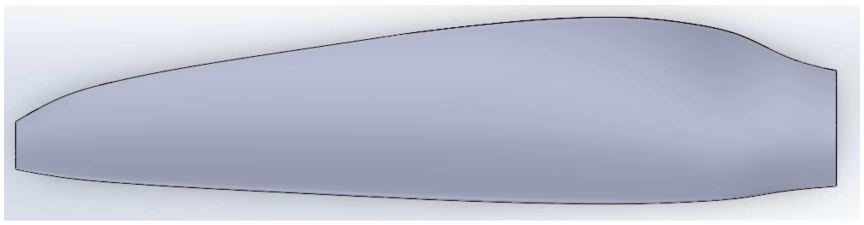


Figure 7.1: T-motor 15 X 5 CF geometry - top view

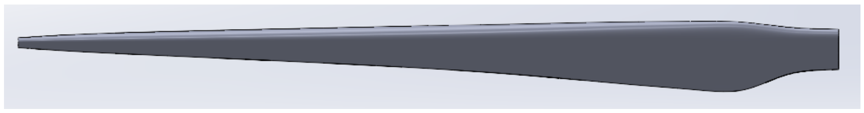


Figure 7.2: T-motor 15 X 5 CF geometry - front view

r/R	**c/R**	**β [°]**
0.1500	0.1253	17.1306
0.2000	0.1442	17.2652
0.2500	0.1805	21.1957
0.3000	0.1949	19.3554
0.3500	0.2005	17.1974
0.4000	0.2002	15.4911
0.4500	0.1961	14.0602
0.5000	0.1896	12.8539
0.5500	0.1818	11.9513
0.6000	0.1728	11.2557
0.6500	0.1624	10.7501
0.7000	0.1521	10.2445
0.7500	0.1417	9.7389
0.8000	0.1308	9.2299
0.8500	0.1189	8.7131
0.9000	0.1047	8.2156
0.9500	0.0848	7.8400
1.0000	0.0302	8.1234

Table 7.2: ML model geometry - T Motor 15 X 5 Carbon Fiber

7.3 Operating conditions

The analyses are carried out at $J = 0$ to enable the use of all the trained models. Moreover, the environmental conditions correspond to sea level, while the rotor rotational speed is set to $314.16\frac{rad}{s}$, i.e. 3000 RPM.

7.4 Input

The inputs that can be selected in this code, listed in Appendix C.1, are very similar to those described in Chapter 6; therefore, only the differing ones are presented here.

- **Type of analysis** (`'costruzione'` o `'pareto'`): this is the main choice to be made.

  Selecting `'costruzione'`, the code optimises the loaded rotor. Selecting `'pareto'`, the code generates the Pareto front together with the various rotors optimised for different percentages of allowable modification. The two modes are described in detail in the following paragraphs.

- **Type of optimisation**: specifies the direction in which the code proceeds to improve rotor efficiency.

  If the aim is to increase CT, the code modifies the initial rotor to raise CT while keeping CP constant. If the aim is to reduce CP, the code modifies the initial rotor to lower CP while keeping CT constant.

- **Tolerances**: different types of tolerance can be implemented depending on the chosen analysis.

  When `'costruzione'` is selected, only a single tolerance value can be entered, whereas with `'pareto'` a vector of tolerances can be provided. This arrangement allows identification of the tolerance at which the desired improvements are achieved. The rotor is modified solely by altering its geometry (radius excluded) and not the operating conditions. For example, a tolerance of 5 % permits the normalised chord and β values to vary by that percentage.

In addition to defining these parameters, it is necessary to import the geometry of the rotor to be optimised. Three vectors[2] are required:

1. **Station**: indicates the spanwise stations along the rotor for which data are available.

2. **Chord**: vector of the rotor chord lengths along the blade span.

3. **Pitch**: vector of the pitch angles of the profiles along the blade span.

The values are then processed automatically into the format accepted by the ML model in use.

Since the two analysis types share most of the code, the common parts are presented first, followed by a detailed examination of the two analyses.

7.5 Common codes

7.5.1 Tolerance application

The tolerance is applied by modifying the upper and lower bounds of the GA[3]. The function implemented for this purpose is shown below.

[2]The rotor diameter must also be available.

[3]Before the tolerance is applied, `ub` and `lb` are identical and equal to the vector describing the characteristics of the rotor to be modified.

Listing 7.1: Tolerance function

```
function [lb, ub] = minimi_massimi(rotore, tolleranza_geometria,
    zeri_e_uni_corda, zeri_e_uni_beta)
     parti_geometria = rotore(1:(sum(zeri_e_uni_corda +
    zeri_e_uni_beta)));
     lb = parti_geometria - tolleranza_geometria*parti_geometria;
     ub = parti_geometria + tolleranza_geometria*parti_geometria;

     lb = [lb, rotore(((sum(zeri_e_uni_corda + zeri_e_uni_beta))+1):
    end)];
     ub = [ub, rotore(((sum(zeri_e_uni_corda + zeri_e_uni_beta))+1):
    end)];
end
```

The specified tolerance causes both the lower and upper limits within which the GA explores the various configurations to be modified by the given percentage.
The code also includes a function that prevents the algorithm from considering geometries whose variations in normalised chord or β exceed those present in the database, like the one introduced in Chapter 6.

7.5.2 Fitness function

Two fitness functions are used, depending on whether the aim is to construct the Pareto front or to generate a single rotor.
For rotor generation, the fitness function employed is very similar to that implemented in the `'diretta'` analysis of Chapter 6 and it is reported in Appendix C.2.
The function adopted for the Pareto front is practically identical to the one shown in Appendix B.2, so it is not reproduced here[4].

7.5.3 Constraint function

As in Chapter 6, the constraint function ensures that, if the rotor is not being optimised with respect to, for example, CT, the generated rotors remain within the desired tolerance on CT[5].
The function is provided in Appendix C.3.

[4]The only difference is that the check on the tip Mach number is omitted, because that value is set by selecting the rotational speed and radius.

[5]CT is kept in the vicinity of the initial rotor value.

7.6 Analysis: `'costruzione'`

The modified rotor is generated by means of a GA, to which the inputs and functions described previously are supplied.

7.6.1 GA

The GA modifies the input rotor by varying its geometric characteristics within the specified tolerance. For each geometry created, the algorithm evaluates performance using the previously trained ML models. The options employed are identical to those presented in Chapter 6 for the `'diretta'` analysis, except for the initial population: in the present case, the population is generated by forming a matrix that contains the initial rotor together with random perturbations whose maximum amplitude equals the prescribed tolerance.

Listing 7.2: InitialPopulationMatrix

```
% Creo una popolazione iniziale con il rotore + variazioni casuali
initialPopulation = repmat(rotore, grandezza_popolazione, 1);
perturbation = tolleranza_geometria*(ub - lb).*randn(
    grandezza_popolazione, nVars);
perturbation(1,:) = 0;
initialPopulation = initialPopulation + perturbation;
initialPopulation = max(min(initialPopulation, ub), lb);
```

After the modified rotor has been generated, a 3D view of both the modified and initial rotor is displayed, and the expected percentage improvement obtained through the implemented modifications is reported.
An example of the output is shown below.

```
=== FINAL RESULTS ===
Percentage change CT: 5.1136 %
Percentage change CP: 0.0000 %
```

7.7 Analysis: `'pareto'`

When `'pareto'` is selected as the analysis type, the procedure described for `'costruzione'` is repeated a number of times equal to the length of the tolerance vector, and the Pareto front is also constructed. This code produces neither numerical values nor geometries as output, but it serves as an initial analysis to determine the tolerance to be adopted. It is expected that, as the tolerance

increases, the modified rotors will progressively approach the Pareto front[6]. The front is again built using MATLAB's `@gamultiobj` function, with the same options and inputs presented in Chapter 6[7].
Since this part of the analysis yields neither rotors nor numerical values, only an example result is provided below. The example is carried out using the T-Motor 15 X 5 CF as the input rotor, and the geometry tolerances used are:

$$5\%, 20\%, 50\%, 90\%$$

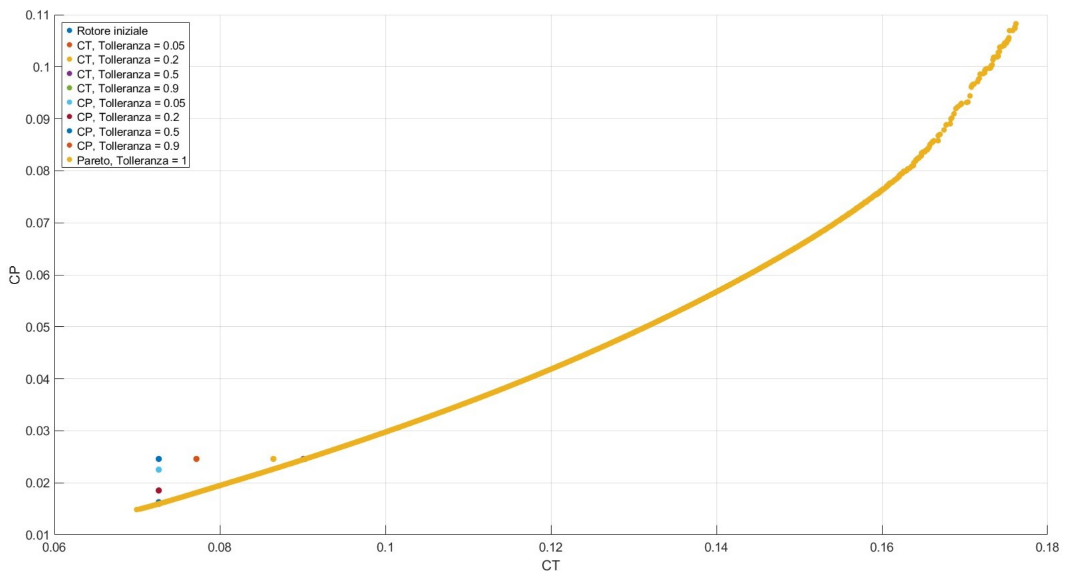


Figure 7.3: Example of the `'pareto'` analysis

From Figures 7.3 and 7.4 it is apparent that, as the tolerance increases, the rotors move progressively closer to the Pareto front. In particular, the rotors generated with a 90 % tolerance practically coincide with the front and they are therefore not visible. The plot is useful because it enables a comparison between the performance of the initial rotor (as evaluated by the ML model) and that of the modified rotors, and it assists in selecting an appropriate tolerance level for modifying the rotor under examination. Although increasing the tolerance moves the design farther from the original rotor characteristics, the graph shows that it provides a greater chance of generating a rotor with improved performance.

[6]For each specified tolerance, the rotor is modified both to increase CT at constant CP and to decrease CP at constant CT.

[7]The initial population is slightly different, as it also contains the rotor to be modified and the already modified rotors.

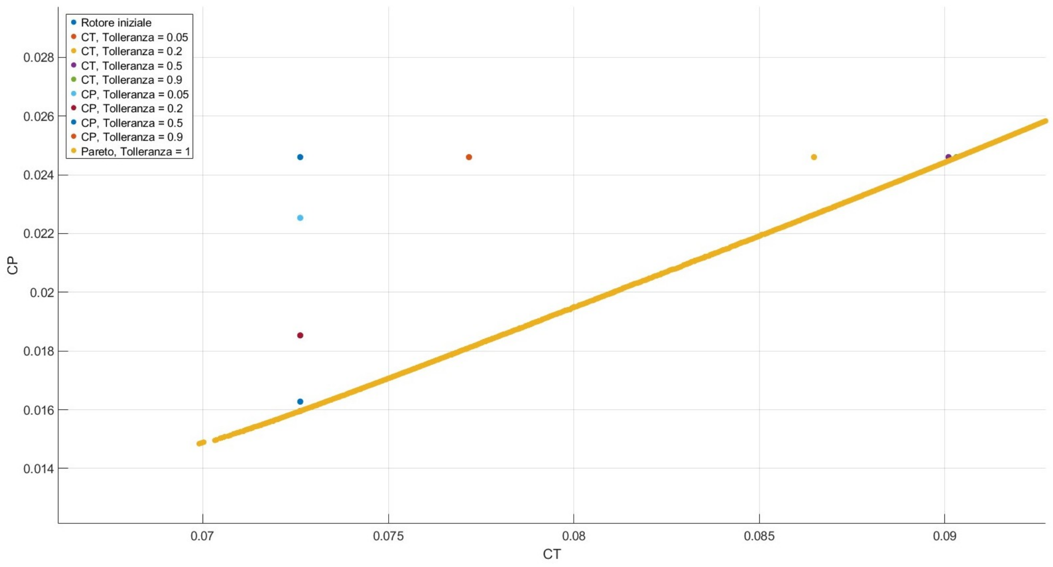


Figure 7.4: Example of the `'pareto'` analysis - zoom

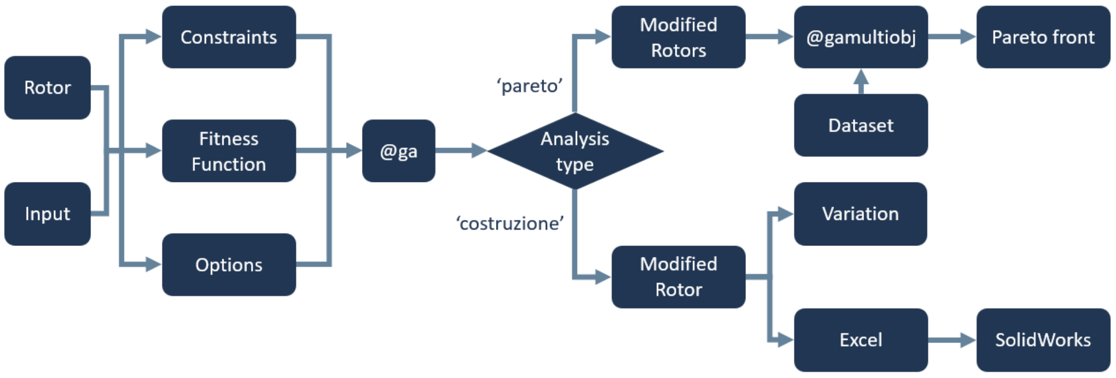


Figure 7.5: Flow chart MATLAB code

Once again, when `'costruzione'` is selected, the rotor data and the chosen airfoil are exported to Excel so that they can subsequently be generated in SolidWorks. The resulting designs must then be evaluated with more accurate analyses such as BEM or CFD to verify that the modified rotor actually provides a performance benefit[8]. As with the code presented in Chapter 6, the advantage of using ML for this type of study lies in the analysis time, which is an order of magnitude shorter

[8]It should be noted that the trained models may not have captured the relationships among the various features correctly; consequently, the results may sometimes differ from expectations.

than that required by BEM[9]. This allows a much larger number of geometries to be analysed simultaneously, enabling different solutions to be explored.
The operation of the MATLAB code is summarised in Figure 7.5.

7.8 Results

This section presents the results obtained by performing the `'costruzione'` analysis. The outcomes from all trained ML models are compared for a tolerance of 5% while improving CT. Subsequently, the results are validated through BEM and CFD analyses[10].
For each ML model used, a table is provided showing the geometry of the modified rotor (similar to Table 7.2), along with two views of the rotor (a top view and a front one), as illustrated in Figures 7.1 and 7.2.

7.8.1 GPR

Static

- **Dimensional**

r/R	c/R	β [°]
0.1500	0.1191	17.9850
0.2000	0.1460	18.1260
0.3000	0.2046	20.3170
0.4500	0.2059	13.9270
0.5500	0.1907	12.5130
0.6000	0.1756	11.3830
0.6500	0.1621	10.7250
0.7000	0.1516	10.1930
0.7500	0.1488	9.7828
0.8000	0.1373	9.4237
0.9000	0.1099	8.6243
0.9500	0.0842	8.1798
1.0000	0.0304	7.7234

Table 7.3: Modified T-Motor – dimensional static GPR

[9]A ML evaluation takes on the order of tenths or hundredths of a second, whereas a BEM analysis takes several seconds.

[10]The NACA 4412 airfoil is employed.

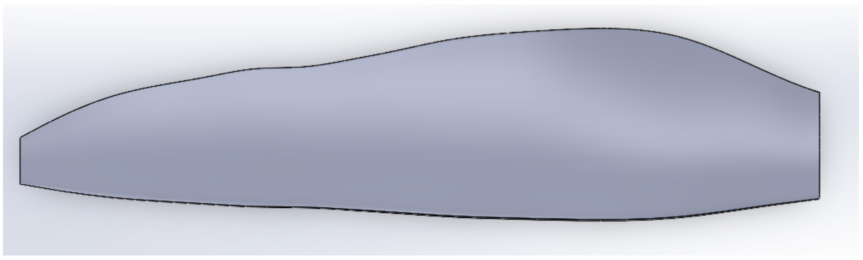


Figure 7.6: Modified T-Motor – dimensional static GPR – top view

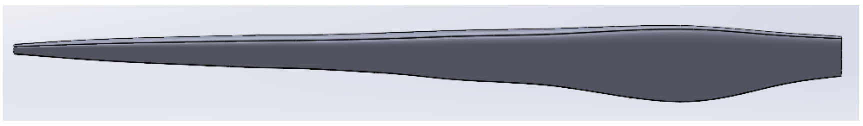


Figure 7.7: Modified T-Motor – dimensional static GPR – front view

- **Non dimensional**

r/R	c/R	$\boldsymbol{\beta}$ [°]
0.1500	0.1191	17.9810
0.2000	0.1513	18.1230
0.3000	0.2046	20.3130
0.4500	0.2059	14.3360
0.5500	0.1906	12.5440
0.6000	0.1643	11.8140
0.6500	0.1543	11.2660
0.7000	0.1482	10.6320
0.7500	0.1488	10.2200
0.8000	0.1373	9.6882
0.9000	0.1099	8.6206
0.9500	0.0889	8.2242
1.0000	0.0317	7.7201

Table 7.4: Modified T-Motor – non dimensional static GPR

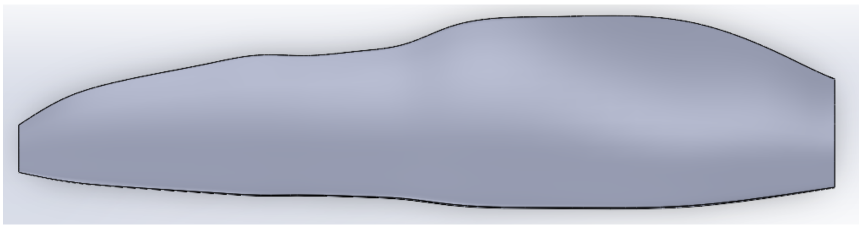


Figure 7.8: Modified T-Motor – non dimensional static GPR – top view

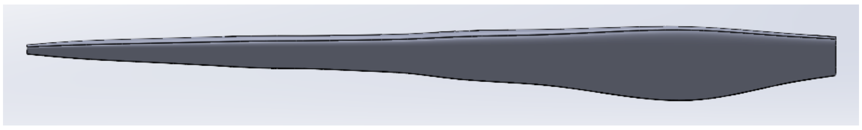


Figure 7.9: Modified T-Motor – non dimensional static GPR – front view

Dynamic

- **Dimensional**

r/R	**c/R**	$\boldsymbol{\beta}$ [°]
0.1500	0.1264	17.9830
0.2000	0.1371	18.1260
0.3000	0.2046	20.3190
0.4500	0.2059	14.7350
0.5500	0.1863	12.5450
0.6000	0.1679	11.8150
0.6500	0.1646	11.2840
0.7000	0.1597	10.7510
0.7500	0.1488	10.2240
0.8000	0.1373	9.6898
0.9000	0.1099	8.6245
0.9500	0.0867	8.1358
1.0000	0.0317	7.7189

Table 7.5: Modified T-Motor – dimensional dynamic GPR

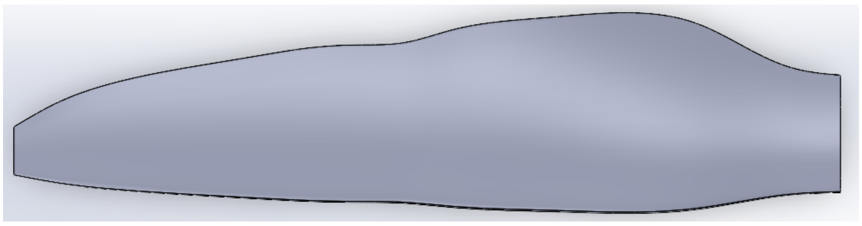


Figure 7.10: Modified T-Motor – dimensional dynamic GPR – top view

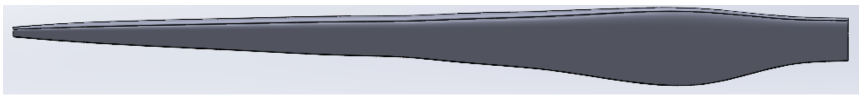


Figure 7.11: Modified T-Motor – dimensional dynamic GPR – front view

- **Non dimensional**

r/R	**c/R**	$\boldsymbol{\beta}$ [°]
0.1500	0.1191	17.9830
0.2000	0.1450	18.1250
0.3000	0.2046	20.3190
0.4500	0.2059	13.6610
0.5500	0.1827	12.5450
0.6000	0.1787	11.8160
0.6500	0.1637	11.2840
0.7000	0.1596	10.7500
0.7500	0.1488	10.2250
0.8000	0.1373	9.6900
0.9000	0.1098	8.6249
0.9500	0.0877	8.2184
1.0000	0.0317	7.7187

Table 7.6: Modified T-Motor – non dimensional dynamic GPR

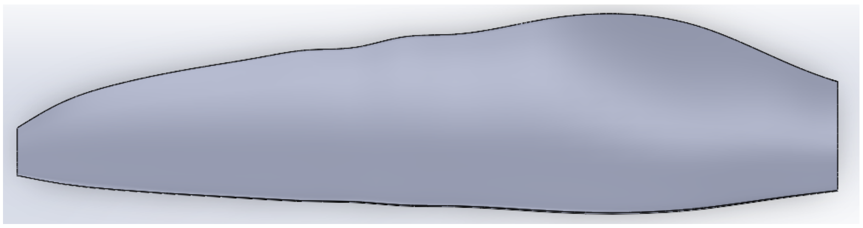


Figure 7.12: Modified T-Motor – non dimensional dynamic GPR – top view

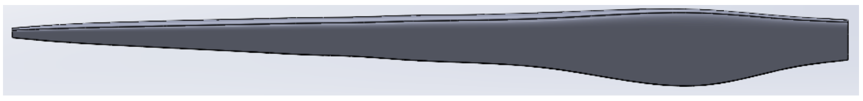


Figure 7.13: Modified T-Motor – non dimensional dynamic GPR – front view

7.8.2 SVM

Static

- **Dimensional**

r/R	**c/R**	$\boldsymbol{\beta}$ [°]
0.1500	0.1191	17.9860
0.2000	0.1460	18.1280
0.3000	0.2022	19.8920
0.4500	0.2059	14.7540
0.6000	0.1761	11.8180
0.7000	0.1535	10.7530
0.7500	0.1488	10.2260
0.8000	0.1373	9.6886
0.9000	0.1099	8.6251
1.0000	0.0317	7.7189

Table 7.7: Modified T-Motor – dimensional static SVM

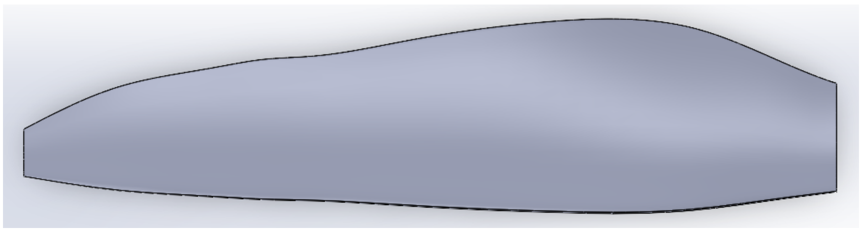


Figure 7.14: Modified T-Motor – dimensional static SVM – top view

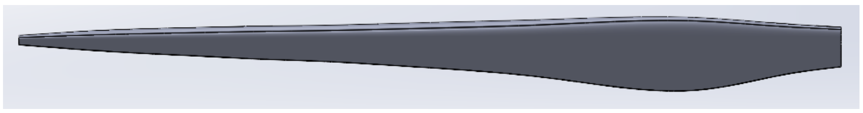


Figure 7.15: Modified T-Motor – dimensional static SVM – front view

- **Non dimensional**

r/R	**c/R**	$\boldsymbol{\beta}$ [°]
0.1500	0.1191	17.9830
0.2000	0.1513	18.1280
0.3000	0.1968	20.2580
0.4500	0.2059	14.3660
0.6000	0.1739	11.8150
0.7000	0.1492	10.6820
0.7500	0.1488	10.2230
0.8000	0.1373	9.6895
0.9000	0.1099	8.6258
1.0000	0.0317	7.7178

Table 7.8: Modified T-Motor – non dimensional static SVM

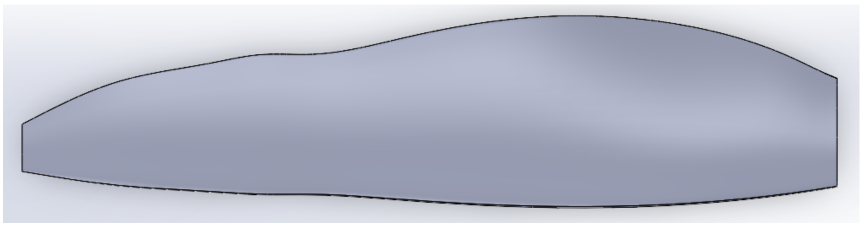


Figure 7.16: Modified T-Motor – non dimensional static SVM – top view

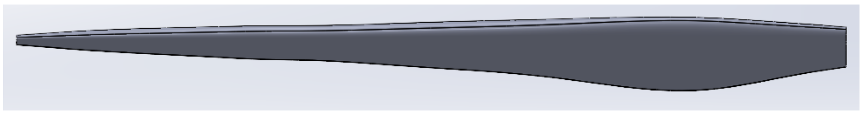


Figure 7.17: Modified T-Motor – non dimensional static SVM – front view

Dynamic

- **Dimensional**

r/R	c/R	β [°]
0.1500	0.1238	17.9850
0.2000	0.1514	18.1260
0.3000	0.2045	20.3220
0.4500	0.1983	14.7620
0.6000	0.1659	11.8170
0.7000	0.1498	10.6320
0.7500	0.1488	10.2240
0.8000	0.1373	9.6810
0.9000	0.0999	8.6255
0.9500	0.0890	8.2287
1.0000	0.0317	7.7179

Table 7.9: Modified T-Motor – dimensional dynamic SVM

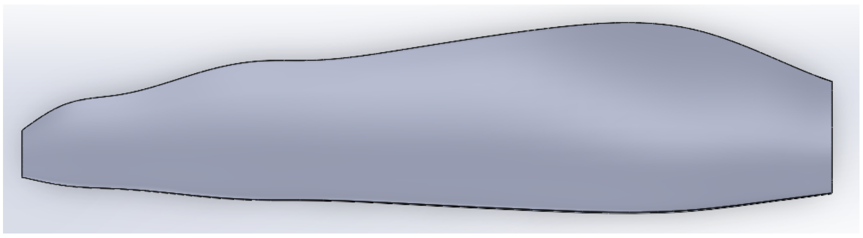


Figure 7.18: Modified T-Motor – dimensional dynamic SVM – top view

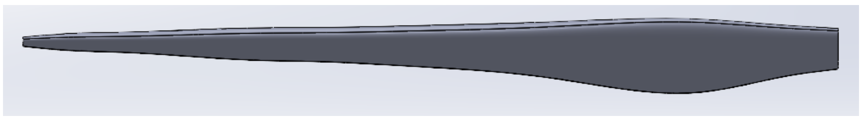


Figure 7.19: Modified T-Motor – dimensional dynamic SVM – front view

- **Non dimensional**

r/R	**c/R**	$\boldsymbol{\beta}$ [°]
0.1500	0.1191	17.9830
0.2000	0.1514	18.1250
0.3000	0.2038	20.3210
0.4500	0.1964	14.7620
0.6000	0.1646	11.8150
0.7000	0.1513	10.6310
0.7500	0.1488	10.2240
0.8000	0.1373	9.6834
0.9000	0.1053	8.6255
0.9500	0.0890	8.2266
1.0000	0.0317	7.7179

Table 7.10: Modified T-Motor – non dimensional dynamic SVM

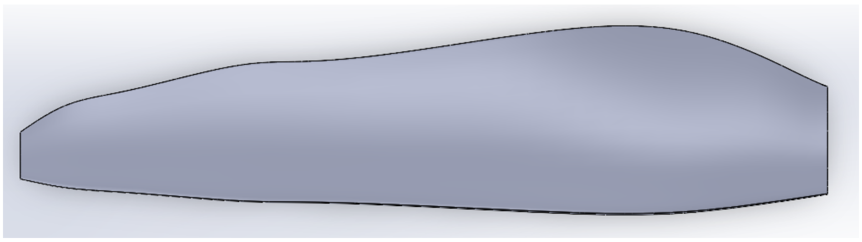


Figure 7.20: Modified T-Motor – non dimensional dynamic SVM – top view

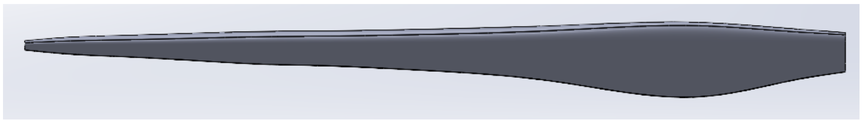


Figure 7.21: Modified T-Motor – non dimensional dynamic SVM – front view

ML GPR model	**CT variation [%]**		
	ML	**BEM**	**CFD**
Static dimensional	3.7618	4.3554	-
Static non dimensional	5.1123	6.1516	-
Dynamic dimensional	6.2621	7.2392	-
Dynamic non dimensional	6.4648	6.6772	13.6709

Table 7.11: CT variation - GPR

ML GPR model	**CP variation [%]**		
	ML	**BEM**	**CFD**
Static dimensional	0	5.1540	-
Static non dimensional	0	7.3074	-
Dynamic dimensional	0	8.8190	-
Dynamic non dimensional	0	8.0327	6.9767

Table 7.12: CP variation - GPR

ML SVM model	**CT variation [%]**		
	ML	**BEM**	**CFD**
Static dimensional	7.0349	7.2060	-
Static non dimensional	6.8367	6.1767	-
Dynamic dimensional	6.8052	4.8865	-
Dynamic non dimensional	6.5451	5.3042	12.4051

Table 7.13: CT variation - SVM

ML SVM model	**CP variation [%]**		
	ML	**BEM**	**CFD**
Static dimensional	0	8.6283	-
Static non dimensional	0	7.3657	-
Dynamic dimensional	0	5.8440	-
Dynamic non dimensional	0	6.3354	5.8140

Table 7.14: CP variation - SVM

The results presented in Tables 7.11 and 7.13 show that, by following the modifications suggested by the ML, there is indeed an improvement in CT. Moreover, the percentage change is roughly the same across all models, and the values are very close to those obtained with the BEM. For the only two models analysed with CFD, however, the improvement is greater. Tables 7.12 and 7.14 indicate that, although the code was constrained to produce zero variation in CP, analyses with BEM and CFD nonetheless show a percentage increase that sometimes is even larger than that observed for CT. This is because the model introduces some error in evaluating rotor performance and, in addition, the T-Motor 15 X 5 CF exhibits relatively low hover CP values (≈ 0.025) compared with CT (≈ 0.076). Consequently, small absolute variations lead to more pronounced percentage changes.

In hover, the figure of merit (FoM) of a rotor, which is an indicator of its aerodynamic efficiency, is defined as

$$FoM = \frac{CT^{\frac{3}{2}}}{CP} \tag{7.1}$$

This relation shows that the FoM depends non linearly on the CT, whereas its dependence on CP is linear. Consequently, an increase in CT can offset a proportionally similar, or even slightly larger, increase in CP, still leading to an overall improvement in efficiency. Tables 7.15 and 7.16 report the percentage change in FoM for the various modified rotors. In every case considered the FoM increases, confirming that the models have successfully optimised the rotor. Moreover, taking the CFD values as a reference, it is observed a large percentage rise in FoM, indicating that the rotor's performance improves significantly when the modifications proposed by the models are applied.

ML GPR model	**FoM variation [%]**		
	ML	**BEM**	**CFD**
Static dimensional	5.6954	1.3782	-
Static non dimensional	7.7656	1.9202	-
Dynamic dimensional	9.5387	2.0530	-
Dynamic non dimensional	9.8523	1.9887	13.2882

Table 7.15: FoM variation - GPR

ML SVM model	**FoM variation [%]**		
	ML	**BEM**	**CFD**
Static dimensional	10.7358	2.1846	-
Static non dimensional	10.4284	1.9010	-
Dynamic dimensional	10.3795	1.4876	-
Dynamic non dimensional	9.9766	1.6227	12.6253

Table 7.16: FoM variation - SVM

7.9 Conclusions

In this chapter, the operation of a MATLAB code that makes it possible to optimize rotors with the aim of increasing their efficiency was illustrated. The results have shown how all the models are able to modify the T-Motor 15 X 5 CF rotor in order to improve its performance. Therefore, while exhibiting some errors concerning the prediction of rotor performance, especially on CP, all the models succeeded in modifying the rotor while respecting the imposed geometric tolerances and in improving efficiency in hover.
The percentage variation of CP (which is not desired) influences the analyses carried out but, considering the CFD analyses, it does not seem to affect the successful outcome of the optimization. Also in this case, to obtain better results, it proves necessary to retrain the ML models on datasets of larger size and higher quality, or to use more advanced models.

Chapter 8

Rotor Performance Analysis

8.1 Introduction

In addition to the generation and the optimization of rotors, the trained ML models can simply be employed to predict the performance of existing rotors. This chapter reports the performance analyses carried out on the T-Motor 15 X 5 CF rotor, already presented in Chapter 7. All the trained models are used in these analyses. Specifically, to allow a comparison between the static and dynamic models, the performance analyses are conducted at $J = 0$. For the dynamic models, the trend of CT and CP as J varies is also provided. The results are compared with BEM, CFD and experimental analyses (when available).

8.2 Data Collection

Experimental data available in the literature were used for the performance analyses of the T-Motor 15 X 5 CF rotor.
Specifically, the reference data were extracted from a NASA publication [31], which provides rotor performance measurements under known conditions. These data were used as a benchmark to analyse the errors made by the models.

8.2.1 NASA Data

From the data in Table 8.1, only the columns Density[1], RPM, Fz and Mz are used. After extracting and converting the values to the International System of Units (SI), formulas (3.1) and (3.2) were used to non dimensionalize the force and

[1]Used for non dimensional ML models and to non dimensionalize forces and moments.

moment values[2]. The NASA data are useful for analysing the hover performance of the rotor, but they do not provide data for $J > 0$, for which BEM and CFD analyses are used.

Speed	**q**	**Density**	**RPM**	**Fx**	**Fy**	**Fz**	**Mx**	**My**	**Mz**	**Vesc**	**I1**
(ft/s)	(lb/ft^2)	(slug/ft^3)		(lb)	(lb)	(lb)	(in-lb)	(in-lb)	(in-lb)	(V)	(A)
0.00	0.00	0.00238	2000	-0.014	-0.001	0.459	0.036	0.073	0.358	16.726	0.976
0.00	0.00	0.00238	2500	-0.013	-0.000	0.720	0.058	0.066	0.558	16.715	1.772
0.00	0.00	0.00238	3000	-0.014	0.003	1.049	0.015	0.067	0.802	16.698	2.934
0.00	0.00	0.00238	3500	-0.020	0.006	1.427	-0.008	0.071	1.070	16.676	4.638
0.00	0.00	0.00238	4000	-0.031	-0.003	1.897	0.024	0.136	1.414	16.633	7.541
0.00	0.00	0.00238	4500	-0.028	-0.001	2.388	0.028	0.153	1.770	16.574	11.960

Table 8.1: Experimental data for the T-Motor 15 × 5 CF rotor (NASA) [31]

8.3 Results at $J = 0$

Several tables containing NASA data, BEM results, and predictions from the ML models are presented. For each RPM value, the percentage error is calculated using:

$$Error = \left| \frac{value_{model} - value_{NASA}}{value_{NASA}} \right| \cdot 100 \tag{8.1}$$

The average error is calculated by averaging the error over the number of rotational speeds considered.

8.3.1 CT

In Table 8.2 the NASA and BEM values are reported, while in Tables 8.3 and 8.4 there are the values predicted by the ML models. In the tables, the mean percentage errors with respect to NASA values are also reported.

[2]Since a moment is available, the formula $CQ = \frac{M_z}{\rho n^2 D^5}$ must be used and it is multiplied by 2π to obtain CP.

RPM	**NASA**	**BEM**
2000	0.071095	0.0647
2500	0.071374	0.0676
3000	0.072213	0.0692
3500	0.072173	0.0708
4000	0.073457	0.0716
4500	0.073063	0.0723
Error [%]	-	3.99

Table 8.2: NASA and BEM CT values

RPM	**Stat. dim.**	**Dyn. dim.**	**Stat. non dim.**	**Dyn. non dim.**
2000	0.075674	0.069101	0.071708	0.069645
2500	0.077307	0.070898	0.075938	0.073290
3000	0.07891	0.072626	0.079640	0.076311
3500	0.080478	0.074277	0.082835	0.078730
4000	0.082001	0.075841	0.085631	0.080507
4500	0.083475	0.077313	0.088146	0.081564
Error [%]	10.24	2.67	10.97	6.79

Table 8.3: CT values for ML models – GPR

RPM	**Stat. dim.**	**Dyn. dim.**	**Stat. non dim.**	**Dyn. non dim.**
2000	0.082869	0.075539	0.077342	0.075668
2500	0.084346	0.077832	0.081139	0.080270
3000	0.085840	0.080006	0.084567	0.084445
3500	0.087340	0.082047	0.087657	0.088075
4000	0.088838	0.083947	0.090527	0.090928
4500	0.090323	0.085694	0.093284	0.092697
Error [%]	19.86	11.89	18.66	18.09

Table 8.4: CT values for ML models – SVM

Analysing the results, it can be noted that the ML models captured the trend of CT increasing with RPM although, if compared to NASA values, all models show a more pronounced increase in CT with RPM, while compared to BEM values the variation is similar. Among the various trained models, both for GPR and for SVM, the one with the lowest prediction error (compared to NASA values) is the model trained with the dynamic dimensional database, as it shows a less marked increase of CT with RPM. However, in general, all ML models of GPR perform better than those of SVM. Moreover, for this particular case, the error made by the dynamic dimensional GPR model is lower than the one made by BEM[3]. It is also important to note that in the transition from static to dynamic models the percentage errors decrease, indicating that having more data available, even if not at $J = 0$, allows the models to generalise better.

8.3.2 CP

In Table 8.5 the NASA and BEM values are reported, while in Tables 8.6 and 8.7 there are the values predicted by the ML models. Once again, the tables list the mean percentage errors with respect to the NASA values.

RPM	**NASA**	**BEM**
2000	0.023227	0.0221
2500	0.023170	0.0220
3000	0.023126	0.0220
3500	0.022668	0.0219
4000	0.022935	0.0218
4500	0.022684	0.0217
Error [%]	-	4.43

Table 8.5: NASA and BEM CP values

RPM	**Stat. dim.**	**Dyn. dim.**	**Stat. non dim.**	**Dyn. non dim.**
2000	0.027410	0.024442	0.024801	0.023000
2500	0.027297	0.024519	0.025080	0.023163
3000	0.027208	0.024603	0.025588	0.023556
3500	0.027140	0.024691	0.026228	0.024118
4000	0.027092	0.024785	0.026931	0.024752
4500	0.027065	0.024882	0.027643	0.025342
Error [%]	18.44	7.35	13.44	4.82

Table 8.6: CP values for ML models – GPR

[3]This is not a definitive conclusion since results could differ with other rotors.

RPM	**Stat. dim.**	**Dyn. dim.**	**Stat. non dim.**	**Dyn. non dim.**
2000	0.031702	0.028633	0.026822	0.025245
2500	0.031505	0.028758	0.026600	0.025453
3000	0.031392	0.028903	0.026920	0.026126
3500	0.031360	0.029066	0.027708	0.027195
4000	0.031410	0.029248	0.028940	0.028539
4500	0.031541	0.029447	0.030623	0.029988
Error [%]	37.09	26.32	21.33	18.02

Table 8.7: CP values for ML models – SVM

The percentage errors for almost all the models are higher than the ones for CT. The reason is that the trend of CP with rotational speed is more complex than the one of CT, and the models fail to capture it. The models with the lowest mean percentage error, for both GPR and SVM, are the dynamic non dimensional ones. Again, the GPR models exhibit lower percentage errors than the SVMs; therefore, the model with the lowest mean percentage error is the dynamic non dimensional GPR, even though it shows an excessive increase in CP as the RPM rises and thus a large error at high RPM (≈ 20 %). A more constant error across the various RPMs is maintained by the dynamic dimensional GPR, which proves better because it is more reliable. Moreover, even in this case, the errors made by the dynamic models are lower than those of the static models, confirming the earlier observations.

8.4 Results at $J > 0$

Since the NASA data provide no values for $J > 0$, the ML models are compared with the results of BEM and CFD.

8.4.1 CT trend

By analysing Figures 8.2–8.6, it can be seen that all the ML models captured the trend of CT as the advance ratio varies. Moreover, the models understood that higher rotational speeds imply higher CT. Compared with the non dimensional models, the dimensional ones display a less pronounced increase in CT with RPM, more in line with the behaviour obtained from the BEM analysis.

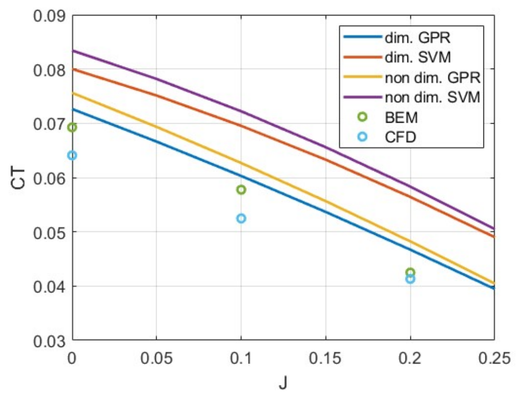


Figure 8.1: 3000 RPM analysis

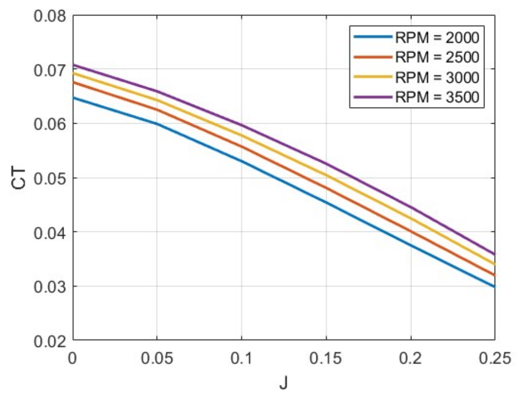


Figure 8.2: BEM

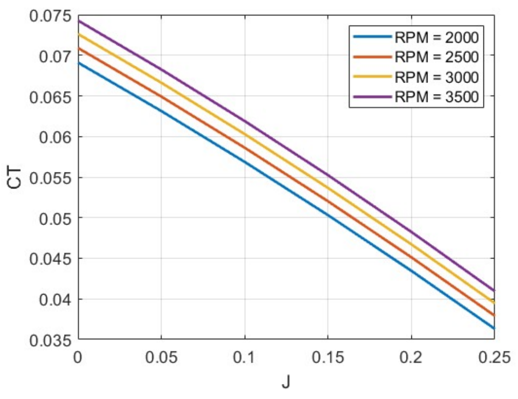


Figure 8.3: Dimensional GPR

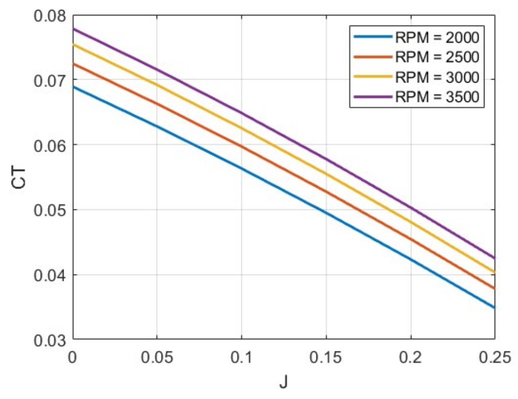


Figure 8.4: Non dimensional GPR

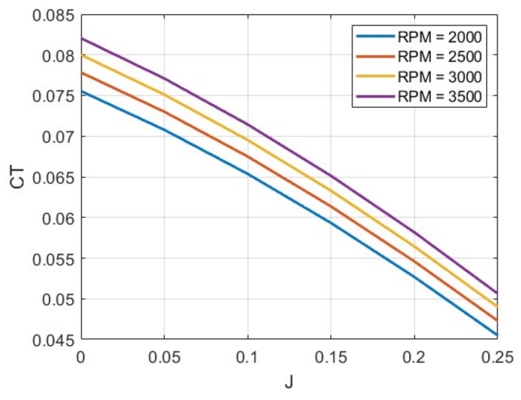


Figure 8.5: Dimensional SVM

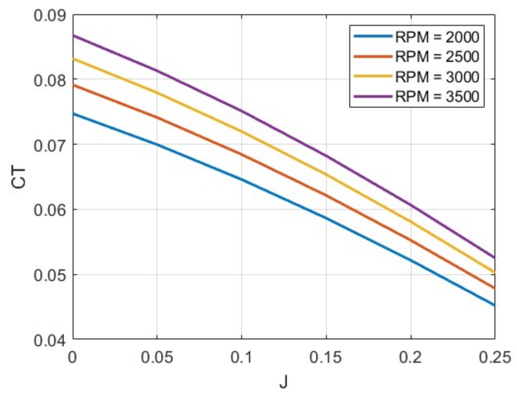


Figure 8.6: Non dimensional SVM

Figure 8.1 shows that the model which most closely matches the data obtained from the BEM and CFD simulations is, once again, the dimensional GPR. The errors of this model with respect to the CFD results are about 10 %, and the model exhibits a slightly sharper decrease in CT as J increases. Overall, both GPR models perform better than the SVM models, confirming the earlier findings.

8.4.2 CP trend

In this case, the conclusions reached by analysing Figures 8.8-8.12 are similar to the ones for CT. Indeed, all the models have understood that having higher rotational speeds entails having higher CP and, again, it can be found that the non dimensional models show a more accentuated variation with the RPM.

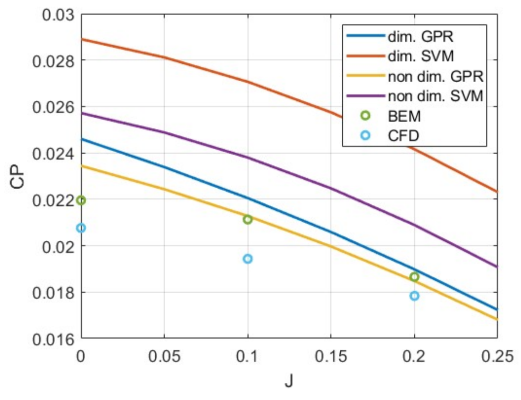


Figure 8.7: 3000 RPM analysis

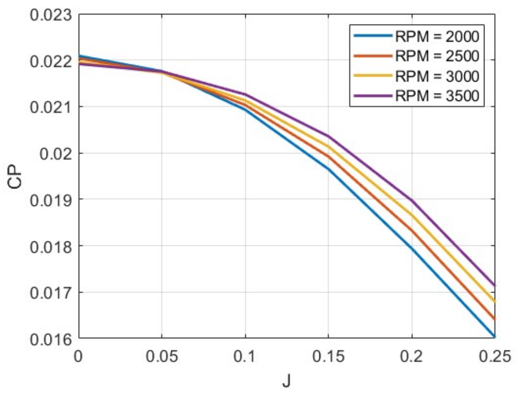


Figure 8.8: BEM

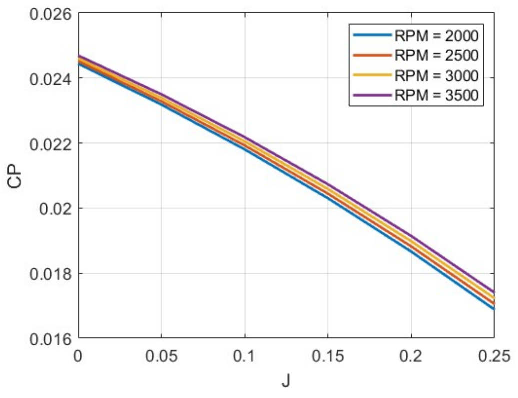


Figure 8.9: Dimensional GPR

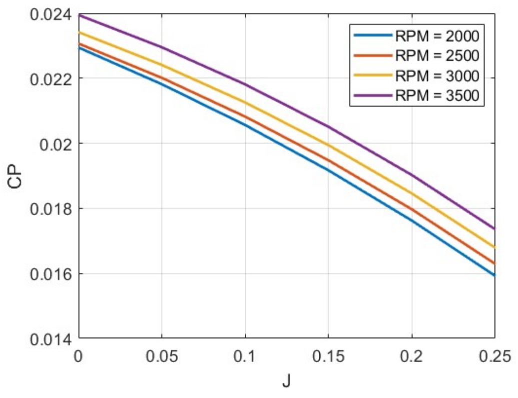


Figure 8.10: Non dimensional GPR

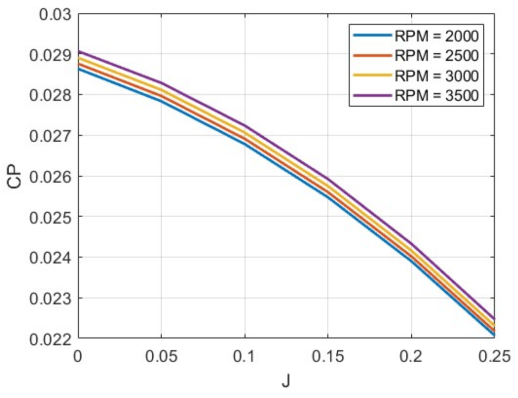


Figure 8.11: Dimensional SVM

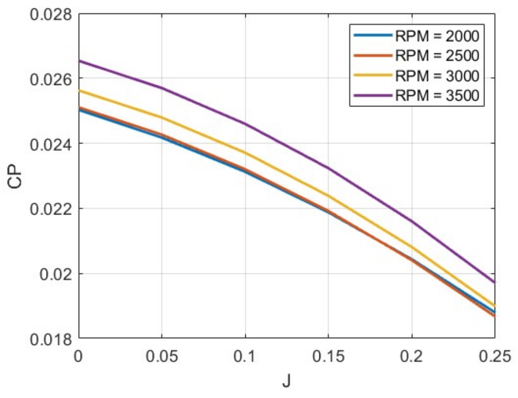


Figure 8.12: Non dimensional SVM

From Figure 8.7 it can be seen that even in this case, the best models are the GPRs although, if compared with the previous case, the best is the non dimensional GPR. This could be expected, as it has already been noted in Table 8.6. The problem with this model is that, as stated in that paragraph, it shows an excessive increase in CP with the RPM.

8.4.3 η trend

The efficiency trend (Equation (3.4)) is a direct consequence of the trends of CT and CP. All the models (Figures 8.15, 8.16, 8.17, and 8.18) show that efficiency increases with rising RPM, in line with the BEM results (Figure 8.14).

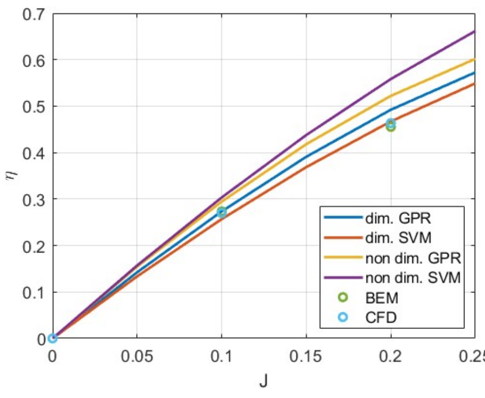


Figure 8.13: 3000 RPM analysis

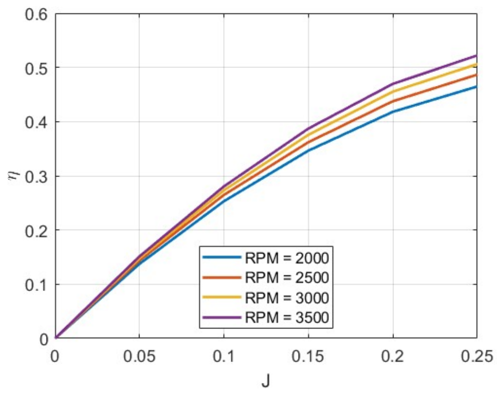


Figure 8.14: BEM

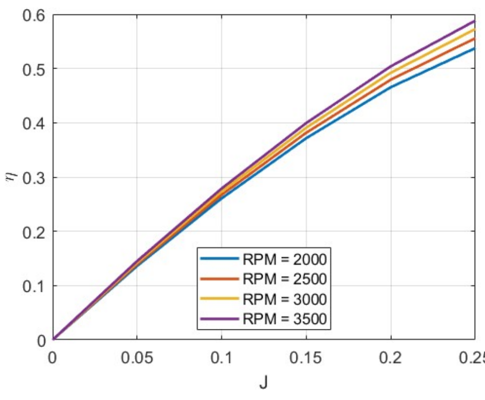


Figure 8.15: Dimensional GPR

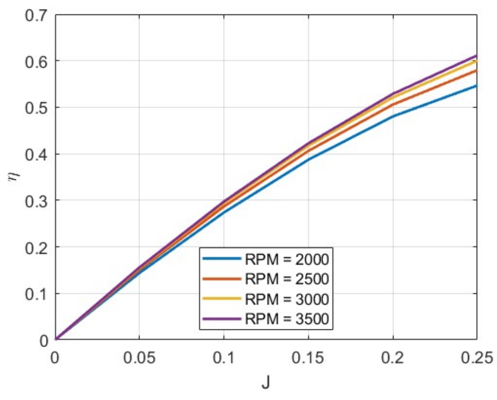


Figure 8.16: Non dimensional GPR

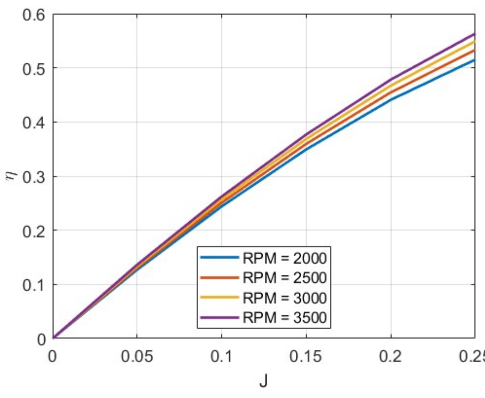


Figure 8.17: Dimensional SVM

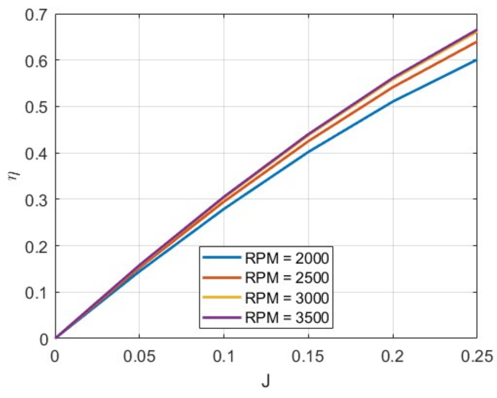


Figure 8.18: Non dimensional SVM

Figure 8.13 shows that efficiency is predicted very well by the models but, as J increases, an over-estimate appears because the models' prediction of CP decreases more sharply than in the BEM and CFD results.

8.5 Conclusions

This chapter has demonstrated the possibility of using ML models to analyse the performance of the T-Motor 15 X 5 CF rotor.
In particular, the static CT analysis showed that for the GPR models the mean percentage errors range from 2 % to 11 %, whereas for the SVM models the errors are larger. All the models exhibit a pronounced increase in CT with rising RPM, which makes the error grow; for this kind of analysis, the best model proves to be the dimensional dynamic GPR, because it shows a less marked rise in CT. It was also seen that, for the rotor considered, this model's errors are lower than those of the BEM. The static CP analysis, on the other hand, revealed that the various models did not manage to capture adequately how CP varies with RPM. In this case, the model with the lowest errors relative to the NASA values is the non dimensional dynamic GPR, but its errors increase sharply as the RPM rises. For this reason, the model to be chosen is still the dimensional dynamic GPR; indeed, although it has a higher mean error, it remains more or less constant as RPM varies. These two analyses also showed that using larger databases helps reducing the errors made by the models.
When examining the trends of CT and CP as J varies, it can be observed that all the models behave similarly in both cases. The trends of both CT and CP are in line with those of the BEM and CFD, and, even for these analyses, the best models prove to be the GPRs, particularly the dynamic GPR.

Chapter 9

Conclusions and Future Developments

This thesis has shown that simple ML algorithms can be used to predict rotor performance and significantly accelerate their preliminary design.
Of the various analyses performed with the models, optimisation and performance evaluation produced positive results, whereas rotor generation exposed the models' limitations. At present, it is not possible to use these models to generate rotors with precise performance targets because of the errors they introduce. These errors can stem from many causes: inaccuracies in the database, models' excessive simplicity for this type of analysis, limited coverage of the design space in the database, and more.
The results indicate that, among all the models examined, the dimensional dynamic GPR appears to produce the lowest errors with the available data.
From a computational standpoint, the advantage is clear: a single prediction takes only fractions of a second, whereas a CFD simulation can require hours or even days and a BEM analysis several seconds; the benefit becomes decisive when hundreds or thousands of different configurations must be evaluated. Nevertheless, the scalability of GPR remains an open issue because its cubic complexity with respect to the number of samples makes training progressively more demanding beyond a certain threshold.

In light of these results, future work should aim to improve model performance along two paths:

1. experiment with more complex and scalable learning models, such as deep neural networks, capable of capturing the non linear correlations between chord distribution, angle of incidence and aerodynamic response;

2. enhance and expand the database. This could involve purchasing and scanning

rotors to obtain more accurate geometric data and collecting additional information beyond what was used to train the ML models in this thesis. These improvements would help the models generalise better and enable the analysis of further cases, such as the effect of airfoil profile on performance.

In addition, new codes could be developed to perform analyses beyond those presented here, broadening the possible applications of the ML models.

Appendix A

Feature Selection Codes

This appendix contains the main MATLAB scripts developed for running and managing the feature selection process through a GA. The presented codes include:

- the fitness function used to evaluate each individual on the basis of predictive performance for the CT and CP targets;
- the callback function `@gaoutfun`, which allows the evolution of the number of selected features to be monitored over time;
- a script that automatically selects the most suitable features according to the chosen configuration (model and database type).

All scripts were designed to guarantee modularity, flexibility, and future extensibility.

A.1 Fitness function code

Listing A.1: fitnessFunction

```matlab
function error_tot = fitnessFunction_nuova(subset, dati, output_CT,
    output_CP, c, scelta_modello)

    selected_stations = find(subset == 1);
    if isempty(selected_stations)
        error_tot = 1e6;
        return;
    end

    selected_idx = [];
    for i = selected_stations
        selected_idx = [selected_idx, i, i + 18];
    end

    colonne_extra = 37:size(dati, 2);
    selected_idx = [selected_idx, colonne_extra];

    dati_reduced = dati(:, selected_idx);

    risultato_CT = modello_ML(dati_reduced, output_CT, c,
   scelta_modello);
    mse_CT = kfoldLoss(risultato_CT, 'LossFun', 'mse');
    error_CT = sqrt(mean(mse_CT));

    risultato_CP = modello_ML(dati_reduced, output_CP, c,
   scelta_modello);
    mse_CP = kfoldLoss(risultato_CP, 'LossFun', 'mse');
    error_CP = sqrt(mean(mse_CP));

    error_tot = error_CT / mean(output_CT) + error_CP / mean(
   output_CP);
end
```

The fitness function takes as input:

- the stations selected by the GA;
- the database data;
- the CT and CP outputs from the database;
- the database partitioning into folds for grouped cross-validation;
- the chosen ML model.

The first part of the code checks whether the vector of stations selected by the GA is empty and, in this case, it assigns a large error value[1]. Next, the database columns corresponding only to the stations selected by the GA are extracted, and a reduced dataset that is used to train the chosen ML model on both CT and CP is generated. Finally, the errors on CT and CP are combined using Equation (5.2), yielding the function's output.

[1]In the present case this step could be omitted, because the constraints guarantee a non zero number of stations. It has been included so that the code can also be used when no station constraints are imposed.

A.2 Code `@gaoutfun`

Listing A.2: @gaoutfun

```
function [state, options, optchanged] = gaoutfun(options, state, flag
    )
    persistent bestFeatureCounts bestGenerations;
    optchanged = false;
    if strcmp(flag, 'init')
        bestFeatureCounts = [];
        bestGenerations = [];
    elseif strcmp(flag, 'iter')
        [~, bestIdx] = min(state.Score);
        bestIndividual = state.Population(bestIdx, :);
        currentFeatureCount = sum(bestIndividual);
        bestFeatureCounts = [bestFeatureCounts; currentFeatureCount];
        bestGenerations = [bestGenerations; state.Generation];
        assignin('base', 'bestFeatureCounts', bestFeatureCounts);
        assignin('base', 'bestGenerations', bestGenerations);
    elseif strcmp(flag, 'done')
        figure;
        plot(bestGenerations, bestFeatureCounts, 'b-o', 'LineWidth',
    1.5);
        xlabel('Generation');
        ylabel('Number of Selected Stations');
        title('Evolution of Stations in the Best Solution');
        grid on;
    end
end
```

Without going into implementation details, at every generation the function saves the number of features used by the best individual and it plots the graph once the GA has finished its execution.

A.3 Automatic feature selection code

The following MATLAB script implements the logic required to automatically select the correct features based on the chosen model (GPR or SVM) and on the database type. This module is called within other scripts of the thesis to ensure consistency in feature selection during training and evaluation phases.

Listing A.3: Script for automatic feature selection

```
if strcmp(scelta_modello, 'GPR')
    if strcmp(scelta_stat_dinam, 'statico')
        if strcmp(scelta_dim_nondim, 'dimensionale')
            features_selection_geometria = [1 2 4 7 9 10 11 12 13 14
    16 17 18 19 20 22 25 27 28 29 30 31 32 34 35 36];
```

```
        elseif strcmp(scelta_dim_nondim, 'non dimensionale')
            features_selection_geometria = [1 2 4 7 9 10 11 12 13 14 16 17 18 19 20 22 25 27 28 29 30 31 32 34 35 36];
        end
    elseif strcmp(scelta_stat_dinam, 'dinamico')
        if strcmp(scelta_dim_nondim, 'dimensionale')
            features_selection_geometria = [1 2 4 7 9 10 11 12 13 14 16 17 18 19 20 22 25 27 28 29 30 31 32 34 35 36];
        elseif strcmp(scelta_dim_nondim, 'non dimensionale')
            features_selection_geometria = [1 2 4 7 9 10 11 12 13 14 16 17 18 19 20 22 25 27 28 29 30 31 32 34 35 36];
        end
    end
elseif strcmp(scelta_modello, 'SVM')
    if strcmp(scelta_stat_dinam, 'statico')
        if strcmp(scelta_dim_nondim, 'dimensionale')
            features_selection_geometria = [1 2 4 7 10 12 13 14 16 18 19 20 22 25 28 30 31 32 34 36];
        elseif strcmp(scelta_dim_nondim, 'non dimensionale')
            features_selection_geometria = [1 2 4 7 10 12 13 14 16 18 19 20 22 25 28 30 31 32 34 36];
        end
    elseif strcmp(scelta_stat_dinam, 'dinamico')
        if strcmp(scelta_dim_nondim, 'dimensionale')
            features_selection_geometria = [1 2 4 7 10 12 13 14 16 17 18 19 20 22 25 28 30 31 32 34 35 36];
        elseif strcmp(scelta_dim_nondim, 'non dimensionale')
            features_selection_geometria = [1 2 4 7 10 12 13 14 16 17 18 19 20 22 25 28 30 31 32 34 35 36];
        end
    end
end
% Creo vettore numerico (0 o 1) per i valori da 1 a 18
zeri_e_uni_corda = double(ismember(1:18, features_selection_geometria));
% Creo vettore numerico (0 o 1) per i valori da 19 a 36
zeri_e_uni_beta = double(ismember(19:36, features_selection_geometria));
% Trovo la posizione del valore 13 in features_selection_geometria
colonna_075 = find(features_selection_geometria == 13) % colonna di features_selection_geometria in cui si trova il 13 che sarebbe c/R_0.75
```

Inside each `if` block there are the geometry stations obtained through the feature selection procedure described in Chapter 5. Subsequently, vectors are created to count the number of stations of the chord and of β (identical in this case), and finally the position of the column containing the value of c/R at 75 % of the span is found, which is needed to compute the Reynolds number.

Appendix B

Codes for Rotors Generation

In this appendix the main MATLAB scripts used by the code described in Chapter 6 are reported. The presented codes includes:

- the inputs that the user can select to set the desired analysis;
- the fitness functions used to evaluate the rotors within the GA;
- the constraint functions applied to the GA.

B.1 Input

The code lists all the choices the user can make to have the ML model to generate the desired rotor. The options are described in detail in Section 6.2.

Listing B.1: Input

```
% Scegliere che tipo di analisi fare 'pareto' o 'diretta' (tutto
    minuscolo)
scelta1 = 'pareto'
% Scegliere il database che deve utilizzare 'statico' o 'dinamico' (
    tutto minuscolo)
scelta_stat_dinam = 'statico'
% Scegliere se si vuole utilizzare un modello dimensionale o non
    dimensionale
% 'dimensionale' o 'non dimensionale'
scelta_dim_nondim = 'non dimensionale'
% Scegliere il modello di machine learning da utilizzare. 'GPR' o '
    SVM'
scelta_modello = 'SVM'
% Scegliere se plottare l'andamento di CT, CP e eta con J. Valido
    solo SE
% scelta2 = 'dinamico'. 'si' o 'no'
```

```
grafici = 'si';
vettore_J = 0:0.1:0.8; % Definisce l'intervallo in cui costruire i
    grafici
% Scegliere se dopo il GA ottimizzare localmente 'si' o 'no'. VALIDO
    SOLO SE scelta1 = 'diretta'
ottimizzazione_locale_GA = 'si';
% Se ottimizzazione_locale_GA = 'si' scegliere come ottimizzare '
    fmincon' o
% 'patternsearch'
ottimizzazione_locale_GA_funzione = 'fmincon';
% Scegliere se dopo il GA multiobiettivo ottimizzare localmente 'si'
    o 'no'. VALIDO SOLO SE scelta1 = 'pareto'
ottimizzazione_locale_GA_multiobiettivo = 'no';
% Popolazione Elite SE scelta1 = 'diretta'
Elite = 1;
% Velocità del suono e Mach massimo al tip
mach_massimo_al_tip = 0.3;
% Valori target. Mettere zero al target che non si vuole imporre
% Valori su prestazioni
target_CT = 0.15;
target_CP = 0;
% Valori geometria e condizioni operative
target_raggio = 0;
target_omega = 0;
target_J = 0; % mettere -1 se non si vuole questo vincolo. VINCOLO
    VALIDO SOLO SE scelta2 = 'dinamico'
% Tolleranze sui target (scostamento massimo dal valore voluto).
    Mettere zero se si vogliono i valori precisi
tolleranza_raggio = 0.0001;
tolleranza_omega = 5;
tolleranza_J = 0;
tolleranza_CT = 0; % SE scelta1 = 'diretta' Valore in percentuale
    sennò è assoluto
tolleranza_CP = 0; % SE scelta1 = 'diretta' Valore in percentuale
% Definizione grandezza popolazione GA (numero più alto maggiore
    accuratezza ma maggior tempo di analisi)
grandezza_popolazione = 40000;
numero_generazioni = 500000; % massimo numero di generazioni per il
    GA
% Imposta il profilo con cui costruire il rotore
Naca = '4412'
% Definisco pesi. SE scelta1 = 'diretta'
% Più è alto più si forza il codice a cercare soluzioni migliori (
    avvicina le soluzioni al fronte di Pareto)
peso_CT_CP = 9999;
```

B.2 Fitness function – `'pareto'`

This fitness function evaluates the performance of the rotors generated by the multi-objective GA. If the tip Mach or CP constraints are violated, an infinite penalty is applied to force the GA to discard the rotor.

Listing B.2: Fitness function 'pareto'

```
function CT_CP = stima_CT_CP(vettore_caratteristiche, Modello_ML_CT,
    Modello_ML_CP, densita_aria_SL, viscosita_dinamica_aria_SL,
    velocita_del_suono_SL, mach_massimo_al_tip, scelta_dim_nondim,
    colonna_075)
     penalita = 0;
     if strcmp(scelta_dim_nondim, 'non dimensionale')
         raggio = vettore_caratteristiche(end);
         raggio_75 = raggio*0.75;
         rpm_rad_sec = vettore_caratteristiche(end-1);
         velocita_75 = rpm_rad_sec*raggio_75;
         corda_75 = vettore_caratteristiche(colonna_075)*raggio;
         vettore_caratteristiche(end) = (densita_aria_SL*velocita_75*
    corda_75)/viscosita_dinamica_aria_SL;
         vettore_caratteristiche(end-1) = (rpm_rad_sec*raggio)/
    velocita_del_suono_SL;
         if vettore_caratteristiche(end-1) > mach_massimo_al_tip
             penalita = Inf;
         end
     elseif strcmp(scelta_dim_nondim, 'dimensionale')
         if (vettore_caratteristiche(end-1)*vettore_caratteristiche(
    end))/velocita_del_suono_SL > mach_massimo_al_tip
             penalita = Inf;
         end
     end
     CT = Modello_ML_CT.predictFcn(vettore_caratteristiche);
     CP = Modello_ML_CP.predictFcn(vettore_caratteristiche);
     if CP <= 0
         penalita = Inf;
     end
     CT_CP = [-CT + penalita, CP + penalita];
end
```

B.3 Constraint functions – `'pareto'`

The combination of the two functions reported below allows the application of non linear constraints in the use of the GA. By analysing the values in the database, the admissible variation intervals are extracted and applied as constraints.

Listing B.3: Function to compute intervals from database

```
function [massimi_corda, massimi_beta] = funzione_massimi_database(
    dati, zeri_e_uni_corda, zeri_e_uni_beta)
    numRotori = size(dati,1); % Numero di rotori presenti nel
    database
    Nc = sum(zeri_e_uni_corda); % Numero di componenti che descrivono
     la corda
    Nb = sum(zeri_e_uni_beta);  % Numero di componenti che descrivono
     il beta
    % Inizializzo: [min, max] per ogni differenza tra stazioni
    massimi_corda = zeros(Nc-1, 2);
    massimi_beta = zeros(Nb-1, 2);
    for j = 1:(Nc-1)
        diffs_c_j = zeros(numRotori, 1);
        for i = 1:numRotori
            c = dati(i, 1:Nc);
            diffs_c_j(i) = c(j+1) - c(j);  % Differenza reale
        end
        massimi_corda(j,1) = min(diffs_c_j);  % Limite inferiore
        massimi_corda(j,2) = max(diffs_c_j);  % Limite superiore
    end
    for j = 1:(Nb-1)
        diffs_b_j = zeros(numRotori, 1);
        for i = 1:numRotori
            b = dati(i, (Nc+1):(Nc+Nb));
            diffs_b_j(i) = b(j+1) - b(j);  % Differenza reale
        end
        massimi_beta(j,1) = min(diffs_b_j);
        massimi_beta(j,2) = max(diffs_b_j);
    end
end
```

Listing B.4: Non linear constraint function

```
function [c, ceq] = nonlcon_funzione_massimi(x, massimi_corda,
    massimi_beta)
    % massimi_corda e massimi_beta sono matrici Nx2:
    % colonna 1 = limite minimo
    % colonna 2 = limite massimo
    Nc = size(massimi_corda, 1) + 1;
    Nb = size(massimi_beta, 1) + 1;
    corda = x(1:Nc);
    beta = x(Nc+1:Nc+Nb);
    diff_c = diff(corda);
    diff_b = diff(beta);
    minDiffC_lim = massimi_corda(:,1);
    maxDiffC_lim = massimi_corda(:,2);
    minDiffB_lim = massimi_beta(:,1);
    maxDiffB_lim = massimi_beta(:,2);
    c = [diff_c' - maxDiffC_lim;    % diff_c <= maxDiffC_lim
```

```
          -diff_c' + minDiffC_lim;    % diff_c >= minDiffC_lim
           diff_b' - maxDiffB_lim;
          -diff_b' + minDiffB_lim];
    ceq = [];
end
```

B.4 Fitness function - 'diretta'

This fitness function evaluates rotor performance by how close their CT or CP is to the set target. If no target is set, the GA seeks solutions nearer to the Pareto front improving both CT and CP. The formulas are those presented in Section 6.4.1.

Listing B.5: Fitness function 'diretta'

```
function valore = punteggio(x, target_CT, Modello_ML_CT, peso_CT, target_CP, Modello_ML_CP, peso_CP, media_output, densita_aria_SL, viscosita_dinamica_aria_SL, mach_massimo_al_tip, velocita_del_suono_SL, scelta_dim_nondim, colonna_075)
    valore = 0;
    if strcmp(scelta_dim_nondim, 'non dimensionale')
        raggio = x(end);
        raggio_75 = raggio*0.75;
        rpm_rad_sec = x(end-1);
        velocita_75 = rpm_rad_sec*raggio_75;
        corda_75 = x(colonna_075)*raggio;
        x(end) = (densita_aria_SL*velocita_75*corda_75)/viscosita_dinamica_aria_SL;
        x(end-1) = (rpm_rad_sec*raggio)/velocita_del_suono_SL;
        if x(end-1) > mach_massimo_al_tip
            valore = Inf;
        end
    elseif strcmp(scelta_dim_nondim, 'dimensionale')
        if (x(end-1)*x(end))/velocita_del_suono_SL > mach_massimo_al_tip
            valore = Inf;
        end
    end
    if target_CT > 0
        valore = valore + abs(Modello_ML_CT.predictFcn(x) - target_CT)/target_CT;
    else
        valore = valore - (Modello_ML_CT.predictFcn(x)/media_output(1))*peso_CT;
    end
    if target_CP > 0
        valore = valore + abs(Modello_ML_CP.predictFcn(x) - target_CP)/target_CP;
    else
        valore = valore + (Modello_ML_CP.predictFcn(x)/media_output(2))*peso_CP;
    end
end
```

B.5 Constraint function – '**diretta**'

This function applies constraints considering both performance targets and station to station variations from the database.

Listing B.6: Constraint function 'diretta'

```
function [c, ceq] = vincoli(x, target_CT, target_CP, Modello_ML_CT, Modello_ML_CP, tolleranza_CT, tolleranza_CP, densita_aria_SL, viscosita_dinamica_aria_SL, velocita_del_suono_SL, scelta_dim_nondim, colonna_075, massimi_corda, massimi_beta)
    numero_target_non_nulli = nnz([target_CT, target_CP]);
    c = zeros(numero_target_non_nulli, 1);
    if strcmp(scelta_dim_nondim, 'non dimensionale')
        raggio = x(end);
        raggio_75 = raggio*0.75;
        rpm_rad_sec = x(end-1);
        velocita_75 = rpm_rad_sec*raggio_75;
        corda_75 = x(colonna_075)*raggio;
        x(end) = (densita_aria_SL*velocita_75*corda_75)/viscosita_dinamica_aria_SL;
        x(end-1) = (rpm_rad_sec*raggio)/velocita_del_suono_SL;
    end
    conteggio = 1;
    if target_CT > 0
        CT_val = Modello_ML_CT.predictFcn(x);
        c(conteggio) = abs(CT_val - target_CT)/target_CT - tolleranza_CT;
        conteggio = conteggio + 1;
    end
    if target_CP > 0
        CP_val = Modello_ML_CP.predictFcn(x);
        c(conteggio) = abs(CP_val - target_CP)/target_CP - tolleranza_CP;
        conteggio = conteggio + 1;
    end
    [c_massimi, ~] = nonlcon_funzione_massimi(x, massimi_corda, massimi_beta);
    c = [c; c_massimi];
    ceq = [];
end
```

Appendix C

Codes for Rotors Optimization

In this appendix, the main MATLAB scripts used in the implementation of the code described in Chapter 7 are presented. The scripts shown include:

- the inputs that the user can select to set up the desired analysis;
- the fitness function used to evaluate the rotors within the GA;
- the constraint function applied to the GA.

C.1 Input

This script lists all the various choices that the user can make to perform the desired analysis. The options have been partly described in Section 7.4 and the remaining part in Section 6.2.

Listing C.1: Input

```
% Scegliere il database che deve utilizzare 'statico' o 'dinamico' (
    tutto minuscolo)
scelta_stat_dinam = 'dinamico'

% Scegliere il funzionamento del codice ('costruzione' o 'pareto')
scelta2 = 'pareto';

% Scegliere se si vuole utilizzare un modello dimensionale o non
    dimensionale
% 'dimensionale' o 'non dimensionale'
scelta_dim_nondim = 'dimensionale'
```

```

% Scegliere il modello di machine learning da utilizzare. 'GPR' o '
    SVM'
scelta_modello = 'GPR'

% Scegliere se plottare l'andamento di CT, CP e eta con J. Valido
    solo SE
% scelta1 = 'dinamico' e scelta2 = 'costruzione'. 'si' o 'no'
grafici = 'si';
vettore_J = 0:0.1:0.8; % Definisce l'intervallo in cui costruire i
    grafici

% Scegliere se dopo il GA ottimizzare localmente 'si' o 'no'. VALIDO
    SOLO SE scelta2 = 'costruzione'
ottimizzazione_locale_GA = 'si';
% Se ottimizzazione_locale_GA = 'si' scegliere come ottimizzare '
    fmincon' o
% 'patternsearch'
ottimizzazione_locale_GA_funzione = 'fmincon';

% Scegliere se dopo il GA multiobiettivo ottimizzare localmente 'si'
    o 'no'. VALIDO SOLO SE scelta2 = 'pareto'
ottimizzazione_locale_GA_multiobiettivo = 'no';

% Mettere 0 al target che non si vuole migliorare, mettere 1 al
    target che
% si vuole migliorare SOLO SE scelta2 = 'costruzione'
migliorare_CT = 1;
migliorare_CP = 0;

% Definire le tolleranze
tolleranza_CT = 0; % tolleranza assoluta
tolleranza_CP = 0; % tolleranza assoluta

% Definizione grandezza popolazione GA (numero più alto maggiore
    accuratezza ma maggior tempo di analisi)
grandezza_popolazione = 100;
numero_generazioni = 20000; % massimo numero di generazioni per il GA
% SOLO SE scelta3 = 'pareto';
grandezza_popolazione_pareto = 20000;

% Popolazione Elite SE scelta2 = 'costruzione'
Elite = 1;

% Tolleranza sulla geometria in percentuale. SE scelta2 = '
    costruzione'
% solo un valore. SE scelta2 = 'pareto' può essere un vettore per
    poter valutare più casistiche insieme.
tolleranze = [0.05, 0.2, 0.5, 0.9];
```

```
% Definire la tolleranza sulla geometria per costruire il fronte di
    pareto.
% SOLO UN NUMERO
tolleranza_pareto = 1;

% Imposta il profilo con cui costruire il rotore
Naca = '4412';

% Più è alto più si forza il codice a cercare soluzioni migliori (
    avvicina le soluzioni al fronte di Pareto)
peso = 9999;
```

C.2 Fitness function

The following fitness function evaluates the performance of the rotors generated by the GA by analysing the values of CT and CP.

Listing C.2: Fitness function

```
function valore = punteggio(x, migliorare_CT, migliorare_CP, rotore,
    Modello_ML_CT, Modello_ML_CP, peso, densita_aria_SL,
    viscosita_dinamica_aria_SL, velocita_del_suono_SL,
    scelta_dim_nondim, colonna_075)
    valore = 0;

    if strcmp(scelta_dim_nondim, 'non dimensionale')
        raggio = x(end);
        raggio_75 = raggio*0.75;
        rpm_rad_sec = x(end-1);
        velocita_75 = rpm_rad_sec*raggio_75;
        corda_75 = x(colonna_075)*raggio;
        x(end) = (densita_aria_SL*velocita_75*corda_75)/
    viscosita_dinamica_aria_SL;
        x(end-1) = (rpm_rad_sec*raggio)/velocita_del_suono_SL;

        raggio = rotore(end);
        raggio_75 = raggio*0.75;
        rpm_rad_sec = rotore(end-1);
        velocita_75 = rpm_rad_sec*raggio_75;
        corda_75 = rotore(colonna_075)*raggio;
        rotore(end) = (densita_aria_SL*velocita_75*corda_75)/
    viscosita_dinamica_aria_SL;
        rotore(end-1) = (rpm_rad_sec*raggio)/velocita_del_suono_SL;
    end

    CT_rotore = Modello_ML_CT.predictFcn(rotore);
    CP_rotore = Modello_ML_CP.predictFcn(rotore);

    if migliorare_CT == 1
        valore = valore + CT_rotore - Modello_ML_CT.predictFcn(x)/
CT_rotore*peso;
    elseif migliorare_CT == 0
        valore = valore + abs(CT_rotore - Modello_ML_CT.predictFcn(x)
)/CT_rotore*peso;
    end
    if migliorare_CP == 1
        valore = valore + CP_rotore + Modello_ML_CP.predictFcn(x)/
CP_rotore*peso;
    elseif migliorare_CP == 0
        valore = valore + abs(CP_rotore - Modello_ML_CP.predictFcn(x)
)/CP_rotore*peso;
    end
end
```

valore is calculated in different ways depending on whether improvement of a specific performance metric is desired:

- if improving CT is desired:

$$valore = valore + CT\_rotor - \frac{CT\_new\_rotor}{CT\_rotor} \cdot peso$$

- if improving CT is not desired:

$$valore = valore + \frac{|CT\_rotor - CT\_new\_rotor|}{CT\_rotor} \cdot peso$$

- if improving CP is desired:

$$valore = valore + CP\_rotor + \frac{CP\_new\_rotor}{CP\_rotor} \cdot peso$$

- if improving CP is not desired:

$$valore = valore + \frac{|CP\_rotor - CP\_new\_rotor|}{CP\_rotor} \cdot peso$$

Rotors characterized by a lower *valore* are the best.

C.3 Constraint function

The function applies the constraints, taking into account both the desired performance improvement (CT or CP) and the variations among the stations present in the database.

Listing C.3: Constraint function

```
function [c, ceq] = vincoli(x, rotore, migliorare_CT, migliorare_CP, Modello_ML_CT, Modello_ML_CP, tolleranza_CT, tolleranza_CP, densita_aria_SL, viscosita_dinamica_aria_SL, velocita_del_suono_SL, scelta_dim_nondim, colonna_075, massimi_corda, massimi_beta)
    numero_target_nulli = sum([migliorare_CT, migliorare_CP] == 0);
    c = zeros(numero_target_nulli, 1);

    if strcmp(scelta_dim_nondim, 'non dimensionale')
        raggio = x(end);
        raggio_75 = raggio*0.75;
        rpm_rad_sec = x(end-1);
        velocita_75 = rpm_rad_sec*raggio_75;
        corda_75 = x(colonna_075)*raggio;
        x(end) = (densita_aria_SL*velocita_75*corda_75)/viscosita_dinamica_aria_SL;
        x(end-1) = (rpm_rad_sec*raggio)/velocita_del_suono_SL;
        raggio = rotore(end);
        raggio_75 = raggio*0.75;
        rpm_rad_sec = rotore(end-1);
        velocita_75 = rpm_rad_sec*raggio_75;
        corda_75 = rotore(colonna_075)*raggio;
        rotore(end) = (densita_aria_SL*velocita_75*corda_75)/viscosita_dinamica_aria_SL;
        rotore(end-1) = (rpm_rad_sec*raggio)/velocita_del_suono_SL;
    end

    conteggio = 1;
    if migliorare_CT == 0
        CT_rotore = Modello_ML_CT.predictFcn(rotore);
        CT_val = Modello_ML_CT.predictFcn(x);
        c(conteggio) = abs(CT_val - CT_rotore) - tolleranza_CT;
        conteggio = conteggio + 1;
    end
    if migliorare_CP == 0
        CP_rotore = Modello_ML_CP.predictFcn(rotore);
        CP_val = Modello_ML_CP.predictFcn(x);
        c(conteggio) = abs(CP_val - CP_rotore) - tolleranza_CP;
    end

    [c_massimi, ~] = nonlcon_funzione_massimi(x, massimi_corda, massimi_beta);
    % Appendi i nuovi vincoli a quelli precedenti
    c = [c; c_massimi];
    % Vincoli di uguaglianza (ceq = 0)
    ceq = [];
end
```

Bibliography

[1] Arthur L. Samuel. «Some Studies in Machine Learning Using the Game of Checkers». In: *IBM Journal of Research and Development* 3.3 (1959), pp. 210–229 (cit. on p. 3).

[2] Michael Chen. *Cos'è il Machine Learning?* `https://www.oracle.com/it/artificial-intelligence/machine-learning/what-is-machine-learning/` (cit. on p. 4).

[3] LUM School of Management. *Machine Learning: cos'è e come funziona.* `https://www.lum.it/machine-learning/` (cit. on p. 4).

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006 (cit. on pp. 4, 27, 28, 30).

[5] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective.* MIT Press, 2012 (cit. on pp. 4, 28).

[6] Jacob Murel and Eda Kavlakoglu. *Cos'è l'apprendimento per rinforzo?* `https://www.ibm.com/it-it/think/topics/reinforcement-learning` (cit. on p. 4).

[7] MathWorks. *Che cos'è il Machine Learning?* `https://it.mathworks.com/discovery/machine-learning.html` (cit. on pp. 5, 6).

[8] IBM. *Cos'è l'apprendimento supervisionato?* `https://www.ibm.com/it-it/topics/supervised-learning` (cit. on p. 6).

[9] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning.* Springer, 2009 (cit. on p. 6).

[10] MathWorks. *Overfitting.* `https://it.mathworks.com/discovery/overfitting.html` (cit. on p. 6).

[11] Pedro Domingos. «A few useful things to know about machine learning». In: *Communications of the ACM* 55.10 (2012), pp. 78–87 (cit. on p. 7).

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016 (cit. on pp. 7, 32).

[13] Tim Mucci. *Overfitting vs. underfitting.* `https://www.ibm.com/it-it/think/topics/overfitting-vs-underfitting` (cit. on p. 8).

[14] Ron Kohavi. «A study of cross-validation and bootstrap for accuracy estimation and model selection». In: *IJCAI.* 1995, pp. 1137–1145 (cit. on p. 7).

[15] Isabelle Guyon and André Elisseeff. «An introduction to variable and feature selection». In: *Journal of machine learning research* 3 (2003), pp. 1157–1182 (cit. on pp. 8, 38).

[16] Rafael Cruz and David S. Wishart. «Towards a standard process for machine learning in industry». In: *Proceedings of the 2016 International Conference on Data Science and Advanced Analytics (DSAA).* IEEE. 2016, pp. 134–143 (cit. on p. 9).

[17] J.B. Brandt, R.W. Deters, G.K. Ananda, O.D. Dantsker, and M.S. Selig. *UIUC Propeller Database, Vols 1-4.* `https://m-selig.ae.illinois.edu/props/propDB.html`. 2024 (cit. on p. 15).

[18] APC Propellers. *APC Propellers Technical Information - File Downloads.* `https://www.apcprop.com/technical-information/file-downloads/?v=cd32106bcb6d` (cit. on pp. 17, 18).

[19] MathWorks. *Regression Learner App.* `https://it.mathworks.com/help/stats/regression-learner-app.html` (cit. on pp. 25, 27).

[20] MathWorks. *Regression Learner.* `https://it.mathworks.com/help/stats/regressionlearner-app.html` (cit. on p. 26).

[21] Ali Rahimi and Benjamin Recht. «Random features for large-scale kernel machines». In: *Advances in Neural Information Processing Systems (NeurIPS).* 2007, pp. 1177–1184 (cit. on p. 31).

[22] Leo Breiman. «Random forests». In: *Machine Learning* 45.1 (2001), pp. 5–32 (cit. on p. 31).

[23] Jianchang Yang and Vasant Honavar. «Feature subset selection using a genetic algorithm». In: *IEEE Intelligent Systems* 13.2 (1998), pp. 44–49 (cit. on p. 38).

[24] Bing Xue, Mengjie Zhang, Will N. Browne, and Xin Yao. «A survey on evolutionary computation approaches to feature selection». In: *IEEE Transactions on Evolutionary Computation* 20.4 (2016), pp. 606–626 (cit. on p. 38).

[25] Melanie Mitchell. *An Introduction to Genetic Algorithms.* MIT press, 1998 (cit. on p. 38).

[26] Witold Siedlecki and Joseph Sklansky. «A note on genetic algorithms for large-scale feature selection». In: *Pattern Recognition Letters* 10.5 (1989), pp. 335–347 (cit. on p. 39).

[27] MathWorks. *Genetic Algorithm Options - MATLAB & Simulink*. `https://it.mathworks.com/help/gads/genetic-algorithm-options.html` (cit. on pp. 41, 54, 59).

[28] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms.* John Wiley & Sons, 2001 (cit. on p. 53).

[29] Manuel Carreno Ruiz and Domenic D'Ambrosio. «ROT8: A Matlab app for low Reynolds number airfoil and rotor aerodynamic design». In: *AIAA Aviation 2023 Forum*. 2023, p. 3379 (cit. on p. 61).

[30] M. Carreño Ruiz, M. Scanavino, D. D'Ambrosio, G. Guglieri, and A. Vilardi. «Experimental and numerical analysis of hovering multicopter performance in low-Reynolds number conditions». In: *Aerospace Science and Technology* (2022) (cit. on p. 62).

[31] Carl Russell, Gina Willink, Colin Theodore, Jaewoo Jung, and Brett Glasner. *Wind Tunnel and Hover Performance Test Results for Multicopter UAS Vehicles*. Tech. rep. NASA Ames Research Center, 2018 (cit. on pp. 75, 93, 94).