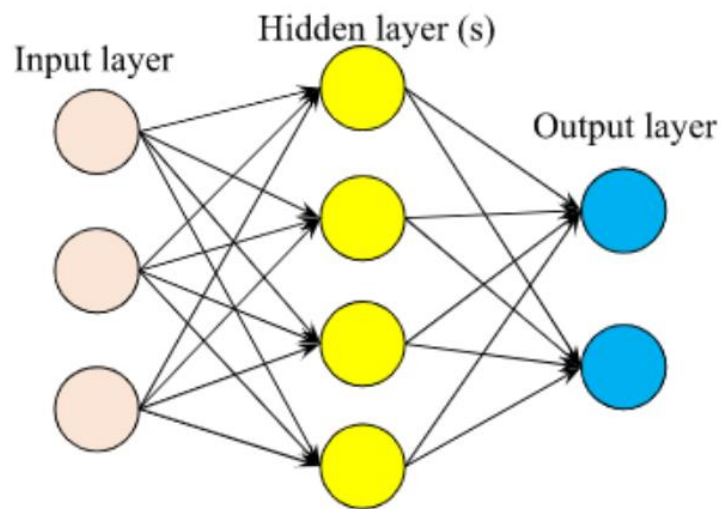


Christopher Ring-Tsingos Gulbrandsen
Sivert Risa Angaard

Exploring machine learning in the design of reinforced concrete beams.

Master's thesis in Civil and Environmental Engineering
Supervisor: Daniel Cantero
June 2023





MASTER'S THESIS 2023

SUBJECT AREA: Machine learning in concrete design	DATE: 11.06.2023	TOTAL NO. OF PAGES 161
--	---------------------	---------------------------

TITLE:

Exploring machine learning in the design of reinforced concrete beams

Utforsking av maskinl ring i dimensjonering av armerte betongbjelker

BY:



Christopher Ring-Tsingos Gulbrandsen



Sivert Risa Angaard

SUMMARY:

This master's thesis delves into the application of machine learning (ML) in the design of reinforced concrete structures, focusing on three key tasks: anticipating capacity failure mode, predicting load capacity, and cost optimization design. By employing ML classification techniques to predict failure modes, regression techniques to estimate structural capacities, and ML models to optimize cost in T-section design, this research reveals the potential of ML in addressing complex structural design tasks. A significant part of the study is the digitization of design rules from the European Standard EN 1992-1-1, which facilitates the creation of several comprehensive datasets for model training.

The findings highlight the potential of ML in enhancing efficiency and decision-making in structural engineering, while underscoring the need for cautious application given its impact on structural safety. The thesis emphasizes the weight of dataset characteristics on ML model performance and shows the importance of ML proficiency among practitioners. Further exploration and incorporation of ML into education and the long-term growth of the structural engineering field is encouraged.

Preface

This master's thesis represents the culmination of our studies, marking the final stage of our Master of Science Degree at the Norwegian University of Science and Technology (NTNU), within the department of structural engineering. The research and preparation were undertaken from January to June 11th, 2023.

Our academic journey has been profoundly shaped by our growing interest in structural engineering, with a particular focus on the domain of concrete. Throughout our bachelor's thesis, we both engaged with the structural design of concrete, continually seeking methods to enhance the efficiency of related calculations.

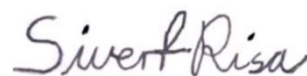
Our fascination was further fueled during a summer internship that introduced us to parametric modelling. This experience was reinforced by a course in parametric design, where artificial intelligence (AI) was introduced. Furthermore, we were part of the first cohort that had the opportunity to select machine learning for our preliminary project, allowing us to conduct an in-depth study of machine learning's application in structural engineering. This project provided us with our initial understanding of the transformative potential of machine learning.

As we delved deeper into this field, we became aware that we were among the first in our faculty to approach machine learning from this specific perspective. We also became increasingly aware of the skills we needed to develop further in order to effectively incorporate machine learning into structural engineering practices. Yet, given the significant rise in AI's popularity and its potential to reshape various industries, we were excited by the challenge. We saw an opportunity to explore machine learning as part of the day-to-day tasks of structural engineers, especially in concrete design.

We would like to express our sincere gratitude to our supervisor, Daniel Cantero, for believing in the project, and Muhammad Zohaib Sarwar, for providing valuable information regarding machine learning.



Christopher Ring-Tsingos Gulbrandsen



Sivert Risa Angaard

Abstract

In the era of digital transformation, Machine Learning (ML), a vital subset of Artificial Intelligence (AI), has found its way into a variety of fields, including structural engineering. This master's thesis explores the application and comparison of ML models in the design of reinforced concrete (RC) beams, with a focus on three distinct, yet interconnected tasks: predicting the anticipated capacity failure mode (Task 1), predicting capacities (Task 2), and cost optimization of T-section design (Task 3). These tasks serve as gateways to understanding classification, regression, and optimization techniques in ML, respectively.

Task 1 leverages ML classification techniques, providing an in-depth exploration of multiclass classification models in assessing structural integrity and safety. Task 2 delves into regression models, employing ML regression techniques for accurate structural capacity estimation in RC beams, demonstrating the potential of these models in augmenting conventional capacity prediction procedures. Task 3 employs a multivariate regression model trained on an optimized dataset for predicting cost-effective designs, underscoring the potential of ML in optimizing structural design. To execute these tasks, datasets were generated by digitally encoding design rules from the European Standard EN 1992-1-1 (EC2).

MLP demonstrated good results in tasks 1 and 3, while SVR stood out as the overall top performer in Task 2. Despite these promising findings across all tasks, the results were somewhat affected by data characteristics, such as class imbalance, multicollinearity, heteroscedasticity, and outliers, all of which impact the performance of ML models. Although we can confidently conclude on the best-fitting models for the tasks, future work should focus on improving these models over time. This could involve delving deeper into the challenges that we have identified, as well as incorporating new, unseen experimental data. Such efforts will enhance the utility of these ML models as practical tools for structural engineers.

This master's thesis, through a detailed literature review, the provision of comprehensive documentation including scripts related to the models and dataset generation, as well as the practical application of ML algorithms, provides a significant foundation for future exploration in the intersection of ML and structural engineering. It promotes a data-driven approach to understanding and designing structural systems and ensures that others can replicate and build upon our study. Lastly, it encourages the integration of ML in structural engineering practices and education, while highlighting the role of ML in addressing complex structural design tasks.

Sammendrag

I den digitale transformasjonens æra, har maskinlæring, et lovende underområde av kunstig intelligens, funnet veien inn i diverse fagfelt, inkludert bygg- og konstruksjonsteknikk. Denne masteroppgaven utforsker bruken og sammenligningen av maskinlæringsmodeller i dimensjonering av slakkarmerte betongbjelker, med fokus på tre distinkte, men sammenkoblede oppgaver: predikering av forventet kapasitetsoverskridelse (Oppgave 1), predikering av kapasiteter (Oppgave 2), og kostnadsoptimalisering av T-tverrsnitt (Oppgave 3). Oppgavene fungerer som inngangsporter for å forstå klassifisering, regresjon og optimaliseringsteknikker innen maskinlæring.

Oppgave 1 benytter seg av maskinlæringsklassifiseringsteknikker for å gi en grundig undersøkelse av multiklasse-klassifiseringsmodeller. Disse modellene blir brukt for å vurdere strukturell integritet og sikkerhet. Oppgave 2 fokuserer på regresjonsmodeller, og utforsker tilhørende maskinlæringsteknikker for å estimere kapasitet i armerte betongbjelker. Oppgaven illustrerer potensialet disse modellene har til å forsterke konvensjonelle metoder for estimering av kapasitet. Oppgave 3 bruker en multivariat regresjonsmodell trent på et optimalisert datasett for å predikere kostnadseffektive design. Dette understreker maskinlærings potensial i strukturell designoptimalisering. For å utføre oppgavene, ble datasett generert ved å digitalisere dimensjoneringsregler fra den Europeiske Standarden EN 1992-1-1 (EC2).

MLP modellen viste gode resultater i oppgavene 1 og 3, og i oppgave 2 ble SVR den best presterende modellen. Til tross for lovende resultater i alle oppgaver, ble resultatene noe påvirket av datakarakteristikker som klasseubalanse, multikollinearitet, heteroskedastisitet og avvikende observasjoner, som påvirket modellens ytelse. Til tross for at vi med sikkerhet kan identifisere modellene som var best egnet for de ulike oppgavene, bør fremtidig arbeid fokusere på å forbedre dem over tid. Dette kan innebære en dypere undersøkelse av de identifiserte utfordringene, samt trening på ny, usett eksperimentell data. Slike tiltak vil forbedre bruken av disse modellene som praktiske verktøy for fagfeltet bygg- og konstruksjonsteknikk.

Gjennom detaljert litteraturgjennomgang og praktisk anvendelse av maskinlæringsalgoritmer, gir denne masteroppgaven et betydelig grunnlag for fremtidig utforskning i skjæringspunktet mellom maskinlæring og konstruksjonsteknikk. Videre fremmer den en datadrevet tilnærming til forståelse og design av strukturelle systemer. Til slutt oppfordrer den til ytterligere integrasjon av maskinlæring i både praksis og utdanning innen konstruksjonsteknikk, samt understreker den viktige rollen maskinlæring vil få i å løse komplekse oppgaver i bransjen.

Content

MASTER'S THESIS 2023	I
Preface.....	II
Abstract	III
Sammendrag.....	IV
Content.....	V
List of Figures.....	IX
List of Tables.....	XI
1. Introduction.....	1
1.1. Background.....	1
1.2. Objective	1
1.3. Outline of this thesis	3
2. Theoretical Framework.....	4
2.1. Optimization.....	4
2.1.1. Trust-region method.....	5
2.2. Machine Learning.....	6
2.2.1. Learning processes	7
2.2.2. Classification.....	8
2.2.2.1. Decision Trees.....	8
2.2.2.2. K-Nearest Neighbors	11
2.2.3. Regression	15
2.2.3.1. Multiple Linear Regression.....	16
2.2.3.2. Support Vector Regression	17
2.2.4. Multilayer Perceptron	21
2.3. Hyperparameter tuning.....	28
2.4. Evaluation metrics.....	29
2.4.1. Evaluation metrics for classification algorithms.....	30
2.4.1.1. Confusion matrix.....	31

2.4.1.2.	Precision score.....	32
2.4.1.3.	Recall score.....	32
2.4.1.4.	Precision-Recall tradeoff.....	32
2.4.1.5.	F1-score	33
2.4.1.6.	Accuracy score	33
2.4.1.7.	Macro and Weighted -average.....	33
2.4.2.	Evaluation metrics for regression algorithms.....	34
2.4.2.1.	Mean Squared Error and Mean Absolute Error.....	34
2.4.2.2.	R-squared.....	35
2.4.2.3.	Graphical analysis.....	36
3.	Methodology	38
3.1.	Research design and overall approach	38
3.2.	Materials and software	39
3.3.	Reflection and quality assurance.....	40
3.3.1.	Validity	40
3.3.2.	Generalizability	41
3.3.3.	Replicability	42
4.	Dataset generation for RC beam design	44
4.1.	Limit state design of RC sections under bending.....	45
4.2.	Task 1 and 2: Rectangular RC beams	46
4.2.1.	Dataset generation for Task 1 and 2.....	51
4.3.	Task 3: RC T-section Optimization.....	53
4.3.1.	Dataset generation for Task 3	57
5.	Task 1: Predicting anticipated failure mode.....	58
5.1.	Data preprocessing.....	58
5.2.	Hyperparameter tuning.....	62
5.2.1.	DT.....	62
5.2.2.	kNN.....	64
5.2.3.	MLP.....	65

5.3.	Results.....	69
5.3.1.	Classification reports	69
5.3.2.	Confusion matrices	70
5.3.3.	Decision boundaries	71
5.4.	Results after balancing with under-sampling	72
5.4.1.	Confusion Matrices	73
5.5.	Discussions	74
6.	Task 2: Predicting capacities.....	80
6.1.	Data preprocessing.....	80
6.1.1.	Feature correlation	81
6.2.	Hyperparameter tuning.....	85
6.3.	Results.....	87
6.3.1.	Moment capacity prediction	87
6.3.2.	Shear capacity prediction.....	89
6.3.3.	Load capacity.....	90
6.4.	Discussions	92
7.	Task 3: Cost optimization.....	97
7.1.	Data preprocessing.....	97
7.1.1.	Dealing with outliers.....	99
7.1.2.	Feature correlation.....	101
7.2.	Hyperparameter tuning.....	102
7.3.	Results.....	104
7.3.1.	Results on the preliminary dataset.....	104
7.3.2.	Results after the removal of delta_lim and outliers.....	107
7.4.	Discussions	109
8.	Conclusions.....	112
8.1.	Limitations	114
8.2.	Future work.....	115
9.	References.....	116

10. Appendices 123

List of Figures

Figure 1: Objective visualized	3
Figure 2: Yearly distribution of ML-related articles in structural engineering [1]	6
Figure 3: Summary of types of learning	7
Figure 4: Visualization of a DT [14]	8
Figure 5: An example [26] of kNN with $k = 3$ (solid line circle), $k = 5$ (dotted line circle).....	12
Figure 6: Illustration of a best fitted line in linear regression [31]	16
Figure 7: Illustration of the principles of SVM [35]	18
Figure 8: Illustration of the principles of SVR [36].....	18
Figure 9: Illustration of how the kernel trick is used [36].....	20
Figure 10: Processing neuron [1]	21
Figure 11: Network architecture [1]	22
Figure 12: Plot of Inputs vs. Outputs for logistic Sigmoid. [45].....	25
Figure 13: Plot of Inputs vs. Outputs for reLu [45].....	26
Figure 14: Visualization of different cases of learning rates [50].....	27
Figure 15: k-fold cross validation [60]	29
Figure 16: Binary classification problem (2x2 matrix) [63].....	31
Figure 17: Example of multiclass classification problem (3x3 matrix)	31
Figure 18: Example of a predicted vs. actual value plot [69]	36
Figure 19: Visualization of a) ideal residual plot and b) suboptimal residual plot [71].....	37
Figure 20: Methodology flow chart.....	38
Figure 21: a) Rectangular and T-beam section; b) strains at ultimate limit state and c) stresses at ultimate limit state [76].	45
Figure 22: Class distribution in the data set.....	58
Figure 23: Scatterplot of feature pairs.....	59
Figure 24: Linear and Nonlinear boundaries [82].....	60
Figure 25: Plot of the learning curves over max depth.....	63
Figure 26: Plot of misclassification error over k	65
Figure 27: Signs of overfitting [83]	67
Figure 28: Training and Validation Metrics over Epochs	67
Figure 29: Impact of Learning Rates on Accuracy over Epochs.....	68
Figure 30: Decision boundaries for two pairs of features for a)kNN; b)MLP; c)DT	72
Figure 31: Relationship between the independent variables and the moment capacity	82
Figure 32: Relationship between the independent variables and the shear capacity	82
Figure 33: Relationship between the independent variables and the load capacity.....	82

Figure 34: Correlation matrix for the independent variables and the moment capacity.....	83
Figure 35: Correlation matrix for the independent variables and the shear capacity.....	84
Figure 36: Correlation matrix for the independent variables and the load capacity	84
Figure 37: Learning curves for a) Moment capacity, b) Shear capacity, c) Load capacity	87
Figure 38: Illustration of the actual values and the predicted moment capacities for both the MLR and the SVR model	88
Figure 39: Residual plots for the moment capacity predictions for the MLR model.....	88
Figure 40: Residual plots for the moment capacity predictions for the SVR model.....	89
Figure 41: Illustration of the actual values and the predicted shear capacities for both the MLR and the SVR model.....	89
Figure 42: Residual plots for the shear capacity predictions for the MLR model	90
Figure 43: Residual plots for the shear capacity predictions for the SVR model.....	90
Figure 44: Illustration of the actual values and the predicted load capacities for both the MLR and the SVR model.....	91
Figure 45: Residual plots for the load capacity predictions for the MLR model.....	91
Figure 46: Residual plots for the load capacity predictions for the SVR model	91
Figure 47: Frequency and value of α -parameter and successful optimization convergences	97
Figure 48: Frequency and value of hf -parameter in dataset.....	99
Figure 49: a)Summary; b)Outliers that are outside of the upper and lower quartiles by 1.5 times the interquartile range [87]	100
Figure 50: Box plots showing outliers for the entire dataset.....	100
Figure 51: Correlation matrix.....	101
Figure 52: Loss curves over Epochs.....	103
Figure 53: Predicted vs. Actual and residual plot for b.....	105
Figure 54: Predicted vs. Actual and residual plot for bw.....	105
Figure 55: Predicted vs. Actual and residual plot for h.....	105
Figure 56: Predicted vs. Actual and residual plot for d.....	105
Figure 57: Predicted vs. Actual and residual plot for As	106
Figure 58: Predicted vs. Actual and residual plot for hf.....	106
Figure 59: Predicted vs. Actual and residual plot for cost.....	106
Figure 60: Predicted vs. Actual and residual plot for b after preprocessing	107
Figure 61: Predicted vs. Actual and residual plot for bw after preprocessing.....	107
Figure 62: Predicted vs. Actual and residual plot for h after preprocessing	107
Figure 63: Predicted vs. Actual and residual plot for d after preprocessing	108
Figure 64: Predicted vs. Actual and residual plot for As after preprocessing.....	108
Figure 65: Predicted vs. Actual and residual plot for cost after preprocessing.....	108

Figure 66: Predicted vs. Actual and residual plot for hf after preprocessing..... 108

List of Tables

Table 1: Overview of the independent variables' upper and lower limits and their ranges for the datasets in Task 1 and 2.....	51
Table 2: Definition of design variables	53
Table 3: KNeighborsClassifier Classification Report	69
Table 4: DecisionTreeClassifier Classification Report	70
Table 5: MLPClassifier Classification Report	70
Table 6: KNeighborsClassifier Confusion Matrix.....	71
Table 7: DecisionTreeClassifier Confusion Matrix.....	71
Table 8: MLPClassifier Confusion Matrix.....	71
Table 9: Sampling strategies for RUS	73
Table 10: KNeighborsClassifier Confusion Matrix after RUS	73
Table 11: DecisionTreeClassifier Confusion Matrix after RUS.....	73
Table 12: MLPClassifier Confusion Matrix after RUS	74
Table 13: Values of different measurements of the model's accuracy in predicting moment capacity.....	88
Table 14: Values of different measurements of the model's accuracy in predicting shear capacity	89
Table 15: Values of different measurements of the model's accuracy in predicting moment capacity.....	90
Table 16: Evaluation metrics of MLP.....	106
Table 17: Evaluation of MLP after preprocessing.....	109

1. Introduction

1.1. Background

Traditional structural analysis and design methods can be time-consuming and overly complex for practical implementation, especially when dealing with nonlinear systems. Herein lies the appeal of Machine Learning (ML): it offers a promising alternative that can save significant time and effort.

ML, widely recognized as the most successful branch of Artificial Intelligence (AI), has been making waves across various fields, and structural engineering is no exception [1]. This progress is driven by recent advances in ML techniques, improved computational capabilities, and the availability of large datasets. This growing interest signals a shift towards a more data-driven approach to understanding and designing structural systems. However, despite this increase, ML remains somewhat enigmatic to many structural engineering students, largely due to limited exposure and understanding. This thesis is presented as a springboard into this exciting intersection between ML and structural engineering.

1.2. Objective

We aim to delve into three distinct, but interrelated tasks within the sphere of Reinforced Concrete (RC) beam design. These tasks, while relatively straightforward from a structural engineering perspective, provide a valuable opportunity to explore the intricacies of various ML models and techniques.

Anticipating capacity failure mode (Task 1): This task introduces the concept of classification, a central technique in ML. It leverages multiclass classification techniques to understand and predict the anticipated failure mode among moment, shear and deflection in RC beams. While the concept of anticipating failure modes is a fundamental aspect of structural engineering, applying ML to this task allows for a deeper understanding of classification models and their applications. This task addresses the need for identifying and predicting potential failure modes in RC structures, which is a vital aspect of structural integrity and safety assessment.

Predicting moment, shear, and load capacity (Task 2): This task ventures into the realm of regression, another fundamental ML technique. It employs ML regression techniques to accurately estimate the structural capacities of RC beams. While load capacity prediction is a

standard procedure in structural engineering, using ML for this task provides an opportunity to delve into regression models and their nuances. It focuses on the accurate estimation of structural capacities - a key step in the design process that ensures structures are not only safe, but also efficiently designed to manage anticipated loads.

To facilitate the learning process of the first two tasks, the design rules from the European Standard EN 1992-1-1 (EC2) [2] concerning moment, shear, and deflection capacities of RC structures have been digitally encoded using Python. This coding effort has enabled the generation of a comprehensive dataset, capturing a wide range of design scenarios, upon which the ML models can train. By translating these design rules into a machine-readable format, we have not only produced a valuable resource for training ML models, but also opened a path for automated design checks and optimizations, aligning with the digital transformation in the field of structural engineering.

Cost optimization of T-Section design (Task 3): This task brings us into the area of optimization, a primary goal in structural engineering. To ensure the dataset for Task 3 contains optimized sections for the ML model to train on, an optimization process is performed based on the EC2 design rules and current rules of practice. The ML model employed here is a multivariate regression model, which is trained to predict cost-effective designs. This approach ensures that the design is not only structurally sound and capable of withstanding the anticipated loads, but is also cost-effective. By applying ML to this task, we can explore how ML can be used for optimization problems, specifically in the context of cost optimization in structural engineering.

In essence, each of these tasks, while rooted in the practicalities of structural engineering, serves as a gateway to a distinct aspect of ML. This approach enables a comprehensive exploration and comparison of ML techniques within a practical and relevant context. It not only strives to identify the best suited models for the given tasks, but also aims to demonstrate the potential of ML to enhance structural engineering. The objective and the tasks that drive this investigation are summarized below:

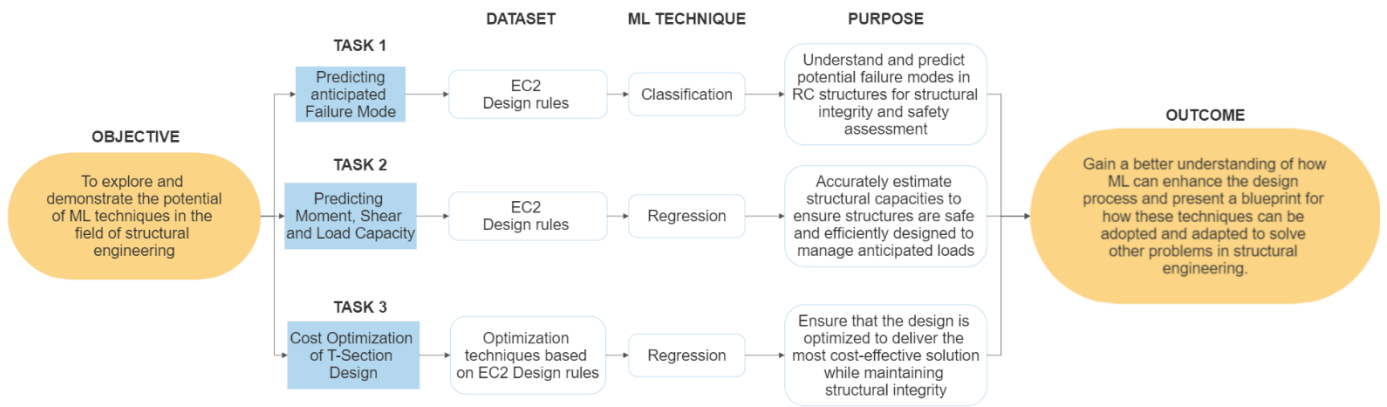


Figure 1: Objective visualized

1.3. Outline of this thesis

The structure of this thesis follows a logical flow, sectioned into eight chapters, each following a coherent narrative for the application of ML in the three tasks presented above. Beginning with the theoretical framework in Chapter 2, we lay the foundation for understanding the principles of optimization and ML concepts used in this thesis.

Our research methodology, including the various software and Python packages used to produce the results, is discussed in Chapter 3. Chapter 4 sheds light on the conversion of EC2 design rules, specifically those related to limit state design concerning moment, shear, and deflection, into a machine-readable format. These transformed rules then serve as the foundation for generating datasets for each of the three tasks. The datasets derived from these rules form the training data for our ML models. An added layer of complexity is introduced for Task 3, where an optimization process is employed in the dataset generation.

Subsequently, we delve into the three main tasks: anticipating failure mode (Chapter 5), predicting capacities (Chapter 6), and cost optimization of T-section design (Chapter 7). Each task leverages unique ML techniques, involves data preprocessing, hyperparameter tuning, and analysis of results.

The thesis concludes with a summary of key findings (Chapter 8), followed by the limitations of our current work and prospects for future research directions. Throughout, this thesis serves as a comprehensive exploration of ML's application in structural engineering.

2. Theoretical Framework

This chapter aims to give a thorough understanding of the fundamental ideas that guide our model building, investigation, and discussion throughout this work. The chapter is segmented into four main sections: Optimization, Machine Learning, Hyperparameter tuning and Evaluation metrics, each containing several sub-sections to delve into specifics.

The first part focuses on optimization, specifically elaborating on trust-region methods. These techniques are fundamental to the dataset generation of the optimization task (Chapter 4.3.1). The second part introduces ML - the foundation of our predictive models. It presents an overview of the learning process, then segues into specific ML techniques, namely Classification and Regression, and the various models used in this thesis for each technique. Following this, the third part focuses on the vital process of hyperparameter tuning. It digs into techniques such as *GridSearchCV* and *RandomSearchCV*, which have been extensively employed in the thesis for the calibration and optimization of machine learning models, as discussed in detail in Chapter 2.3. In the last part, the primary emphasis lies on the key evaluation metrics employed for assessing the performance of these models. Each category of ML techniques, be it classification or regression, requires distinct metrics to provide a comprehensive understanding of the model's effectiveness and suitability for the tasks.

2.1. Optimization

Optimization is a process that seeks to find the “best” solution from all feasible solutions [3]. In structural engineering, this often involves determining the most effective design that also meets structural requirements while minimizing costs and material usage.

Mathematically, structural optimization can be formulated as a problem of finding the minimum or maximum of an objective function, subject to a set of constraints. The objective function represents the parameter that we wish to optimize, such as minimizing the total weight of a structure, or maximizing its load-bearing capacity. The constraints represent the limitations or requirements that the design must meet. These could include physical constraints (like the strength of available materials or geometrical restrictions), regulatory constraints (such as design rules), or practical constraints (like budget and time limitations).

2.1.1. Trust-region method

One fundamental approach to solving optimization problems is through iterative methods. A prominent example of these methods is the trust-region method. In Task 3 (Chapter 7), this approach was effectively utilized to navigate the presented optimization issue. In this technique, a trust-region is established around the current estimate of the optimal solution. This region is where the model is considered a reliable approximation of the objective function, hence the term “trust-region”. The size of the region is dynamically adjusted based on the quality of the model's approximation of the objective function [4]. The method measures the agreement between the model and the objective function, using a ratio of the difference in their values at the current and next step. When the model's approximation is good, indicated by a ratio close to 1, the trust-region is expanded, allowing the algorithm to explore a larger portion of the solution space in the next iteration. Conversely, when the model approximation is poor, reflected by a ratio close to 0 or negative, the trust-region is contracted to take a more cautious step in a smaller portion of the solution space. If the ratio is neither close to 1 nor close to 0, indicating satisfactory agreement, the size of the trust-region remains unchanged for the next iteration [5].

Real-world optimization problems frequently involve various limitations. In our T-section cost optimization task, we impose physical constraints, including geometrical properties, capacity, and material limitations. These conditions necessitate the use of constrained optimization. In this approach, we aim to minimize the cost, serving as our objective function, within the set of feasible solutions dictated by these constraints.

A common algorithm to solve this kind of problem is the trust-region constrained algorithm, often abbreviated to the *trust-constr* method, which combines the trust-region concept with constrained optimization. It incorporates two sub-methods: the Byrd-Omojokun Trust-Region SQP (BTR-SQP) method and the Trust-Region Interior-Point (TR-IP) method [6]. The BTR-SQP method, primarily used when dealing with equality constraints, can be interpreted as a sequential quadratic programming method with a trust region. The algorithm simplifies the optimization problem into smaller unconstrained trust region subproblems, which are easier to solve [7]. In essence, it takes advantage of quadratic programming efficiency in exploring the solution space while maintaining the trust-region concept [5].

However, when the problem includes inequality constraints (such as Eq. (47) in Chapter 4.3), the *trust-constr* method transitions to using the TR-IP method. This technique is well-equipped for handling inequality constraints by transforming the original problem into a series of equality-constrained problems. It introduces slack variables to convert inequality constraints into

equalities and employs a barrier function to penalize solutions violating the constraints. Similar to the BTR-SQP method, it also employs the trust-region concept to dynamically adjust the search region based on the progress of the optimization [8] [9].

In the context of ML-model training, optimization is used to modify the model's parameters in order to reduce a loss function, which measures the difference between the predictions made by the model and the actual data. Typically, this is accomplished using an iterative method, like gradient descent, where the model parameters are gradually adjusted in the direction that minimizes the loss function.

2.2. Machine Learning

ML equips computers with the ability to make predictions using predefined datasets and algorithms. While its inception dates back to 1943, ML's significant evolution began in the 1990s, establishing it as a powerful AI tool. It has been utilized to tackle a variety of intricate real-world problems, ranging from speech and image recognition to traffic prediction, autonomous vehicles, and medical diagnostics, among others [1].

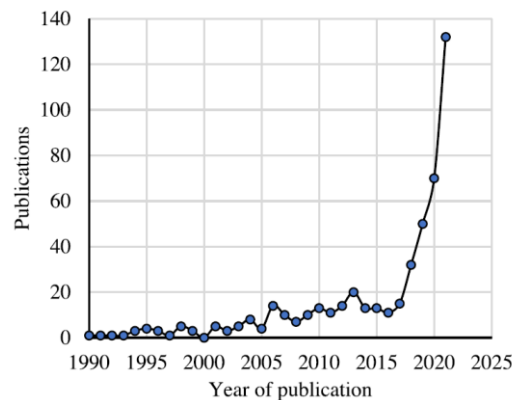


Figure 2: Yearly distribution of ML-related articles in structural engineering [1]

In recent years, the field of structural engineering has experienced a notable increase in the application of ML, as demonstrated by the exponential growth in the number of related publications produced annually in Figure 2. This rapid progress can be ascribed to the ongoing enhancements in ML algorithms, the rise in computational power, and the accessibility of newly established databases.

2.2.1. Learning processes

ML is broadly categorized into three distinct learning processes: supervised learning, unsupervised learning, and reinforcement learning. The primary difference between these categories revolves around how they handle data, specifically whether the data is "labeled" or "unlabeled". Labeled datasets consist of data points or samples that are associated with a known outcome or target variable. For instance, in a supervised learning scenario such as classifying emails, each email (data point) would have an associated label, i.e., "spam" or "not spam".

In contrast, unlabeled datasets are comprised of data points that lack corresponding target variables [10]. The outcomes or answers for these data points remain unknown. This type of data is primarily used in unsupervised learning tasks, such as clustering and dimensionality reduction. In these tasks, the algorithm's goal is to uncover underlying patterns, simplify the data, or extract insight from the data, rather than making predictions about specific outcomes.

This thesis predominantly focuses on supervised learning, a process that utilizes labeled datasets for training algorithms. It is an ideal methodology for regression and classification problems, which constitute the core of our investigation [10]. To illustrate, in the context of our tasks, the labeled data would include features of the RC beams along with their corresponding outcomes like anticipated failure mode or load capacity.

Reinforcement learning, the third learning type, operates on a trial-and-error basis to train algorithms [11]. The different learning processes are visualized below.

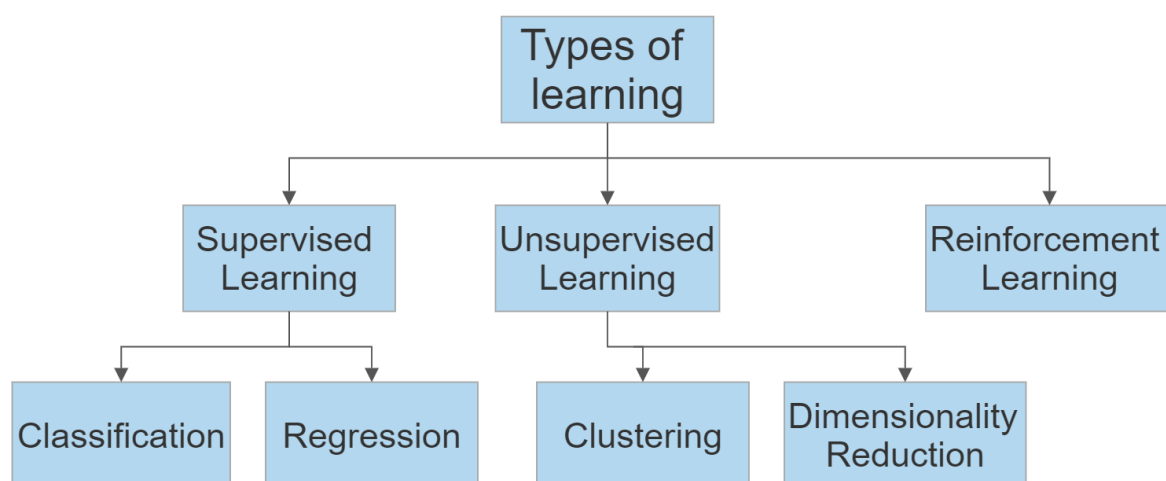


Figure 3: Summary of types of learning

2.2.2. Classification

Classification is a fundamental concept in the field of ML, aiming to categorize data points or objects into predefined categories or classes based on their distinctive characteristics. This process involves building models or algorithms that can learn from existing labeled data to make predictions or decisions about new, unseen data [12]. Classification encompasses various algorithms that are widely used in ML. Some of the most common classification algorithms include Decision Trees, K-Nearest Neighbors and Neural Networks. Each, with its own strengths and weaknesses, will be used to tackle Task 1 (Chapter 5).

2.2.2.1. Decision Trees

Decision Tree (DT) is a supervised learning technique that may be used for both classification and regression problems, however it is most commonly utilized for classification. It is a tree-structured classifier in which internal nodes contain dataset features, branches represent decision rules, and each leaf node represents the outcome.

There are two nodes in a DT: decision nodes and leaf nodes. Decision nodes are used to make decisions based on input data, while leaf nodes represent the predicted outcome or class label for a given input, and do not contain any more branches. It is named a DT because, like a tree, it begins with a “root node”, and then branches out to form a tree-like structure, as shown in Figure 4. A DT asks a question and then divides into subtrees based on the answer (yes/no) [13].

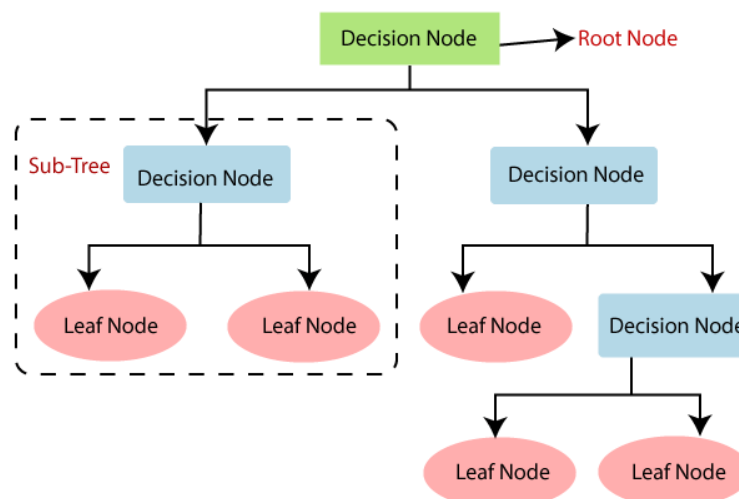


Figure 4: Visualization of a DT [14]

Some advantages of employing DT [15] are:

- DTs exhibit an inherent capability to imitate human decision-making, enhancing their comprehensibility. They project a structured, tree-like framework, providing an effective visualization of the decision-making logic. The simplified decision-making process behind Task 1 is visualized in Appendix E.
- DTs are capable of handling both numerical and categorical data in multi-output scenarios.
- The performance of DTs is not influenced by the nonlinear relationships between parameters.
- Compared to other algorithms, DTs require less data cleaning, as they can handle datasets with potential errors and missing values.

Cons of employing DT [15]:

- There is a risk of creating overly complex trees that do not generalize the data well, a phenomenon known as overfitting. Overfitting occurs when a model captures not only the relevant patterns, but also the noise in the training data to the extent that it negatively impacts the performance of the model on new data.
- DTs can be unstable because even small variations in the input can produce a drastically different tree. This instability is indicative of high variance, which can, however, be reduced by techniques like bagging and boosting.
- DT learners may create biased trees if some classes dominate in the dataset.

Many ML models rely on hyperparameters to optimize their predictive capabilities. They control the behavior and performance of the model and are often adjusted through hyperparameter tuning as showed in Chapter 2.3 and the “Hyperparameter tuning” section of each task. The hyperparameters tuned in Task 1 and 2 will be those provided by Scikit-learn. An explanation of what Scikit-learn is, can be found in Chapter 3.2. The hyperparameters that affect the DT [16] are:

Max depth

The *max_depth* parameter can be used to specify the maximum depth of our DT. Shallow decision trees do not overfit the data in general, but they perform poorly (high bias, low variance). Deep trees, on the other hand, tend to overfit and perform well (low bias, high variance) [17]. Ideally, we are looking for a tree that is not so shallow that it has low performance, and not so deep that it overfits the training dataset. We can anticipate that as the depth of the DT increases, the performance of both train and test datasets will improve to a point, and then, as the tree becomes

too deep, it will begin to overfit the training dataset at the expense of lowering the performance of the test set [17].

Minimum split samples

This parameter can contribute to minimizing overfitting by setting a minimum number of samples for each internal node split, which is represented by *min_samples_split*. As previously stated, each internal node is a question or test on a feature, and the branches that emerge from it represent all potential responses to that question. The value specifies the minimum number of samples necessary to divide a node. If a node's sample count is less than the parameter, the node will not be split and will therefore become a leaf node. By increasing the *min_samples_split* value, we may prevent overfitting to the training data by requiring that a node have a minimum number of samples before it can be split, resulting in a simpler tree with less potential to overfit the data. Setting this value too high, on the other hand, may result in underfitting, in which the model is too basic and does not capture significant relationships in the data.

Minimum leaf samples

Min_samples_leaf parameter controls the minimum number of samples required to be at a leaf node. Increasing *min_samples_leaf* can help prevent overfitting and improve generalization, but may also reduce the model's ability to capture complex patterns in the data.

Max features

Another feature to consider is the *max_features* parameter. This specifies the maximum number of features to consider when finding the optimum split at each node of the tree [18]. *Max_features* is set to *none* by default, which means that all features are considered for each split. Considering all features may result in overfitting, and to avoid this, we can adjust the parameter to a lower value, which can improve model generalization by limiting the number of features considered in each split. Values such as *log2* or *sqrt* of the total number of features in the dataset are frequently employed and provide a good compromise between model complexity and generalization performance.

Criterion [19]

The final parameter to consider is *criterion*, which assesses the quality of a split. The two most popular criteria used by scikit-learn are *gini* and *entropy*. The *gini* (Gini Impurity) quantifies the likelihood of incorrect classification if a label is assigned randomly to any data point within the dataset. It achieves its lowest value, 0, when all instances within a node correspond to a single target category. Consequently, no further splitting is required for such a node. Therefore, feature

splits yielding a lower *gini* are preferred as they drive the decision tree towards optimal classification. The *gini* is calculated as:

$$Gini\ Impurity = 1 - \sum_{i=1}^n (p_i)^2 \quad (1)$$

where the probability of choosing an item with label i is p_i .

Entropy is a metric indicating the level of disorder or randomness in the relationship between features and the target variable. The feature yielding lower *entropy* is chosen for the optimal split. *Entropy* reaches its maximum when the probability of both classes is identical, signaling a high degree of disorder. Conversely, when *entropy* is at its minimum (0), it signifies a pure node, indicating no disorder or a clear distinction between classes. The *entropy* is calculated using the following formula:

$$Entropy = - \sum_{i=1}^n p_i \log_2 p_i \quad (2)$$

where, as before, p_i is the probability of class i .

Entropy is more complex as it makes use of logarithms as seen in Eq. (2) and can therefore be more computationally expensive than the *gini*. There is ongoing debate about which metric is superior, with some studies, such as the one conducted by Laura Elena Raileanu and Kilian Stoffel [20], suggesting that they typically yield similar results. However, certain experiments [21] [22] have concluded that *entropy* is indeed more effective when dealing with unbalanced datasets.

2.2.2.2. K-Nearest Neighbors

The k-nearest neighbor (kNN) algorithm is a well-known and widely used method for classification because of several interesting features, including good generalization and easy implementation. Although simple, it is known to be capable of achieving results comparable to those of much more complex algorithms [23]. It is a type of instance-based learning, which means that it does not construct a general internal model, but simply stores instances of the training data. In other words, all computation is deferred until it is asked to make predictions on new unseen data [24].

KNN classification works by identifying the k -nearest data points (that is, the closest neighbors) in the training dataset to the new data point for which we want to make a prediction. To predict the class label for the new data point, the algorithm then uses the class label most common among its k -nearest neighbors [25].

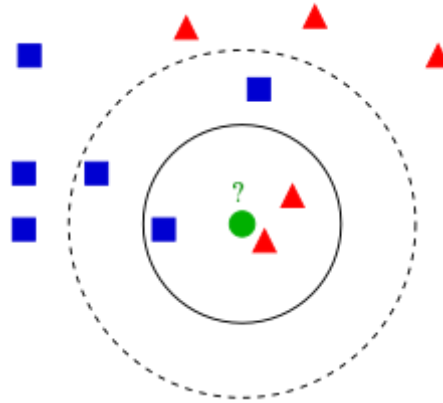


Figure 5: An example [26] of k NN with $k = 3$ (solid line circle), $k = 5$ (dotted line circle)

Figure 5 depicts an example of k NN implementation for a case with two classes. Each feature in the two-dimensional feature space in which the samples are placed is of one dimension. The objective is to categorize the new data point represented by a green circle into the two classes, which are symbolized by triangles and blue squares, respectively. Given that there are two triangles and only one square in the inner circle produced by $k = 3$, the new data point is assigned to the red triangles. If $k = 5$, it is assigned to the blue squares since now there are three squares and only two triangles.

As can be seen, this algorithm uses distance for classification, so if the features have different physical units or scales, normalizing the training data can greatly improve the accuracy [10]. On the contrary, DT as seen in Chapter 2.2.2.1 works by splitting the data based on the feature values, so the scale of the features does not matter. Some of the advantages of k NNs [27] include:

- Ease of implementation: The algorithm is straightforward and easy to understand. It does not require any complex mathematical equations and has few hyperparameters.
- Does not require an explicit training phase. Since it does not require a training period, it is significantly faster compared to other ML algorithms. It simply memorizes the training data and uses it to make predictions on new unseen data. Because of this, it can add new data without affecting the accuracy of the algorithm, as the algorithm can adapt to new data without having to retrain the model.

The disadvantages of kNNs [27] consist of:

- Sensitivity to noisy data, meaning a lack of robustness against outliers, missing values and inconsistencies that do not follow the expected pattern of the dataset. Especially outliers can skew the distance calculations and lead to incorrect results. Per definition, an outlier is an observation that is “*different from or inconsistent with the rest of the data*” [28].
- Tendency to make slow predictions when given a large amount of data or a large number of input variables, as it can become computationally expensive to search through all the training examples to find the nearest neighbors.
- Curse of dimensionality: The algorithm is known to perform poorly when given high-dimensional data inputs. This is sometimes referred to as the peaking phenomenon [29], where after the algorithm reaches the optimal number of features, additional features cause a rise in the number of classification errors. In other words, as the number of features increases, the amount of data needed to obtain accurate predictions increases exponentially.

The hyperparameters of kNN are the number of neighbors (k), the weighted function, and the distance metric (p) [27].

Value of k

In kNN, k refers to the number of nearest neighbors that are used to make a prediction for a new data point. Defining k can be a balancing act, as a higher value of k will result in a less distinct decision boundary and can reduce overfitting, but it can also lead to less accurate predictions for more complex datasets. A lower value of k will result in a more complex decision boundary and can increase overfitting but may also lead to more accurate predictions for complex datasets.

KNN ties in classification can occur when the value of k is even and the number of neighbors that belong to different classes is equal. This can lead to ambiguity in the classification result, as there is no clear majority class. Therefore, it is recommended to have an odd number for k to ensure that there cannot be an equal number of neighbors belonging to different classes.

Distance metric [13]

The distance metric is used to calculate the distance between data points and is defined by the distance function in the form of the Minkowski metric described in Eq. (3).

$$D(x, y) = \left(\sum_{i=1}^k |x_i - y_i|^p \right)^{\frac{1}{p}} \quad (3)$$

Where D is the distance between instances x and y .

Since the dataset contains only continuous input features (Chapter 4.2.1), we will be looking at the following popular distance metrics:

- $p = 1$, is equivalent to using the Manhattan distance, which calculates the distance between real vectors based on the aggregate of their absolute differences. This is beneficial for datasets with varied feature scales, as it does not overly emphasize features with larger variances by avoiding the squared differences between feature values.

$$D(x, y) = \sum_{i=1}^k |x_i - y_i| \quad (4)$$

- $p = 2$, results in the Euclidean distance being utilized, computed as the square root of the sum of the squared differences between a new (x) and an existing point (y). This is fitting for datasets with similar feature scales, as the larger variances in feature values contribute more due to the squared differences in the distance computation.

$$D(x, y) = \sqrt{\sum_{i=1}^k |x_i - y_i|^2} \quad (5)$$

Weight function [30]

The weight function in the kNN algorithm is used to assign weights to the nearest neighbor based on their distance from the new data point. The most frequently used weight functions are *uniform* weighting and *distance* weighting.

Uniform weighting assigns equal weight to all nearest neighbors, no matter how far they are from the new data point. This can be useful when all neighbors are equally important for making a

prediction. When dealing with imbalanced datasets, assigning equal weights to all instances may lead to poor classification performance on the minority classes.

On the other hand, *distance* weighting assigns higher weight to the nearest neighbors and lower weight to the ones farther away. The weights are therefore used in a way that scales the contribution of each class instance to the final prediction. By assigning higher weights to the minority classes their importance increases, leading to improved performance on the minority classes.

Algorithm [30]

The algorithm parameter in scikit-learn's implementation of kNN provides three different algorithms for finding the nearest neighbors:

- Brute force algorithm (*brute*) computes the Euclidean distances between the new data point and all points of the training set to find the nearest neighbors. *Brute* may be the most accurate method due to the consideration of all data points and is most suitable for small datasets or when the number of features is low.
- K-dimensional Tree (*kd_tree*) rearranges the whole dataset in a hierarchical binary tree structure and searches for the nearest neighbors by traversing through the tree. This algorithm is faster than brute force when the dataset is large, and the number of features is high.
- Ball tree (*ball_tree*), is also a hierarchical data structure similar to *kd_tree* and is particularly useful, and has shown to outperform other algorithms, when dealing with high dimensional data. However, it may not perform as well with lower-dimensional data.
- *Auto* in scikit-learn will decide the most appropriate algorithm based on the values passed to fit the method. One can access the chosen algorithm by printing *knn_clf_fit_method*.

2.2.3. Regression

In the field of ML, regression analysis operates as a predictive modeling technique designed to investigate the relationship between a dependent (target) and independent variable(s) (input variables). This technique aims to predict a continuous outcome [10], which refers to a target variable that has the potential to assume any value within a specific range, as opposed to discrete categories. The method is particularly applicable when the output is quantitative [10] such as in our research, where we apply it to predict the moment, shear, and load capacity of a RC beam, or the value of design variables in a T-section. Several regression techniques exist, with Linear

Regression and Support Vector Regression being particularly prominent due to their efficacy and widespread use.

2.2.3.1. Multiple Linear Regression

Linear Regression (LR) is a widely employed and straightforward technique in regression, making it the most utilized method in predictive analysis. The goal of LR analysis is to identify a line that best fits the data points. When performing an LR analysis, we plot various lines and then choose the one that best matches the data points. At the heart of the relationship between the variables and the outcome lies the equation of a straight line, $y = ax + b$. The ideal coefficients a and b are computed to establish this line. The relationship between the dependent and independent variable can be visually represented using a simple x-y coordinate system, as demonstrated in Figure 6.

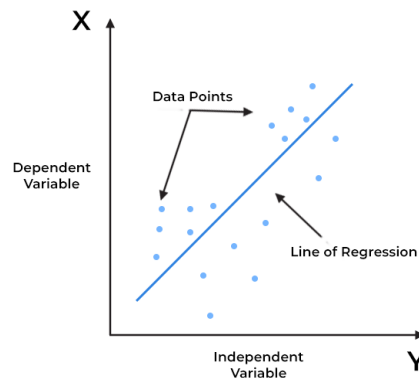


Figure 6: Illustration of a best fitted line in linear regression [31]

While simple LR can map a relationship between two variables, Multiple Linear Regression (MLR) can handle systems where the dependent variable is influenced by more than one independent variable. The MLR model is represented as follows [32]:

$$Y = a + b_1X_1 + b_2X_2 + \dots + b_nX_n + \varepsilon$$

where,

Y is the dependent variable

a is the y-intercept (the value of Y where X=0)

b_1, b_2, \dots, b_n are the coefficients of the independent variables

X_1, X_2, \dots, X_n are the independent variables

ε is the error term

Some advantages of employing MLR are [33]:

- MLR's mathematical formula is relatively simple to comprehend and interpret. As a result, the algorithm is simple to learn.
- MLR, with its straightforward procedure, can yield sufficiently accurate results.
- Compared to other complicated methods, these models can be trained quickly and effectively even on systems with less computational capability.
- Finding the nature of the relationship between variables is frequently done using LR, which almost perfectly fits linearly separable datasets.

Some disadvantages of employing MLR are [33]:

- The MLR method assumes the independent variables are independent of each other, which is not always the case.
- MLR may occasionally suffer from underfitting, a situation where the ML model fails to capture the underlying complexity of the data accurately.
- Since the majority of naturally occurring phenomena are nonlinear, the linear regression technique cannot adequately fit complicated datasets since it assumes that the input and output variables have a linear relationship.
- Due to its sensitivity to outliers in the data, MLR can severely degrade performance and produce models with low accuracy, as it affects the slope of the regression line drastically.

As for hyperparameters, MLR does not typically involve hyperparameter tuning. The coefficients in MLR are not considered hyperparameters but parameters of the model. They are determined by the model during training, usually via a method (cost function) like least squares, where the model minimizes the sum of the squares of the differences (residuals) between the predicted and actual values [34]. The residuals' properties are reviewed in detail in Chapter 2.4.2.3.

2.2.3.2. Support Vector Regression

A sort of regression analysis called Support Vector Regression (SVR) was developed using the Support Vector Machine's (SVM) underlying concepts. Therefore, understanding the fundamental idea of an SVM is essential to understanding how SVR works. SVM is a very effective and well-liked method due to its precision and simplicity, and is usually employed for classification issues. The fundamental principle of the SVM method is to separate different sets of data features, referred to as vectors, and then locate an ideal separating hyperplane that has a greatest margin,

as illustrated in Figure 7. Support vectors, which affect the hyperplane's position and orientation, are the data points found on the margins. In other words, SVM uses support vectors to optimize the margin [1].

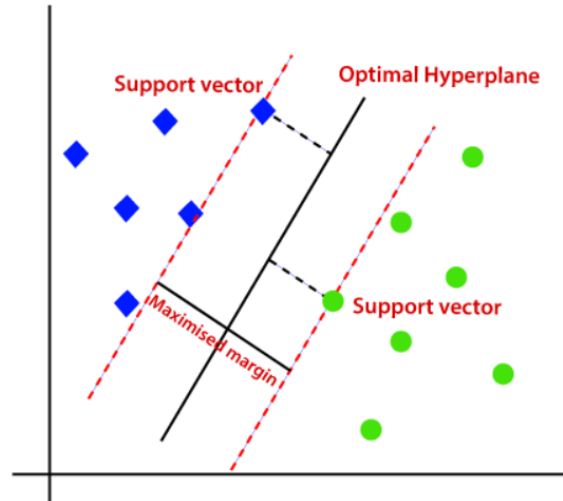


Figure 7: Illustration of the principles of SVM [35]

For regression issues, SVR uses the same principle as SVM. The SVR algorithm uses LR to identify a function that, within a decision boundary, best fits data points. The hyperplane with the greatest number of data points within a threshold value is the best fitted line, as shown in Figure 8. Finding a separating hyperplane is unachievable since data in the majority of real-world applications cannot be separated in a linear fashion. In this instance, a kernel function and penalty parameter are used [1].

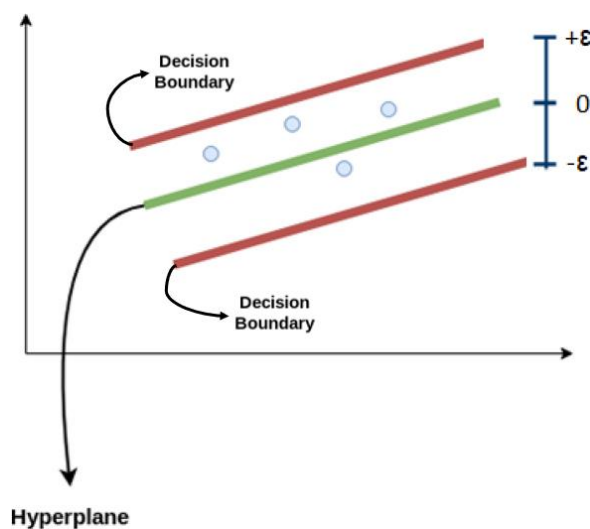


Figure 8: Illustration of the principles of SVR [36]

SVRs share the same advantages and disadvantages as SVMs. Some of the advantages [37] are:

- They are effective in high dimensional spaces, meaning they perform well when each datapoint has many attributes, making it suitable for datasets with many features, or sections with many properties, as in our case.
- They tend to not overfit when dealing with noisy datasets and are less sensitive to outliers compared to other regression algorithms.
- SVRs can provide good results with limited information and work well with unstructured data.
- With a convenient kernel solution function, SVRs can solve complex problems.

Some of the disadvantages of SVRs [37] are:

- Selecting the right kernel function for SVR can be challenging and may require expertise and experimentation.
- Due to the complexity of the model and the lack of explainable reports, SVRs can sometimes be difficult to interpret.
- When the dataset is large, the training time can be inefficient.

The hyperparameters of SVR are the Kernel function, the regularization parameter (C), the margin of tolerance (ϵ), the degree (for polynomial kernel) and gamma (γ) (for RBF kernel).

Kernel function

Several methods, such as logistic and linear regression, can process and categorize data exhibiting linear behavior. However, SVRs extend this capability by effectively handling data that demonstrates substantial nonlinearity. SVRs may successfully handle complicated, multi-dimensional data thanks to the application of kernel functions, which transform the original nonlinearly separable data into a new space where the data are linearly separable. We refer to this as the “kernel trick” and is illustrated in Figure 9. The performance of SVR models is greatly influenced by the choice of kernel function, with radial basis function, linear, and polynomial kernels being the most commonly used [1].

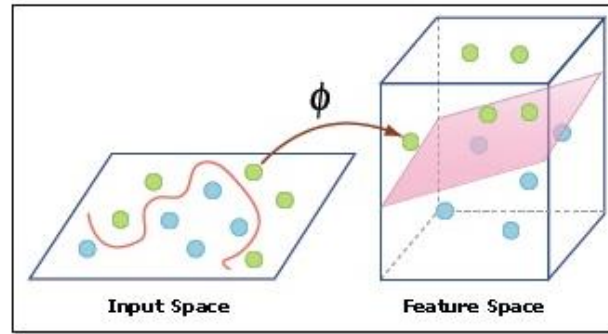


Figure 9: Illustration of how the kernel trick is used [36]

The three kernels are explained as follows [1]:

- Linear kernel (*linear*): Dot product between two observations.
- Polynomial kernel (*poly*): Allows curved lines in the input space.
- Radial basis function (*rbf*): Creates complex regions in the feature space.

Epsilon (ϵ)

This parameter determines the size of the ϵ -insensitive zone, as seen in Figure 8. It serves as a specification for the desired accuracy and can be thought of as the width of the tube around the hyperplane. A smaller ϵ results in a narrower tube and vice versa. Within this tube, no penalty is given to errors, which allows the model to disregard small fluctuations and noise in the data [38].

Regularization parameter (C)

The C -parameter regulates the “penalty” for each misclassified datapoint. In essence, it balances between correctly classifying the training samples and optimizing the margin of the decision function. A smaller C -value implies reduced penalties and consequently, an increased occurrence of misclassifications. However, this also translates to a wider margin for the decision boundary. On the contrary, larger values of C lead to narrower margins, which could trigger overfitting. Nonetheless, it is important to note that while lower C -values might facilitate a broader margin, it also runs the risk of underfitting the data [12].

Gamma (γ)

This parameter is used in nonlinear SVRs with *rbf* kernels. It defines how far the influence of a single training example reaches. Low values mean far, and high values mean close.

When γ is set to *scale*, the value of γ is computed as $1/(n_features * X.var())$, where $n_features$ is the number of features in the dataset and $X.var()$ is the variance of the data. This setting

essentially scales the influence of individual data points based on the variability of the dataset. It tends to work well when the range of values in different features varies considerably.

If γ is set to *auto*, it uses $1/n_features$ as the value of γ . This setting simplifies the influence of individual data points to be inversely proportional to the number of features. It is a simpler option and may be more appropriate for datasets where the features are relatively homogenous [39].

Degree

Refers to the degree of the polynomial kernel function *poly* [38].

2.2.4. Multilayer Perceptron

As observed in the study by Huu-Tai Thai [1], the development of processing power has made Artificial Neuron Networks (ANN) particularly well-liked in the structural engineering field. Neurons comprising a single layer are termed Single-Layer Perceptrons (SLP). As depicted in Figure 10, each neuron processes data by multiplying input values (x_i) with their corresponding weights (w_i) and adding a bias (b) which is iteratively adjusted to diminish the discrepancy between the predicted and actual outputs. The activation function then decides if the neuron should forward its output to the succeeding layer (output layer y), based on the result of the input summation. This process is typically accomplished using backpropagation, an algorithm that calculates the gradient of the error with respect to each weight in the network and uses it to update the weights [40].

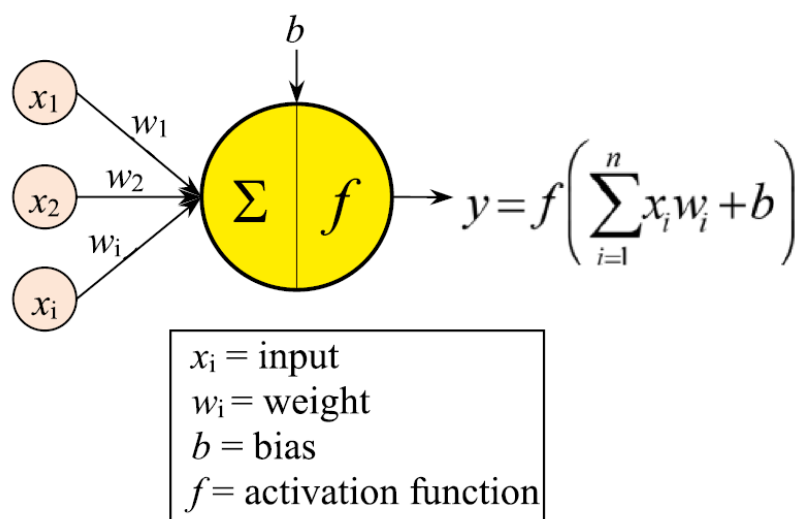


Figure 10: Processing neuron [1]

An SLP alone has limited capability to surpass other ML methods, thus multiple neurons are integrated to form a Neural Network (NN). The advanced configuration, now known as a Multilayer Perceptron (MLP), is a type of feedforward artificial NN that comprises a minimum of three node layers; these include an input layer, one or multiple hidden layers, and an output layer (Figure 11). The input layer accepts the feature-representing input data and relays it to the hidden layers, where linear and nonlinear transformations are performed to derive relevant features from the input data. The output of the hidden layers is then forwarded to the output layer, which contains nodes that correspond to the predicted classes. Notably, each node, except for the input nodes, is a neuron that uses a nonlinear activation function.

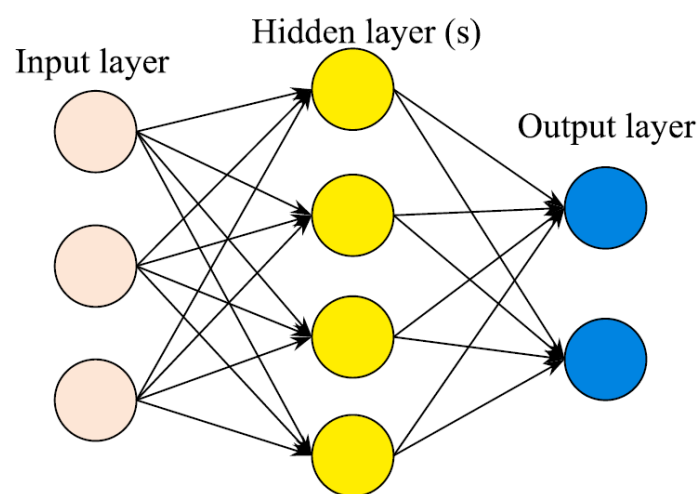


Figure 11: Network architecture [1]

An MLP's architecture will change based on the task it is intended to solve. The number of nodes in the input and output layers is determined by the dimensions of the input and output data, respectively, while the number of hidden layers and the number of nodes in each hidden layer is typically found by trial and error or through a more systematical search with techniques like grid search or random search (explained in Chapter 2.3). As will be demonstrated in this thesis, MLPs can effectively serve the purpose of both regression and classification tasks.

Some of the advantages of MLP [40] include:

- Capability to learn nonlinear models, which can capture complex relationships between the input and the target variable, making them suitable for real-world problems where the underlying relationships may not be linear. The ability comes from the application of nonlinear activation function in each neuron.

- Capability to learn models in real time (on-line learning). The model updates its parameters incrementally, as it receives new data samples without having to retrain the model from scratch. This can be an important feature in cases where the model needs to continuously adapt to new data in real-time.

Some of the disadvantages [40] of MLP include:

- MLP requires several hyperparameters such as the number of hidden neurons, layers, and iterations, which must be tuned in order to achieve better performance for the specific task. Selecting optimal values for these hyperparameters is often challenging and requires a combination of trial and error, intuition, and expertise.
- MLP is sensitive to feature scaling, hence the scale and distribution of the input features can have a big impact on how well it performs. The weights and biases of the neurons are updated during training using an optimization algorithm. The optimization algorithm adjusts the weights and biases of the neurons in an attempt to minimize the error between the predicted output of the MLP and the true output. However, if the input features have different scales or distributions, the optimization algorithm may take longer to converge or may possibly fail to converge. This is due to the fact that features with larger scales or ranges can dominate throughout the optimization process, causing oscillations or slower convergence. Therefore, it is important to preprocess the input data by scaling the features to improve stability and convergence.
- A MLP with hidden layers typically has a non-convex loss function, meaning that it has multiple local minima. During training the MLP aims to identify the set of weights that minimizes the loss function during training. However, since the loss function is non-convex, different weighting schemes can provide similarly low loss function values. Different random weight initializations can therefore result in varying validation accuracy.

There are several crucial parameters to consider and tune when creating an MLP classifier to improve its performance, such as:

Number of hidden layers

The number of layers is a critical parameter in an MLP model, as it heavily affects how well the model can learn and generalize. Increasing the number of hidden layers can improve the model's ability to learn complex nonlinear relationships in the data. However, adding too many hidden layers can lead to overfitting.

Number of neurons in hidden layers

As for the number of hidden layers, the number of neurons in each layer can affect how well an MLP can learn and generalize. The model's ability to learn complex patterns will increase as the number of neurons is increased, but it can also lead to overfitting.

Activation function

The activation function, also known as the transfer function (f), maps the weighted sum of the inputs to an output value (see Figure 10), and introduces nonlinearity into the network [41]. Without an activation function, MLPs would be equivalent to a linear model, which only model linear relationships. Although various layers may use different activation functions, all hidden layers typically use the same function, while the output layer normally uses a different activation function from the hidden layers. This function is based upon the type of prediction required by the model. In the realm of NNs, numerous activation functions are available. However, the scope of this thesis is restricted to the most commonly employed activation functions, which are detailed below. These chosen functions not only hold significant relevance in the field but are also available as options in the Scikit-learn package. It is worth noting that a more extensive list of activation functions can be found in the Keras API, although we limit our focus to those compatible with Scikit-learn for this thesis. Detailed insights on Scikit-learn and Keras packages, as well as their roles and applications in this research, are provided in Chapter 3.2

The *Logistic Sigmoid* function takes a real-valued-input and squashes it between 0 and 1. The larger the input, the closer the value will be to 1, whereas the smaller the input, the closer the output will be to 0 [42] [43]. When the input values are too high or too low, the function saturates at 0 or 1, with a derivative extremely close to 0, which can significantly slow down the learning process [44]. The mathematical representation is as follows:

$$f(x) = \frac{1.0}{1 + e^{-x}} \quad (6)$$

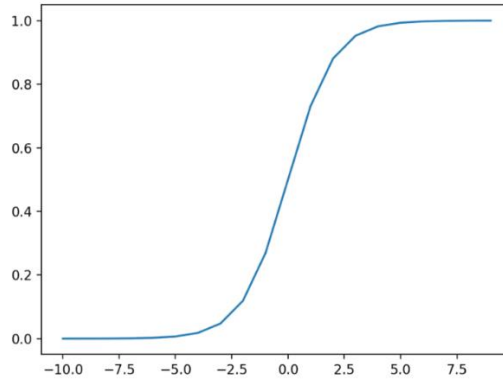


Figure 12: Plot of Inputs vs. Outputs for logistic Sigmoid. [45]

The *tanh* activation function is quite similar to the sigmoid activation. The function squashes the real-valued-input between -1 and 1, providing a zero-centered output which can make learning for the next layer easier [46]. The larger the input, the closer the output value will be to 1, whereas the smaller the input, the closer it will be to -1. However, like the sigmoid functions, *tanh* also suffers from the vanishing gradient problem where the gradients become very small if the input is far from 0. The mathematical representation is:

$$f(x) = \tanh(x) = \frac{\sin(x)}{\cos(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (7)$$

The *ReLU* activation function takes a simple form: for an input value x , if x is positive, it returns x , and if x is negative, it returns 0. This means that it effectively “rectifies” inputs to be non-negative [43]. It is commonly used in the hidden layers of NNs because it has overcome some of the limitations of popular activation functions such as the *Sigmoid* and *Tanh*, as it has shown to be less susceptible to vanishing gradients that prevent deep models from being trained.

The vanishing gradients problem describes the situation where the gradients of the loss function become very small as they backpropagate through the layers of the network. As they become very small, their products become even smaller and approach zero for the earlier layers. Because of this, the weights in the network's earlier layers are updated with very small increments, causing them to learn very slowly or not at all [47]. Although important to have in mind, the likelihood of encountering the vanishing gradient problem in a network with just three and two hidden layers, as used in Task 1 and 2 (Chapter 5.2.3 and 7.2), is low. The *ReLU* function is calculated and plotted as follows:

$$f(x) = \max(0, x) \quad (8)$$

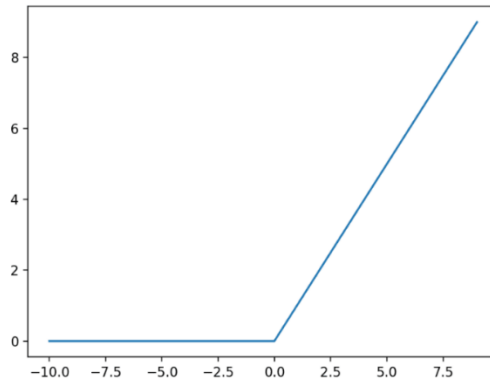


Figure 13: Plot of Inputs vs. Outputs for reLu [45]

The *Softmax* activation function is commonly used in the output layer of multiclass classification tasks. It takes a vector of arbitrary real-valued scores and squashes it to a distribution of values between 0 and 1 that collectively sum to 1 [48]. The output of the *Softmax* function can be interpreted as probabilities for each class in a multi-class problem, and the class with the highest probability can be selected as the prediction.

$$f(x) = \frac{e^x}{\sum e^x} \quad (9)$$

Learning rate

The learning rate determines the step size for updating the model's weight during training and controls in that way the rate of speed at which the model learns. A high learning rate can lead to faster convergence, but it can also cause the model to converge too quickly, resulting in a suboptimal solution and unstable training. A learning rate that is too low can ensure stable training, but can cause the learning process to be too slow or get stuck [49].

The range of values to consider for the learning rate is typically less than 1 and greater than 10^{-6} [49]. A visualization of how different learning rates affect learning can be shown in Figure 14.

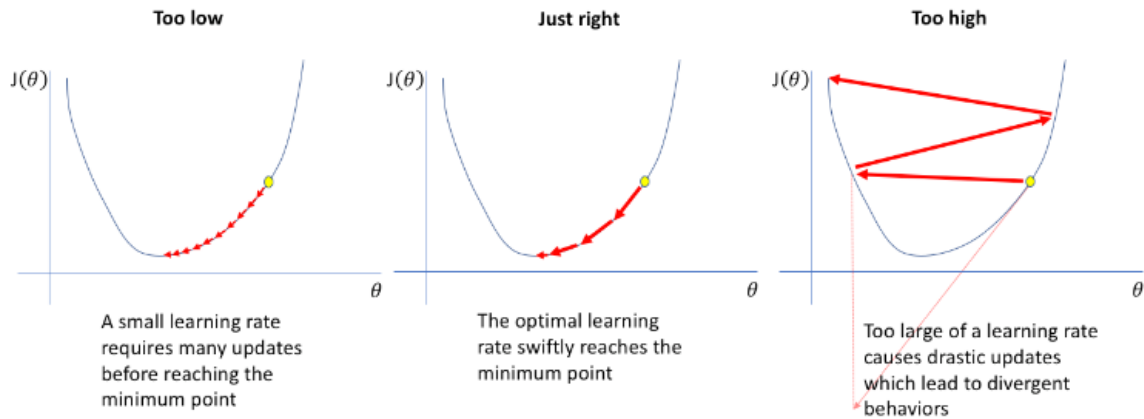


Figure 14: Visualization of different cases of learning rates [50]

Optimizers

Optimizers are algorithms that update the weights of the network during training with the intention of minimizing the loss function.

Limited-memory Broyden-Fletcher-Goldfarb-Shanno (*lbfgs*) is an optimizer in the family of quasi-Newton methods [40]. *Lbfgs* is a good choice for small datasets and networks, where it can converge quickly in a single batch [51] due to its memory limitations.

Stochastic Gradient Descent (*sgd*) is a widely used optimization algorithm for training MLPs. Every iteration, *sgd* randomly selects batches of data and adjusts the weights depending on the gradients computed from these batches. It can be a good choice for large datasets as it reduces the high computational burden, achieving faster iterations in exchange for a lower convergence rate [52].

Adaptive Moment Estimation (*adam*) is a popular stochastic gradient-based optimization algorithm especially used for training deep NNs, including MLPs [53]. The optimizer is really efficient when working on large problems involving large datasets or many parameters as it requires less memory [54]. *Adam* is proposed as the most efficient stochastic optimization since it inherits the strengths of the Root Mean Square prop and the gradient descent with momentum algorithms [53].

Epochs

The number of epochs represent the number of times the learning algorithm will work through the entire training dataset. An epoch is comprised of one or more batches. The number of epochs is often high, allowing the model to run until the error has been minimized sufficiently [55].

Batches

The batch size is a hyperparameter that defines the number of samples to be processed in each iteration. In an iteration, the optimizer uses these samples to compute the gradient of the loss function and update the weights of the network [56]. Smaller batch sizes allow for more frequent network weight updates, potentially speeding up convergence. Conversely, larger batch sizes offer more data-parallelism which in turn improves computational efficiency and scalability [57].

2.3. Hyperparameter tuning

Hyperparameter tuning is a very important step in the ML process, and it involves finding the optimal set of hyperparameters that give the best model result on a given dataset. Grid search and random search are the two most common methods used for hyperparameter tuning and will be used in all models as the hyperparameters are not known in advance. Both methods involve specifying a range of values for each hyperparameter and searching for the optimal combination within that range.

In sci-kit's *GridSearchCV* [58] and *RandomsearchCV* [59] a validation set is created as a part of the cross-validation process. The training set is divided into k folds during the grid search, and a model is trained on k-1 of these folds before being evaluated on the remaining fold, which serves as the validation set. This process is repeated k times so that each fold is used as the validation set once. The average performance across all folds is then used as the evaluation metric for that set of hyperparameters. The set of hyperparameters that produced the best performance in the validation sets is chosen as the final set after all hyperparameters have been evaluated. The process is depicted below in Figure 15.

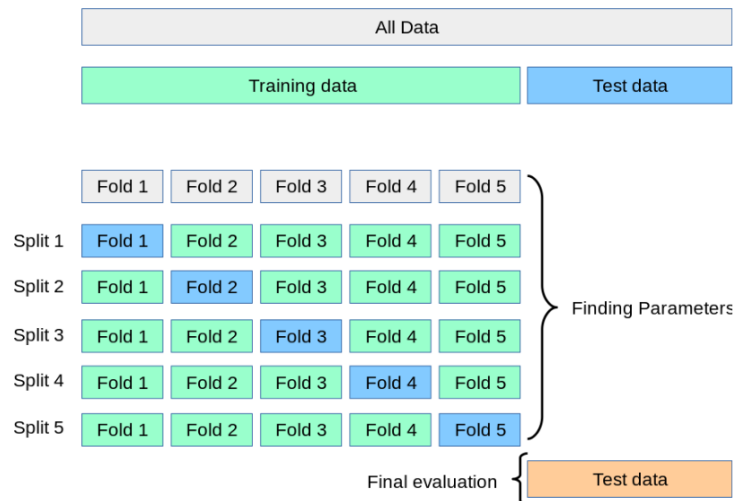


Figure 15: *k*-fold cross validation [60]

Grid search on the one hand will systematically try out every combination of hyperparameters, whereas random search will randomly sample hyperparameters from a predefined search space. The grid search can therefore be more computationally expensive and is better suited for small search spaces. In contrast, random search is more computationally efficient and suitable for large search spaces, having improved exploratory power.

Bergsta and Bengio [61] suggested that for most datasets only a few of the hyper-parameters really matter. While grid search and random search are considered robust tools for hyperparameter tuning, there is a prevailing consensus within the ML community that these techniques should not be used in isolation. Employing multiple strategies is recommended to help identify crucial hyperparameters, safeguard against overfitting and inform a more discerning decision on the optimal values tailored to a specific dataset. For a comprehensive exploration of these strategies and their application, refer to the "Hyperparameter tuning" sections in Chapters 5.2, 6.2 and 7.2 for each respective task.

2.4. Evaluation metrics

It is very important to be able to evaluate the performance of the classification models to reliably use them for solving real-world structural problems. For each task, performance metrics are used to assess how well ML models perform. They help us understand the strengths and limitations of these models when making new predictions. In this chapter, the evaluation metrics which are chosen and used will be reviewed.

2.4.1. Evaluation metrics for classification algorithms

The performance metrics we will be looking at include accuracy, precision, recall and f1-score [62]. Before explaining the various metrics we use to evaluate a classifiers performance, it is important to understand key terminologies such as true positives, false positives, true negatives and false negatives:

- True Positive (TP): Measures the degree to which the model predicts the positive class correctly. For instance, in a classification problem with classes A and B where the goal is to predict class A correctly, then true positives are the number of instances of class A that the model predicted correctly as class A . TP are relevant because they indicate how well our model performs on positive instances [62].
- False Positive (FP): FP occurs when a model incorrectly predicts that an instance belongs to a class when, in fact, it does not. FP are problematic as they can lead to poor decision-making. In some cases, FP can be desirable. For example, if our problem instead classified if a beam failed or not, it would be better to err on the side of caution and have a few FPs than to miss the beams that have failed [62]. However, in our application, where the goal is to predict the anticipated failure mode, it is important to try to keep the FP as low as possible.
- True Negatives (TN): TN are the instances that the model correctly predicts as negative. They are important to measure how well a classification model is performing. For instance, if the model is predicting whether a section fails or not because of shear, a TN would be when the model predicts that the section will not fail because of shear and the section indeed does not fail because of shear. In general, a high number of TN indicates that the model performs well [62].
- False negatives (FN): A FN occurs when the model predicts an instance as negative, when in fact it is positive [62]. For the example of classifying if a beam fails or not, FNs would be very costly as the model would for the given sections fail to identify that the beam is at risk of failing. As FNs are often more consequential than FPs, it is important to take them into account when evaluating the performance of the classification model.

TP, FP, TN and FN should be used collectively to compute metrics such as accuracy, precision recall and F1-score.

2.4.1.1. Confusion matrix

The confusion matrix is used to evaluate the performance of a classification model by comparing its correct and incorrect class predictions. It is especially useful because it gives a better idea of the model's performance than classification accuracy does, as the latter does not give any information about the misclassified instances. The size of the confusion matrix is $N \times N$, where N is the number of target classes. The matrix for a binary classification problem would look like Figure 16.

		Predicted	
		Negative (N) -	Positive (P) +
Actual	Negative -	True Negative (TN)	False Positive (FP) Type I Error
	Positive +	False Negative (FN) Type II Error	True Positive (TP)

Figure 16: Binary classification problem (2x2 matrix) [63]

A well performing model has high TP and TN rates, while low FP and FN rates. The confusion matrix plays a crucial role in evaluating the model's performance when handling imbalanced datasets. For a multiclass classification problem, the confusion matrix is extended to include more rows and columns for each class label. The figure below demonstrates what the components of such a matrix would look like for the moment class.

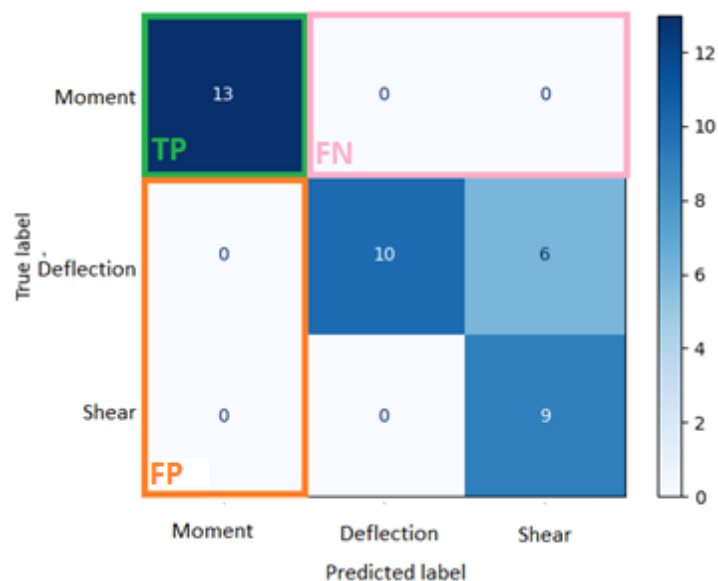


Figure 17: Example of multiclass classification problem (3x3 matrix)

2.4.1.2. Precision score

The model precision score measures the proportion of true positive predictions out of all positive predictions. It answers in other words the question “what proportion of positive identifications was actually correct?” and is also known as positive predictive value [62]. It can be thought of as a measure of quality or correctness. The precision metric is affected by the class distribution, meaning it is very useful for measuring the prediction success when the dataset is imbalanced. Mathematically, the precision score can be expressed as:

$$\text{Precision Score} = \frac{TP}{FP + TP} \quad (10)$$

From the above formula, one can quickly notice that the value of FP impacts the precision score. Thus, if a high precision score is important for the classification requirements, models with lower FP values would be vital.

2.4.1.3. Recall score

The model recall score measures the proportion of true positive predictions out of all actual positives that exist within a dataset. It answers the question “what proportion of actual positives was predicted correctly?”. The metric is also called sensitivity or the true positive rate [62]. The higher the recall score, the better the model performs at identifying both positive and negative instances. Mathematically, the recall score can be expressed as:

$$\text{Recall Score} = \frac{TP}{FN + TP} \quad (11)$$

From this formula one notices that the value of FN impacts the recall score. Thus, while building a predictive model, one should focus on keeping the number of FN as low as possible to get a good recall score.

2.4.1.4. Precision-Recall tradeoff

The precision-recall tradeoff is a common issue that arises when trying to evaluate the performance of a classification model. Precision is a metric that measures the proportion of TP predictions made and is useful for evaluating the model’s ability to avoid FP. Recall on the other hand, measures the proportion of TP instances that were correctly predicted, and is a useful metric for evaluating the model’s ability to avoid FN. Generally, increasing one of the two metrics will decrease the other, because precision and recall are inversely related. So, a model with good

precision will make few FP predictions, but will probably also miss some TP, and vice versa. Therefore, the appropriate balance between precision and recall will depend on the problem's requirements and goals, as well as the characteristic of the dataset [62].

For the example of identifying beams that will fail, optimizing for recall will help to minimize the chance of not detecting a beam that fails. However, this comes at the cost of predicting failure in a beam for which capacity can withstand the load. Optimizing for precision would help predict more accurately if the beam is going to fail or not. However, this would come with the cost of missing failure in some beams.

2.4.1.5. F1-score

Model F1-score assesses the model score as a harmonic mean of precision and recall score. The metric gives equal weight to both the precision and recall for measuring the performance of the model in terms of accuracy, making it an alternative to the accuracy score, which we will look at below. It is useful in a scenario where one tries to optimize both precision and recall, but does not want to rely solely on either [62]. However, though F1-score balances precision and recall, it is not a perfect solution as it can be difficult to determine the optimal balance between the two metrics for a given application. Mathematically, the F1-score can be expressed as:

$$\text{F1 Score} = \frac{2 * \text{Precision Score} * \text{Recall Score}}{\text{Precision Score} + \text{Recall Score}} \quad (12)$$

2.4.1.6. Accuracy score

The model accuracy metric is defined as the ratio of TP and TN to all positives and negative observations. In other words, it measures the proportion of correct predictions out of the total number of predictions. One should be cautious relying on accuracy score alone when evaluating the model performance as it does not consider the relative importance of different types of errors such as FN or FP. The accuracy metric is also not reliable for models trained on imbalanced datasets. For instance, if the dataset consists of 95% of class A, the accuracy of the classifier will likely be very high as it will correctly predict class A most of the time. A better classifier that deals with the class imbalance will most likely exhibit a worse accuracy score.

2.4.1.7. Macro and Weighted -average

In multi-class classification performance evaluation, macro and weighted average are two common ways to aggregate the evaluation metrics.

Macro-averaging

Calculates the metric independently for each class and then takes the average across all classes. This way each class is given equal importance regardless of its frequency in the data. This is useful for when the dataset is imbalanced, as it ensures that the model's performance is evaluated for each class equally and can help identify classes that the model struggles to classify correctly.

Weighted-averaging

Calculates the metric for each class, then multiplies it by a corresponding weight before including it in the average calculation. The weights are determined based on the frequency at which the classes occur. This can provide a more accurate representation of the model's performance on the dataset as a whole.

In conclusion, when working on an imbalanced dataset where the classes are equally important, the use of macro average would be a good choice, while if the performance of the minority class is not that important by assigning greater contribution to the majority classes, weighted average is preferred [64].

2.4.2. Evaluation metrics for regression algorithms

A set of evaluation metrics has been employed to assess the performance of our regression algorithms. These metrics serve as a tangible measure of the predictive capabilities of our models.

2.4.2.1. Mean Squared Error and Mean Absolute Error

A key metric in regression is the Mean Squared Error (MSE), a loss function that measures the average of squared differences between the predicted and actual values. Given its straightforwardness and popularity, MSE is often utilized for error estimation in regression tasks. The mathematical formulation is shown below [65]:

$$MSE = \frac{1}{n} \sum_{i=1}^n (p_i - r_i)^2 \quad (13)$$

where,

n is the total number of observations (data points)

r_i is the actual value of an observation

p_i is the predicted value

Notably, the MSE always remains non-negative due to the squaring operation, which significantly weighs outlier predictions with large errors, proving useful in outlier detection. However, MSE might not always provide a perfect reflection of the model's performance. A single significant prediction error can sometimes obscure the true performance, as it may contribute heavily to the overall error estimate [65]. Therefore, it is ideal to use MSE when large errors bear more importance than smaller ones.

An alternative error estimation function, Mean Absolute Error (MAE), addresses some drawbacks of using MSE. One significant advantage of MAE is that it is less sensitive to outliers than MSE due to linearly weighing the errors, rather than squaring them. This means the MAE provides a consistent measure of model error, which is beneficial in scenarios where extreme errors are not significantly more critical than smaller ones. The mathematical formulation is shown below [66]:

$$MAE = \frac{1}{n} \sum_{i=1}^n |p_i - r_i| \quad (14)$$

However, the MAE does not explicitly prioritize large errors. In some situations, this could be a disadvantage, as substantial errors are treated equivalently to minor ones. As such, MAE is typically a useful metric in many contexts but may be less effective where substantial errors have far greater implications. Using both MSE and MAE together can provide a comprehensive perspective on model performance, as each metric highlights different aspects of prediction errors. This complementary use helps to identify the model's strengths and weaknesses more effectively.

2.4.2.2. R-squared

R-squared, also known as the coefficient of determination, measures the proportion of the variance in the dependent variable that can be explained by the independent variables in a regression model. It is a useful metric for assessing how well the model fits the data. An R-squared of 1 indicates that all changes in the dependent variable are completely explained by changes in the independent variable(s). Conversely, an R-squared of 0 indicates that none of the variability in the dependent variable can be explained by the independent variable(s). The mathematical formulation is [67] [68]:

$$R^2 = 1 - \frac{\sum_{i=1}^n (r_i - p_i)^2}{\sum_{i=1}^n (r_i - \bar{r})^2} \quad (15)$$

where,

\bar{r} is the mean (average) of all actual values

However, it is important to note that adding more independent variables to the model can artificially inflate the R-squared value, even if those variables do not contribute to the model's predictive accuracy.

2.4.2.3. Graphical analysis

While the measures above are significant, they may not encapsulate the full story of model performance. Complementary to these metrics, graphical analyses can reveal underlying trends and patterns not immediately evident in numerical results. In tasks 2 and 3 (Chapter 5 and 6), we have utilized such visual explorations to enhance our understanding and interpretation of the model's outcomes, thereby offering a more comprehensive evaluation.

The deviations between the predicted and actual values for a test set can be graphically represented as shown in Figure 18. The black line signifies an ideal scenario, where predicted and actual values align perfectly. For these plots, a model's performance can be interpreted by how closely the data points cluster around the ideal line. Individual data points, denoting the predicted values, are superimposed on the same line for the SVR and MLR models, to clearly showcase the differences and variations unique to each algorithm. This visual representation effectively supplements our understanding of the discrepancy between predicted and actual values, thereby enhancing our performance evaluation.

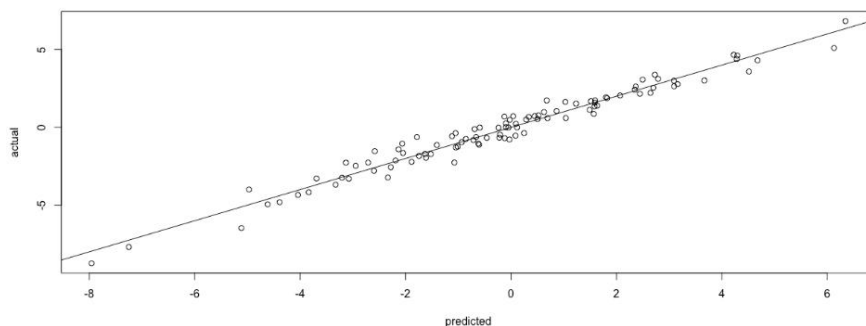


Figure 18: Example of a predicted vs. actual value plot [69]

The final step in our evaluation involves examining the residual plots. Residuals, defined as the difference between the actual and predicted values, are graphically represented against the ideal line, which denotes a residual value of zero. A critical purpose of these plots is to detect potential

anomalies and assess the performance of the regression model. Specifically, they can reveal heteroscedasticity (non-constant variance in the error terms), identify outliers, and check the linearity and independence assumptions associated with most linear regression models [70].

Each data point on these plots can exhibit both positive and negative residual values, indicating underprediction or overprediction by the model, respectively. It is important to note that perfect prediction is a rarity due to the inherent randomness and unavoidable inaccuracies in real-world data. Therefore, optimal regression models aim to capture major trends and predictive information in the dataset, leaving only small, independent, and normally distributed residuals.

In a well-performing residual plot, the data points are symmetrically scattered around the ideal line and in close proximity to it, indicating the randomness of the residuals and the model's success in capturing the key information in the data. Conversely, a plot exhibiting patterns in the residuals or a concentration of residuals in certain areas could point to a model that has failed to capture some inherent patterns in the data.

The figures below illustrate the difference between an ideal and a suboptimal residual plot. Figure 19a showcases an ideal scenario where the data points are evenly distributed around the ideal line. In contrast, Figure 19b depicts a less optimal situation, where the majority of data points lie above the x-axis, indicating a systematic underprediction by the model.

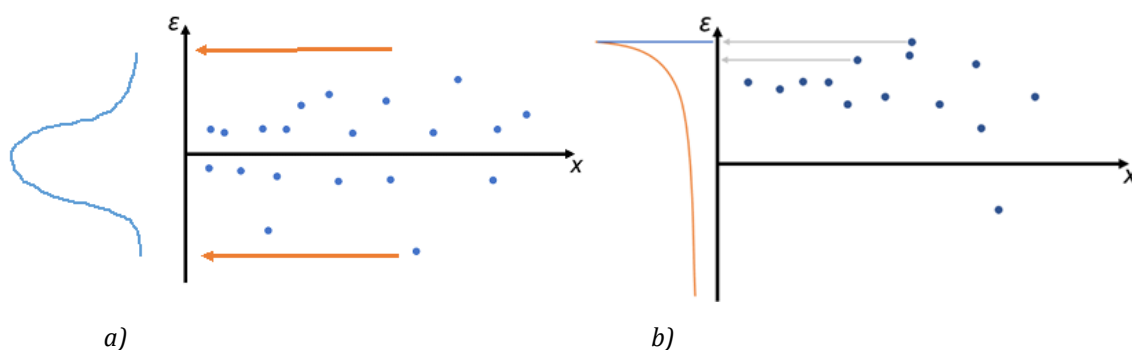


Figure 19: Visualization of a) ideal residual plot and b) suboptimal residual plot [71]

If a model underestimates an observation, the model estimate will be smaller than the actual value. The residual, which is the actual observation value minus the model estimate, will consequently be positive (over the x -axis). On the contrary, when the model overestimates the observation, the residual will be negative (under the x -axis).

3. Methodology

3.1. Research design and overall approach

In this master’s thesis, we aim to explore the potential of ML as a useful tool in the design of RC structures, with a focus on rectangular and T-section beams. The research comprises three main tasks, each addressing different aspects of simply supported RC beam design: (1) predicting anticipated capacity failure mode, (2) predicting load capacity, and (3) cost optimization of T-section design.

Although three different tasks, the flow of our research methodology can be summarized in the following flow chart in Figure 20.

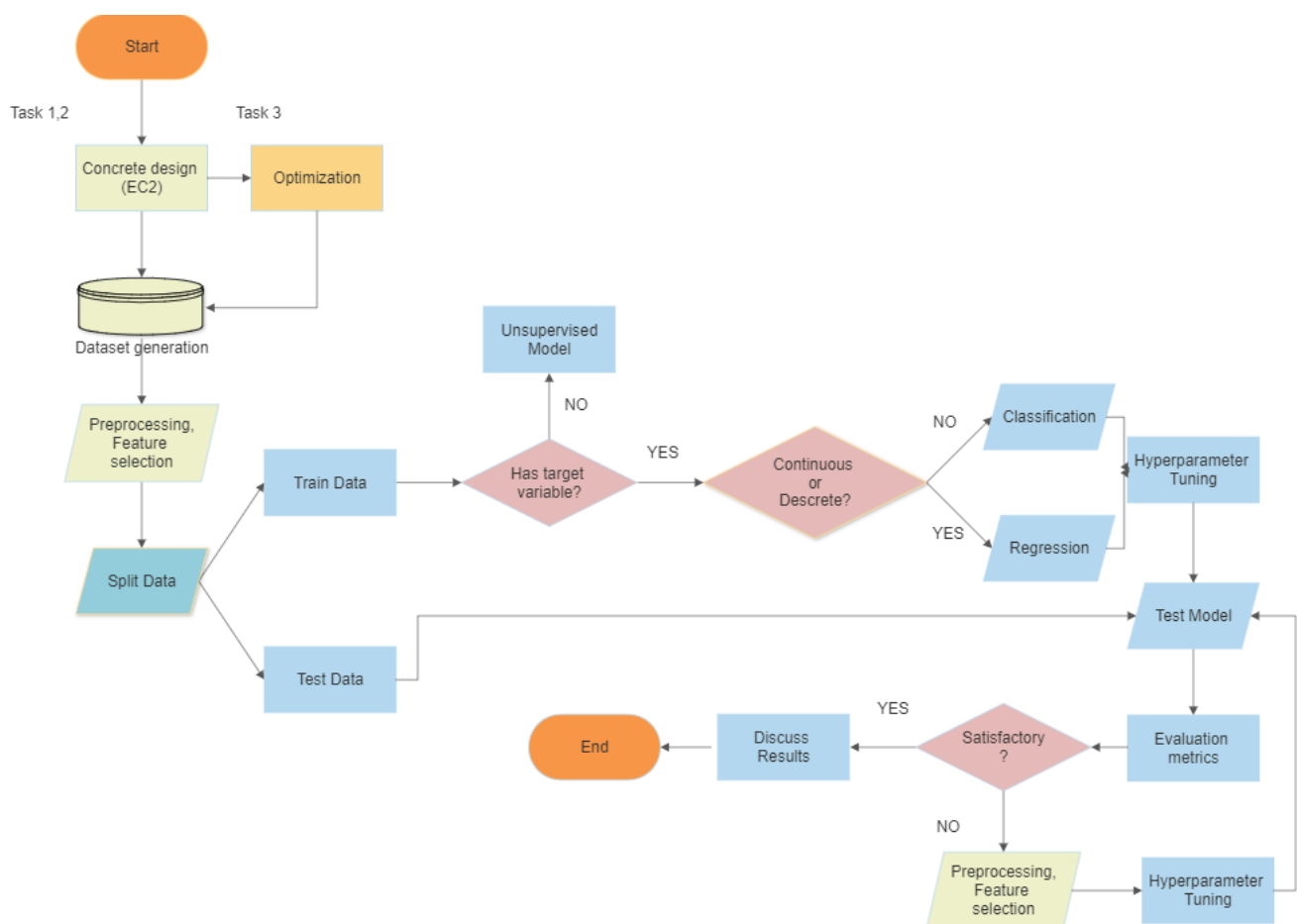


Figure 20: Methodology flow chart

The flowchart begins with the concrete design and dataset generation for each task. An additional step is added to Task 3 which cost-optimizes the T-section design during the data generation.

After preprocessing and feature selection, the data is split into training and test sets. The next step in the process is to determine if our task has target variables. Upon confirming the existence of target variables for all tasks, we were able to proceed with supervised machine learning models. Based on the nature of the target variable (continuous or discrete), the appropriate machine learning models (regression or classification) are selected and hyperparameters are tuned. The models are then tested using the test data, and their performance is evaluated using relevant metrics. If the results are not satisfactory, the process iterates, returning to preprocessing, feature selection, and hyperparameter tuning. This iterative process continues until satisfactory results are achieved, after which the results are discussed, concluding the research process. The flowchart illustrates the systematic and iterative nature of our research methodology, highlighting the critical decision-making steps and the importance of model evaluation and improvement.

Our research can be characterized as a combination of applied, exploratory and quantitative. It is applied in nature as it addresses practical problems in structural engineering and aims to enhance engineering practices using ML techniques. The research is exploratory, particularly given that we had no prior experience in ML and limited understanding of python, before undertaking the thesis. This lack of prior knowledge adds an element of exploration, as we navigated through various ML models and techniques, while seeking to uncover new insights and understandings that can contribute to future research direction in the field of structural engineering, but also provide a foundation for subsequent studies. Lastly, the research is quantitative, relying on unchanging numerical data, employing quantitative methods such as ML methods and performance metrics, to draw conclusions from the data.

3.2. Materials and software

The calculations used to produce the datasets are based on the rules specified in EC2 and EN 1990: Eurocode - Basis of structural design (EC0) [72]. All the numerical analyses were performed using python in the Spyder environment [73]. The ML models were implemented using Scikit-learn [74] and TensorFlow with the API Keras [42]. In this research, the SciPy.optimize [75] module was utilized for generating the dataset for the optimization task. Various Python packages have also been used for different tasks, such as:

- Visualization: Seaborn, Yellowbrick, Matplotlib and Dtreeviz
- Preprocessing: Imbalanced-Learn (imblearn) and Pandas
- Calculations: NumPy, Math and Statsmodels

Spyder

Spyder is a free and open-source scientific environment specifically designed for engineers and data analysts. It offers an intuitive interface, excellent visualization and debugging capabilities, seamless integration with popular libraries, and an active community for support.

Scikit-learn

Scikit-learn (formerly scikits.learn and also known as sklearn) is an open-source Python library that provides a wide range of tools for ML and statistical modeling. It includes algorithms for classification, regression, clustering, dimensionality reduction, and model selection, as well as preprocessing functions and evaluation metrics. Scikit-learn is largely written in python and built on NumPy, SciPy and Matplotlib.

Keras

Keras is an open-source, high-level API written in Python. It serves as an interface for the TensorFlow platform and primarily focuses on deep learning and NNs. Keras includes numerous implementations of commonly used NN building blocks, such as layers, objectives, activation functions, and optimizers.

SciPy.optimize

SciPy.optimize is a module within the SciPy library that offers a variety of functions for optimizing objective functions, either through minimization or maximization, with the ability to handle constraints. The module includes solvers for a wide range of problems, including nonlinear problems, linear programming, constrained and nonlinear least-squares, root finding and curve fitting.

3.3. Reflection and quality assurance

3.3.1. Validity

In the context of our research, validity is defined as the degree to which our methodologies, models, and findings accurately reflect the real-world situations they intend to simulate or predict. It is crucial to the overall strength of our study and concerns the credibility and integrity of our findings. For our thesis, we consider validity in several key aspects:

- **Consistency with established standards:** Our research adheres to established EC2 and EC0 design rules, ensuring that our methodologies align with recognized standards in the field of structural engineering. This provides a solid foundation for the validity of our work.

- **ML models:** The validity of our ML models is demonstrated through their ability to effectively predict outcomes on unseen data. This is evaluated using well-established metrics, as outlined in Chapter 2.4, and through the comprehensive analysis presented in Chapters 5, 6, and 7.
- **Rapidly evolving field:** Machine Learning is an ever-evolving discipline, with new methodologies, algorithms, and libraries being developed and updated regularly. While this dynamism could potentially challenge the longevity of our work's validity, we have taken steps to use the most current and widely accepted methods and tools at the time of our research. We also provide thorough documentation and transparency in our use of these tools to ensure validity.
- **Secondary analysis of design rules:** In the optimization part of our study, we conducted a secondary analysis of the paper by Fedghouche [76]. This approach saved considerable time, allowing us to focus on the ML analysis of the data it produced. While the adoption of these design rules from a secondary source could be viewed as a potential threat to validity, we consider the robustness and credibility of the original source sufficient to uphold the validity of our work.
- **Coding practices:** As beginners in coding, our primary focus was to ensure that the code functioned as intended and produced the desired results. However, given our relative unfamiliarity with coding, we have not prioritized the optimization or efficiency of the code.

3.3.2. Generalizability

A key concern of quantitative research is the extent to which findings can be generalized beyond the context of their study -in other words, whether the research has sufficient external validity [77]. In the context of our thesis, the generalizability might be considered in terms of the following aspects:

- **Applicability of design rules:** The design rules and principles used to generate the various datasets are based on EC2 and EC0. While this ensures that the data is reliable, consistent, and applicable to a wide range of scenarios in structural engineering, it also introduces a certain degree of restriction. The applicability of our findings is primarily confined to contexts and regions where European standards are implemented. Therefore, the generalizability of our research to scenarios or regions that employ different design standards may require additional considerations or adaptations.

- **Transferability to other structures:** Although our study focuses on rectangular and T-sectioned RC-beams, the ML methodologies and principles are applicable to other types of structures.
- **Diversity of the datasets:** The more diverse a dataset is, the more generalizable our results are likely to be. However, due to computational limitations, we had to enforce restrictions on the range of properties for the sections, setting lower and upper boundaries. In context, generating the dataset for the optimization task took approximately four hours. While this process could potentially be expedited through various cloud services, access constraints prevented their use in this thesis. The lower bound limitations are likely representative of lower scale structures such as residential buildings or small commercial buildings, while the upper bound limits have been set to encapsulate the characteristics of beams found in larger scale structures, including high-rise buildings, bridges, and other major infrastructural projects.

For the cost optimization of Task 3, the cost ratios seen in Chapter 4.3 are the ones proposed by Fedghouche (2018) [76], and may be different depending on the geographical location, economic situation, or specific project requirements.

While we believe that the diversity of the data adequately represents a substantial range of cases, we acknowledge that it may not cover all possible scenarios. The specific limits and cost ratios applied in the tasks are detailed in Chapter 4, and can be readily adjusted in the code of Appendices A2 and I, to accommodate unique project requirements. Despite the noted constraints, we maintain confidence in the generalizability of our study given the breadth of conditions encapsulated within our data boundaries.

- **Model performance on unseen data:** Although the ML models are trained on a specific dataset, they should perform reasonably well on new, unseen data. The results in Chapters 5, 6 and 7, as well as the steps taken to hinder overfitting, suggest that the models have learned general patterns and principles that apply beyond the specific examples they were trained on.

3.3.3. Replicability

Replicability is an important aspect of scientific research and refers to the ability of independent scientists to reproduce each other's experiments. Even though the datasets are not provided in the thesis, this research has taken some steps to ensure that others can replicate our study, or even build upon it.

- **Data generation:** The underlying design rules and methodologies, as well as detailed explanation of how the datasets were created are outlined throughout the thesis.
- **Exact parameter settings:** The composition and settings of the model's hyperparameters are thoroughly discussed within the thesis. Additionally, boundary conditions and restrictions are also covered and can be viewed in the accompanying Python scripts, which are included in the Appendix, for increased transparency and accessibility.
- **Evaluation metrics:** Chapter 2.4 provides a clear definition and explanation of the performance metrics used in the assessment of our models.
- **Software and libraries:** The specific software tools and libraries used in this study are detailed, enabling others to replicate or build upon our results using the same resources.
- **Randomness in ML models:** ML models often incorporate elements of randomness. This randomness can lead to variations in model performance even when trained on the same data. To mitigate the impact of this, we used fixed seeds for the random number generators in our code to ensure that our results could be reproduced exactly. However, because ML models often rely on optimization algorithms, they do not always find the exact same solution in every run, even with the same initial conditions. Therefore, the precise outcomes may not be identically reproducible in every replication attempt. It is therefore important to interpret the results in terms of their general trends and comparative performances rather than focusing solely on specific numeric values.

4. Dataset generation for RC beam design

In this section, we will provide a comprehensive overview of the design rules and processes that were employed to create the datasets for each of the three tasks in this research. Central to this is the utilization of the Python scripts in Appendices A and I. Understanding the quality of the datasets is crucial, as the performance and outcomes of the algorithm's heavily rely on the data on which they are trained on. Our research utilizes a total of four datasets:

- One for Task 1, where the primary output is the anticipated failure mode.
- Three for Task 2, with primary outputs being moment capacity, shear capacity, and load capacity.
- One for Task 3, where the primary outputs are the design variables and cost of optimal sections.

The appropriate sample size for a dataset is a topic that sparks much debate. The subjective nature of the decision, along with constraints tied to the availability of samples, makes it a complex issue. As suggested by [78], the minimum sample size for general engineering problems should be 3-20 times the number of variables involved. In the context of ML regression, it has been recommended that the sample size should be 10 times the number of independent input variables [10].

Balancing the representation of a broad range of cross-sections against the feasibility of experimental data, we chose to use 400, 600, and 378 training samples for Tasks 1, 2, and 3, (Chapters 5, 6 and 7) respectively. This decision aimed to create a reasonable yet diverse range of samples for training our ML models.

Given the relatively lower computational time required for generating samples for Tasks 1 and 2, we leveraged the opportunity to test the predictability of the ML models on approximately 1500 unseen sections, thereby offering a robust representation of the models' generalizability. Consequently, the data division for training and testing for Tasks 1 and 2 followed a 20-80 and 30-70 ratio respectively.

Contrastingly, Task 3's dataset generation, which included an optimization process, was more computationally intensive. We managed to produce a total of 540 sections and subjected the ML model to 162 unseen sections for testing and evaluation. This 70-30 data division, a common

practice when training algorithms, ensures a robust training process while providing a meaningful set of data for testing and validation.

In the next chapters, the EC2 design rules used to generate the datasets for Task 1 and Task 2 will be presented, focusing on rectangular, simply supported RC beams. We will then describe how the input data was adjusted to suit the requirements of regression and classification tasks. Finally, the design rules and optimization process used to create the dataset for Task 3 will be shown, which focuses on T-sections for simply supported RC beams.

4.1. Limit state design of RC sections under bending

EC0 and EC2 form the foundation for the design rules utilized in constructing the datasets. Ultimate Limit State (ULS) was employed for designs requiring the analysis of bending moments and shear forces, while Serviceability Limit State (SLS) was used for designs related to deflection assessment. In the context of ULS, we employed the load factors as defined in sections 6.10a and 6.10b of EC0, opting for the worst-case scenario. Conversely, for the SLS, a uniform load factor of 1.0 was applied across all loads. A long-term perspective was used to calculate the deflection, and a ψ -value of 0.3 was applied. In compliance with EC2, the assumptions for the limit state of a typical reinforced beams are depicted in Figure 21a-c.

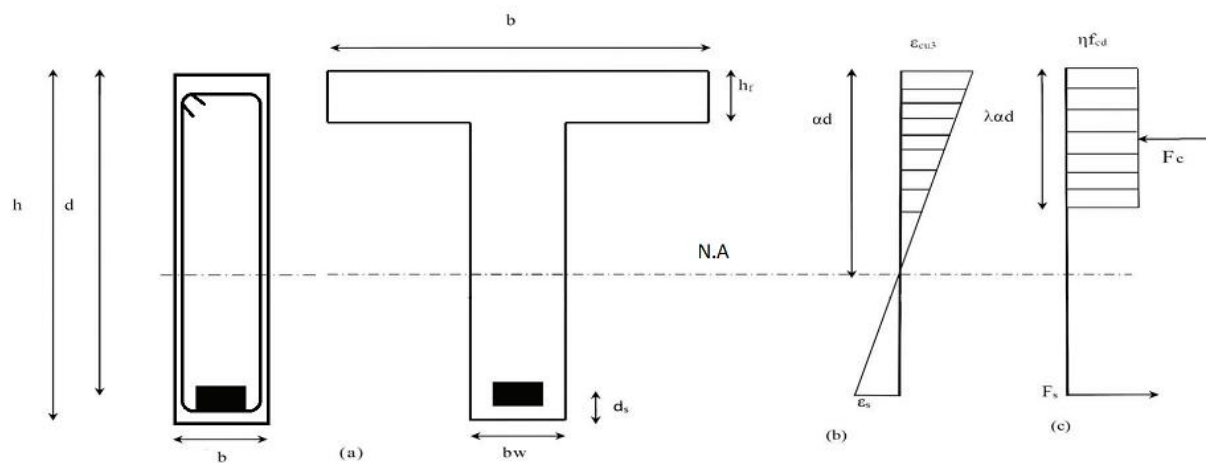


Figure 21: a) Rectangular and T-beam section; b) strains at ultimate limit state and c) stresses at ultimate limit state [76].

ϵ_s and ϵ_{cu3} as shown in Figure 21b's linear strain diagram represents the steel strain and the ultimate strain in concrete under compression, respectively. The parameter α indicates the relative depth of the compressive zone, and the plastic neutral axis is situated at a distance of $\lambda \alpha d$

from the upper fiber. In the uniformly distributed stress diagram, shown in Figure 21c, f_{cd} is the design value of concrete compressive strength.

According to EC2, the option to work with a rectangular stress distribution is available, by introducing the factors λ and η for the depth of the compression zone and design strength, respectively. Both λ and η factors have a linear dependency on the characteristic strength f_{ck} , as described in the following equations [2]:

$$\lambda = 0,8 + \frac{f_{ck} - 50}{400} \quad (16)$$

$$\eta = 1,0 + \frac{f_{ck} - 50}{200} \quad (17)$$

These equations are applied for $50 \leq f_{ck} \leq 90$ MPa, while $\lambda = 0.8$ and $\eta = 1.0$ for $f_{ck} \leq 50$ MPa. F_c and F_s represent the resultants of internal forces in the section and reinforcing steel, respectively. The design yield strength of steel reinforcement is defined as f_{yd} . Moreover, in accordance with EC2 provisions, steel strain is considered unlimited [76]. The maximum elastic strain ε_{yd} corresponding to design yield strength of steel reinforcement can be defined from the relationship $\varepsilon_{yd} = f_{yd}/E_s$, where E_s denotes the elasticity modulus of steel. For the optimal utilization of steel in this task, the strain must always be greater than or equal to the elastic limit strain.

4.2. Task 1 and 2: Rectangular RC beams

Bending moments often pose a weakness for beams, particularly those with large spans and small heights, as they can lead to catastrophic failures. The bending moment (M_{Ed}) for a simply supported beam is located at the midspan and is calculated as:

$$M_{Ed} = \frac{q L^2}{8} \quad (18)$$

Where q is the total load in ULS per meter and L is the beam's length.

The moment capacity calculation, as outlined in Sørensen's book [79], is based on the calculations from EC2. The balanced reinforcement amount is calculated as:

$$A_{s_{bal}} = \frac{f_{yd} A_s}{\lambda n f_{cd} b d \alpha_{bal}} \quad (19)$$

where,

$$\alpha_{bal} = \frac{\varepsilon_{cu}}{\varepsilon_{cu} + \varepsilon_{yd}} \quad (20)$$

It is important to determine whether the cross section is over or under -reinforced, to calculate the moment capacity. To do so, we calculate the balanced reinforcement in Eq. (19) and compare it to the input reinforcement of the section. If the cross section is under-reinforced, α is calculated as:

$$\alpha = \frac{f_{yd} A_s}{\lambda n f_{cd} b d} \quad (21)$$

And if the cross section is over-reinforced, α is calculated as follows:

$$\alpha = \frac{-(E_s A_s \varepsilon_{cu}) \pm \sqrt{(E_s A_s \varepsilon_{cu})^2 - 4 (\lambda n f_{cd} b d) (-E_s A_s \varepsilon_{cu})}}{\lambda n f_{cd} b d} \quad (22)$$

Since the α for over-reinforced cross sections is calculated using the quadratic formula, two solutions arise. In the script we ensure the correct value is selected by requiring it to be greater than α_{bal} and less than 1.0. The moment capacity (M_{Rd}) is then calculated as:

$$M_{rd} = 0,8 \alpha (1 - 0,4 \alpha) f_{cd} b d^2 \quad (23)$$

Beams that are relatively short in length tend to have shear forces as a weak point when it comes to failure. In a simply supported beam, the shear force (V_{Ed}) in each cross section is calculated as:

$$V_{Ed} = \frac{q L}{2} \quad (24)$$

In determining the shear capacity, we calculate both the concrete shear capacity (V_{Rdc}) and the shear reinforcement capacity (V_{Rds}). This is done because, in some instances, the concrete's inherent strength can exceed the capacity provided by the shear reinforcement. The concrete shear capacity is calculated as follows:

$$V_{Rdc} = C_{rdc} k (100 \rho_l f_{ck})^{\frac{1}{3}} b d \quad (25)$$

with a minimum of

$$V_{min} = 0,035 k^{\frac{3}{2}} f_{ck}^{\frac{1}{2}} \quad (26)$$

The shear reinforcement capacity is calculated as the lowest value of Eq. (25) and (26):

$$V_{Rds} = \frac{A_{sw_div_s}}{100} z f_y d \cot(\theta) \quad (27)$$

$$V_{Rd,max} = a_{cw} b z v_1 \frac{f_{cd}}{\cot(\theta) + \tan(\theta)} \quad (28)$$

where θ is assumed to be 90 degrees for vertical stirrups.

Reinforced concrete beams naturally exhibit deflection over time. This deflection, if excessive, can not only be visually off-putting, but also cause discomfort to occupants and potentially damage supported structures. Therefore, it is crucial to manage and control deflection carefully. The long-term behavior of these beams is influenced by factors such as flexural cracking, thermal effects, and deflections caused by shrinkage and creep. However, for the scope of this thesis, the influence of thermal effects has been excluded, and we have assumed a concrete class S. The research has factored in deflection control at mid-span, as an essential aspect of RC design. The maximum allowed deflection is set as:

$$w_{allowed} = \frac{L}{250} \quad (29)$$

Where L is the span of the beam.

Shrinkage, expressed as a percentage, refers to the proportional change in length per unit length. It comprises two components - an autogenous part:

$$\varepsilon_{ca} = \beta_{as} \varepsilon_{ca,inf} \quad (30)$$

and a drying-induced part:

$$\varepsilon_{ds} = \beta_{ds} k_h \varepsilon_{cd0} \quad (31)$$

where,

$$\varepsilon_{ca,inf} = 2,5 (f_{ck} - 10) \quad (32)$$

$$\varepsilon_{cd0} = 0,85 ((220 + 11 + \cdot a_{ds1}) e^{-a_{ds2} \cdot \frac{f_{cm}}{f_{cm0}}} \beta_{RH} \quad (33)$$

The two parts are then added together in order to find the curvature due to shrinkage as shown below:

$$\kappa_s = (\varepsilon_{ds} + \varepsilon_{ca}) \eta As \frac{e}{I} \quad (34)$$

The moment of inertia (I) is calculated as:

$$I = \frac{b h^3}{12} + Ac \left(a - \frac{h}{2} \right)^2 + \eta As e^2 \quad (35)$$

where,

$$a = \frac{Ac \cdot 0,5 h + \eta As d}{Ac + \eta As} \quad (36)$$

Finally, the deflection due to shrinkage can be found as:

$$\delta_{shrinkage} = \frac{\kappa_s L^2}{8} \quad (37)$$

Creep is the time-dependent deformation due to persistent loads. It requires the calculation of an average modulus of elasticity (E-modulus). This E-modulus accounts for both the immediate and delayed effects of bending moments. It is determined using the short-term and long-term parts of the bending moments along with their respective short-term and long-term E-modulus, as shown in the following formula:

$$E_m = \frac{M_{g1} + M_{g2} + M_{g3}}{\frac{M_{g1}}{E_{cL1}} + \frac{M_{g2}}{E_{cL2}} + \frac{M_{g3}}{E_{cm}}} \quad (38)$$

The calculations proceed by assuming a cracked cross section (State 2), which requires the use of specific formulas. Following a series of computations, the contributions of the concrete and reinforcement to the modified moment of inertia are represented as I_{c1} and I_{s1} , respectively. These values are used to calculate the long-term stiffness:

$$EI = E_m I_{c1} + Es I_{s1} \quad (39)$$

Finally, the deflection due to creep can be calculated as:

$$\delta_{creep} = \frac{(5 (g_1 + g_2) L^4)}{384 EI} \quad (40)$$

To ensure that the dataset comprises only cross sections with permissible configurations, it is crucial that they meet the minimum requirements for shear and longitudinal reinforcement according to EC2. The minimum longitudinal reinforcement is calculated according to Eq. (41).

$$As_{min} = \text{maximum of } 0,26 \frac{f_{ctm}}{f_{yk}} b d \quad \text{and} \quad 0,0013 b d \quad (41)$$

Regarding the minimum shear reinforcement, the script compares the shear reinforcement of each cross section with both the maximum allowed stirrup spacing and the required amount of shear reinforcement per meter. The maximum spacing is calculated as per the following formula:

$$s_{max} = \text{minimum of } s_1 \text{ and } s_2 \quad (42)$$

where,

$$s_1 = 0,6 d (1 + \cot(\alpha)) \quad (43)$$

$$s_2 = \text{minimum}(d, 600) \quad (44)$$

Another important consideration in creating a dataset with realistic and feasible cross sections is ensuring sufficient space for the longitudinal reinforcement within the cross section. For instance, placing 4000 mm^2 of rebar within a 100×100 section would be impractical and would skew the thesis results. To address this issue, the script implements certain limitations. First, to simplify the script, a maximum of six rebars with diameters ranging from 16 - 32 mm is set. The script associates different ranges of reinforcement amounts with appropriate rebar diameters. If the spacing in the width is insufficient, the script organizes the rebars into two layers in the vertical direction. If the cross section still lacks sufficient space, it is deemed unsuitable for the dataset. Required space in the width with one and two layers is calculated based on the rules of EC2 where minimum distance between rebars, the diameter of the longitudinal and shear reinforcement and the concrete cover are considered.

4.2.1. Dataset generation for Task 1 and 2

The datasets made for Tasks 1 and 2 (Chapters 5 and 6) were formatted as a CSV file with six columns, five of which represent independent variables and one the dependent variable. The independent variables fluctuate within their respective ranges, resulting in a wide variety of cross section combinations in the datasets. Table 1 displays the independent variables together with their upper and lower bounds. All other variables, not included in the datasets but necessary for calculating values within them, are assigned fixed values. These can be adjusted by the user in the script of Appendix A1. For instance, f_{ck} is utilized to calculate the moment capacity, while E_c is used to determine the deflection due to creep.

Table 1: Overview of the independent variables' upper and lower limits and their ranges for the datasets in Task 1 and 2

Variable		Range	Increment Size
A_s	Longitudinal reinforcement amount (mm^2)	100-4825	100
L	Span of beam (m)	1-12	1
b	Width of cross section (mm)	100-1000	100
h	Height of cross section (mm)	100-1500	100
$A_{sw_div_s}$	Shear reinforcement amount in (mm^2/m)	0,1-2	0,1

The independent variables in the CSV file are A_s , L , h , b , and $A_{sw_div_s}$. The datasets' dependent variables are selected based on the tasks' intended findings, which are moment capacity, shear capacity, load capacity, and the anticipated failure mode.

The creation of the CSV files was accomplished through a series of steps. Initially, all the checks and formulas associated with Chapter 4.2 were transformed into functions. These functions were then compiled into a separate Python file (Appendix A1). This approach not only conserved space but also facilitated cleaner setup, allowing for the reference of that file whenever the various functions were needed.

Furthermore, cross sections were created, for which a separate file was established (Appendix A2). This file allows the user to easily adjust various fixed input variables such as f_{ck} and E_c . The minimum and maximum values of the independent variables, along with their interval sizes, are also stored in this file. These input variables form the foundation and ensure the diversity of the datasets.

The construction of the dataset itself takes place after these variables. The script first creates an empty list, then uses several "for-loops" to append cross sections, each with a unique combination of independent variables. The final and most crucial component of the dataset, the dependent variable, is then found.

For the two datasets in Task 2, which contain shear and moment capacity, the script simply iterates through the list of created cross sections, extracts the variables needed for the specific function (shear capacity or moment capacity), and adds the value to a new empty list. However, to calculate the load capacity, the script needs to iterate through a "for-loop" with an increasing applied live load, as most cross sections do not fail solely due to self-weight.

This principle also applies to the dataset in Task 1, where the anticipated failure mode for each cross section is determined by the utilization ratio that first reaches 1.0, whether it is the shear, bending, or deflection ratio. It is important to highlight that all cross sections across all four datasets incorporate functions associated with design rules regarding rebar spacing and minimum reinforcement.

Finally, once all the necessary data for the datasets has been compiled into lists, another Python file is employed (Appendix A3). This file retrieves these lists from the previous files and utilizes the *pandas* library to generate a CSV file. Given the selected range and interval sizes of the

independent variables, approximately 678,000 cross sections were created. However, as the desired number of cross sections for training is around 500, it was essential to significantly reduce the size of the CSV file. Consequently, another Python file was created (Appendix A4) to randomly delete cross sections until the total count in the dataset was reduced to 2001. This random deletion ensured that the dataset did not exclusively lose cross sections from either the beginning or the end, but rather distributed the deletion throughout the dataset.

4.3. Task 3: RC T-section Optimization

The design variables selected for the optimization of T-sections are presented in Table 2.

Table 2: Definition of design variables

Design variables		Lower bound	Upper bound
b	Effective width of compressive flange	100	3000
b_w	Web width	100	1500
h	Total height	L/16	1500
d	Effective depth	0,9h	0,9h
h_f	Flange depth	150	1500 ¹
A_s	Area of tension reinforcement	100	10000
α	Relative depth of compressive concrete zone	0.1	α_{lim}^2

The objective function for the cost optimization comprises the costs of concrete, steel and formwork per unit length of the beam. This function can be defined as:

$$\frac{C_0}{L} = C_c (b_w h + (b - b_w) h_f) + C_s A_s + C_f [b + 2h] \rightarrow \text{Minimum} \quad (45)$$

Where $\frac{C_0}{L}$ is the total cost per unit length, while C_s , C_c and C_f are the unit costs of steel, concrete and formwork, respectively. The cost function to be minimized can be written as follows:

¹ Assuming the maximum flange depth is limited by the total depth

² Determined based on empirical data for the type of steel (S500, S400) and concrete (C70/85, C20/25) [76], defined in Appendix I

$$C = \frac{C_0}{C_c L} = b_w h + (b - b_w) h_f + \left(\frac{C_s}{C_c} \right) A_s + \left(\frac{C_f}{C_c} \right) [b + 2h] \rightarrow \text{Minimum} \quad (46)$$

The formulation of the cost function, the values of the cost ratios C_s/C_c and C_f/C_c , as well as most of the following constraints are as proposed by Fedghouche (2018) [76].

As in any optimization problem we are also going to introduce some design constraints which are defined in accordance with EC2, including the nonlinear ultimate behaviors of concrete and reinforcing steel, as well as current practices rules. Nonetheless, the suggested formulations can be altered to accommodate for different design codes with minimal modifications. To ensure that the external moment applied on the beam does not exceed the resisting moment provided by the cross-section, the following inequality is defined:

$$M_{Ed} \leq \eta f_{cd} (b - b_w) h_f (d - 0,5 h_f) + \eta \lambda f_{cd} b_w d^2 \alpha (1 - 0,5 \lambda \alpha) \quad (47)$$

To ensure equilibrium between the internal forces and thereby that the structure remains stable, we will use the formulation in Eq. (48). This ensures that the structure can safely withstand the applied loads without experiencing excessive deformations or failure.

$$\alpha = \left(\frac{f_{yd}}{f_{cd}} \right) \left(\frac{A_s}{\eta \lambda b_w d} \right) - \frac{(b - b_w) h_f}{\lambda b_w d} \quad (48)$$

Furthermore, it is important to ensure that the steel reinforcement falls into the minimum and maximum steel reinforcement percentage, to prevent brittle failure, and control cracking. The formulations for this are:

$$\frac{A_s}{b_w d} \geq \rho_{min} \quad (49)$$

$$\frac{A_s}{b_w h + (b - b_w) h_f} \leq \rho_{max} \quad (50)$$

In Eq. (47), (48) and (49), it is assumed that the neutral axis is positioned under the beam flange which ensures that the section behaves as a T-beam shown in Figure 21. To ensure that the strain compatibility between steel and concrete is maintained, to prevent excessive deformations and ensure that the structure behaves as a composite material under applied loads, we will use the following formulation:

$$\varepsilon_{cu3} \left(\left(\frac{1}{\alpha} \right) - 1 \right) \geq \frac{f_{yd}}{E_s} \quad (51)$$

By fulfilling constraint (52), the design can avoid the need for compression reinforcement, which can lead to additional complexity and costs. Moreover, compression reinforcement is typically not required in the design of T-beam sections [76].

$$\lambda \alpha (1 - 0,5 \lambda \alpha) \leq \mu_{limit} \quad (52)$$

μ_{limit} is the limit value of the reduced moment and is provided as a threshold to determine if the T-beam design can avoid the need for compression reinforcement. The value varies depending on the combination of steel reinforcement grade and concrete strength class.

To ensure that the external shear force is smaller than the resisting shear force, we define the following inequality:

$$V_{Ed} \leq V_{Rd,max} = v_1 \frac{f_{cd} b_w z}{\tan(\theta) + \cotg(\theta)} \quad (53)$$

where,

$$v_1 = 0,6 \left(1 - \frac{f_{ck}}{250} \right) \quad (54)$$

$$z = 0,9d \quad (55)$$

and θ is the angle between concrete compression and the main chord. To ensure that the deflection constraint is satisfied, the following formulation is used:

$$\frac{5 q L^4}{384 E_{cm} I_c} \leq \delta_{lim} \quad (56)$$

where according to Sørensens' book [79]:

$$\rho = \frac{A_s}{b_w d} \quad (57)$$

$$I_c = \xi \frac{b_w (\alpha d)^3}{3} + \eta \rho (1 - \alpha)^2 b_w d^3 \quad (58)$$

Lastly, to ensure that the optimal solution adheres to some physical limitations, the following geometric design variable constraints including rules of practice are introduced:

$$h \geq \frac{L}{16} \quad (59)$$

$$\frac{d}{h} = 0,9 \quad (60)$$

$$0,2 \leq \frac{b_w}{d} \leq 0,5 \quad (61)$$

$$\frac{b - b_w}{2} \leq \frac{L}{10} \quad (62)$$

$$\frac{b}{h_f} \leq 8 \quad (63)$$

$$h_f \geq h_{f,min} \quad (64)$$

$$\frac{b}{b_w} \geq 3 \quad (65)$$

$h_{f,min}$ was set to 150mm.

Mathematically, the minimum cost design optimization problem can now be formulated as “find the design variables $b, b_w, b, f, d, h_f, A_s$ and α in Table 2, that minimize the cost function defined in Eq. (46) subjected to the design constraints given in Eq. (47) through Eq. (65)”.

4.3.1. Dataset generation for Task 3

The optimization code that was written for this task (Appendix I) generates a dataset of optimized RC T-beams used to train a ML model. The dataset was saved in a CSV file with the following columns: f_{ck} , f_{yk} , q , g , L , VEd , MEd , Δ_{lim} , b , b_w , h , d , h_f , A_s , α , $cost$, and $success$. The algorithm used for optimization is the *trust-constr* method, which is a trust-region algorithm for constrained optimization (Chapter 2.1.1). It is the most versatile constrained minimization algorithm implemented in SciPy and is suitable for large-scale problems [80]. The *trust-constr* method stands out as particularly appropriate for our task due to its flexibility and robustness in handling both equality and inequality constraints, which are present in our case. Furthermore, this method is proficient at handling nonlinear optimization challenges [81], which are predominant in most of the constraints we are dealing with in this task. A defining feature of the method is its ability to adjust the trust-region according to the progress of the optimization. This can lead to more efficient exploration of the solution space, potentially reducing the time required for dataset generation. The maximum iterations for each section were set to 5000. The x -tolerance, which is the tolerance for the change in the design variables between consecutive iterations, and the g -tolerance, which is the tolerance for the gradient of the objective function, were both set to $1e-5$. When the tolerances are not changing significantly, further iterations are unlikely to produce substantial improvements in the solution and the algorithm will terminate the search.

The dataset was generated for various combinations of concrete strengths (20, 70 MPa), steel reinforcement yield strengths (S400, S500), load values (5-30 kN) and span lengths (5-20 m). For each combination, the script optimizes the design variables as detailed above, and subsequently stores them in the CSV file along with the corresponding cost and the success status of the optimization process.

5. Task 1: Predicting anticipated failure mode

In this section, we aim to investigate the effectiveness of three distinct ML algorithms for predicting the anticipated failure mode of an RC beam using previously unseen data on its properties. The task is framed as a classification problem, as it involves predicting a categorical variable, i.e., the anticipated failure mode. The goal of the ML model is to map the properties of the beam (input features) to the anticipated failure mode (target variable). The target variable is divided into one of three classes, corresponding to one of three failure modes (bending, shear and deflection), making this a multiclass classification task.

This problem is well-suited for supervised learning as the dataset is labeled, consisting of the properties of an RC beam and the corresponding anticipated failure mode.

5.1. Data preprocessing

When using classifiers for a multiclass classification task, it is important to preprocess and prepare the dataset appropriately. First let us look at the number of instances of each class in the dataset in Figure 22.

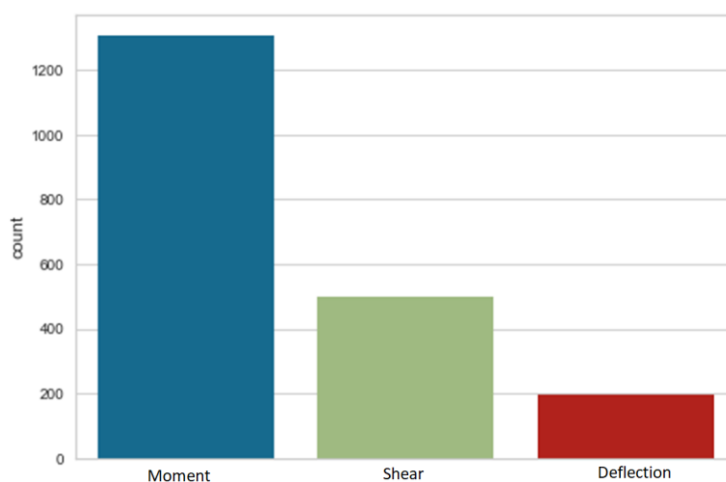


Figure 22: Class distribution in the data set

The dataset is quite imbalanced with the moment class dominating the beam's reason of failure. This suggests that in the process of iterating through different beam geometries and reinforcement configurations, the moment capacity was reached and exceeded first more frequently than the other capacities. The imbalance is likely due to the nature of the design space we have explored with the increase of the beam's properties. As we know, the different failure

modes are affected by different factors, and the proportions of the beam sections we have generated are making certain types of failures more probable.

In general, a simply supported beam under uniformly distributed load would be more susceptible to bending moments. The deflection capacity is important, but often less critical than the moment and shear capacities in RC design. In addition, the $L/250$ limit is relatively conservative, making the beam less likely to fail due to deflection. The large moment frequency reflects that moment failure is the most common concern in the design and analysis of these beams.

By plotting the scatterplots of the features and the classes seen in the figure below, we can get an important insight into the relationship between the features and the target variable.

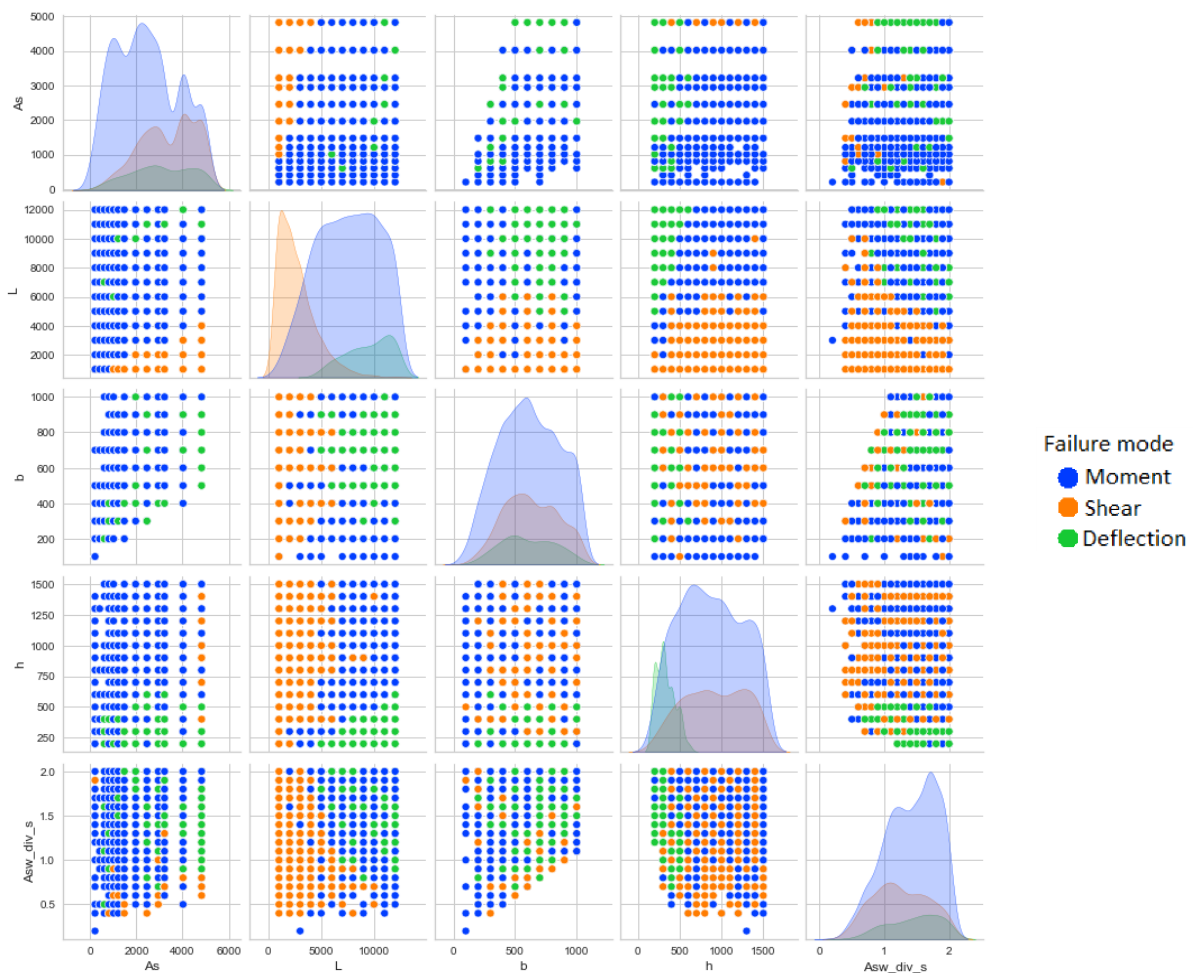


Figure 23: Scatterplot of feature pairs

A careful examination of the second row reveals that the moment class is notably dominant when L and h values are elevated, while A_s exhibits a lower value. When h is comparatively smaller, as seen in the $L-h$ and $h-A_s$ diagrams, deflection class unsurprisingly takes over as the prevailing

failure mode. As displayed in the $L-h$ and $L-b$ diagrams, if the beam's length is relatively modest, shear capacity becomes the primary failure mode. Lastly, a glance at the $L-Asw_{div_s}$ graph illustrates an increase in shear failure modes as Asw_{div_s} decreases. The feature interrelationships highlighted are more discriminative in the scatterplot, leading to enhanced cluster separation for different classes. This is likely due to these feature combinations being intricately tied to the beam's behavior and performance, hence wielding a significant influence over the beam's overall classification.

The scatterplots that exhibit a mixing of classes signify that the interactions between the features are not as directly influential or strong in determining the failure classification. For example, Asw_{div_s} is a significant parameter influencing the beam's shear capacity. However, its correlation with most other features is not as robust or transparent, implying it may not be adequate to explain the failure mechanism independently. Moreover, the relationship between A_s and b did not result in any instances of shear failure, but mixed moment and deflection class instances. This suggests that the association between the longitudinal reinforcement and the beam's width has more bearing on moment and deflection capacities rather than the shear capacity.

Referring to Figure 23, it is noticeable that in feature relationships like $L-A_s$, we can draw a clear line, segregating the moment class from the shear class. This scenario introduces the concept of a linear decision boundary. However, in most relations, the same technique would not separate the classes, hinting at a nonlinear relationship between features and classes. Simply put, the data does not allow for linear separation within the given feature space, as depicted in the figure below.

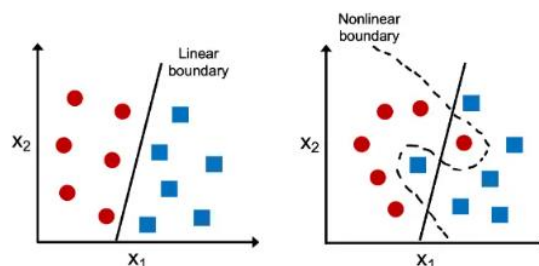


Figure 24: Linear and Nonlinear boundaries [82]

The algorithms chosen for this task, as explained in Chapter 2.2.2 and 2.2.4, are adept at managing the dataset's inherent nonlinearity.

The diagonal plots display the histograms of individual features, demonstrating each feature's distribution across the dataset. As seen in Figure 23, the shear and deflection classes largely coincide with the moment class. This overlapping indicates that data points for shear and deflection classes are not readily distinguishable from the moment class based on the available features. This situation could lead to increased model complexity to accommodate for the difficult separation of classes, and more ambiguous decision boundaries. To mitigate this challenge, we will adjust the scoring method to accommodate the imbalance, use performance metrics such as the F1-score, implement deep learning with a neural network, and finally apply undersampling techniques to counter the imbalance and overlap within the dataset.

Notably, we expect MLP to deliver superior performance given its track record of success in diverse classification tasks, especially when substantial amounts of training data are available [1]. The robustness of MLPs in managing complex nonlinear relationships, as explained in Chapter 2.2.4, is anticipated to effectively tackle the issues of class overlap and class distinction present in the dataset. Conversely, the kNN algorithm, detailed in Chapter 2.2.2.2, may be least suited for this task due to its sensitivity to noisy data, outliers, and high computational burden when dealing with large datasets. Furthermore, it could potentially falter when handling the high dimensionality of the data and the overlap of classes.

Lastly, we will prepare our features differently based on the requirements of each specific ML algorithm. For the kNN algorithm we will normalize the features. Normalization scales the features to fall within a range of [0,1]. This also ensures that all features contribute equally to the distances computed by the kNN algorithm, preventing features with larger magnitudes from dominating, as detailed in Chapter 2.2.2.2.

On the other hand, for the DT algorithm, we will not apply any scaling to the features. This is because DTs are not sensitive to the scale of input features. They make splits based on conditions, not distances, and thus are unaffected by the scale of the features. This property allows us to interpret the feature importance more intuitively when using DTs.

When it comes to the MLP, we will standardize the features. Standardization transforms the features to have a mean of 0 and a standard deviation of 1. This ensures that all features contribute on a similar scale, which is important for the gradient descent algorithm utilized by MLP for learning the optimal weights. Features with larger scales can result in longer training times and potentially prevent the model from learning effectively. Furthermore, standardization might be preferred in many cases, especially when using activation functions that assume input

values centered around 0, such as sigmoid or tanh. These activation functions are optimized for inputs in this range, which can substantially enhance the learning process and the model's overall performance.

5.2. Hyperparameter tuning

For the grid and random search, we selected *f1_macro* as the scoring criterion, which specifies the metric that will be used to evaluate the performance of the model for each combination of parameters. This enables us to care for the performance on all three classes equally. The reason for not choosing *accuracy* as the scoring criterion is due to the overweight of the moment class in our dataset. The optimal parameters chosen in that case would favorize the majority class as predicting it correctly would still give a high accuracy score.

5.2.1. DT

Our grid search identified the optimal hyperparameters for the DT classifier as follows:

- Criterion: *entropy*
- Max Depth: 6
- Max Features: *sqrt*
- Min Samples Leaf: 3
- Min Samples Split: 1

Given our specific task of classifying the anticipated failure mode in RC structures with a dataset consisting of 5 features, 1 target variable, and a sample size in the thousands, the chosen hyperparameters for the DT make a lot of sense.

The selection of *entropy* as the splitting criterion is well-aligned with our multiclass problem. As an effective measure of impurity in the input set, *entropy* can effectively distinguish between our three classes. This selection also corresponds with observations made during our data preprocessing stage (Chapter 5.1), which highlighted a nonlinear relationship between the features and classes. The *entropy* criterion is capable of handling such nonlinearity. Opting for *entropy* over the *gini* may also serve to confirm that the increased complexity of *entropy* makes it better suited for handling imbalanced datasets, as discussed in Chapter 2.2.2.1.

With a relatively small number of features, *sqrt* and *log2* would result in considering roughly similar numbers of features at each split, so the difference might not be substantial. In any case, defining max features will help mitigate overfitting, a major disadvantage of DTs.

Minimum samples per leaf are chosen as 3. This restriction prevents the model from making decisions based on individual instances, thereby avoiding overfitting and aiding in the generation of smoother predictions. However, setting minimum samples required to split an internal node to 1 may potentially lead to overfitting as each sample could become a split point. This is a point of caution as it could lead to high variance, a drawback of DTs. The complexity between the features, as discussed in the preprocessing stage, may have forced the value down. Yet, given the large sample size, the impact can most likely be mitigated.

The model's max depth is set to 6. Given our 5 features, this depth is adequate for capturing interactions without leading to an overly complex or overfit model. The decision is in line with the considerations discussed for avoiding overfitting (Chapter 2.2.2.1).

To further strengthen our assurance of avoiding overfitting, we compared the training and test accuracy at different depths. In this case, we can observe an increase in accuracy on the training dataset up to a point with a tree depth of around 9 to 10 levels where the tree perfectly fits the dataset. The comparison of the training and test scores in Figure 25 reveals that, up to a tree depth of 6, the accuracy on the test set improves with each depth increase, after which it starts to worsen. This is something we would expect to see in an overfitting pattern and aligns with the expectations discussed in the *max_depth* parameter of Chapter 2.2.2.1. We would therefore choose a depth of 6 where the best performance, with an accuracy of 0.89, was achieved.

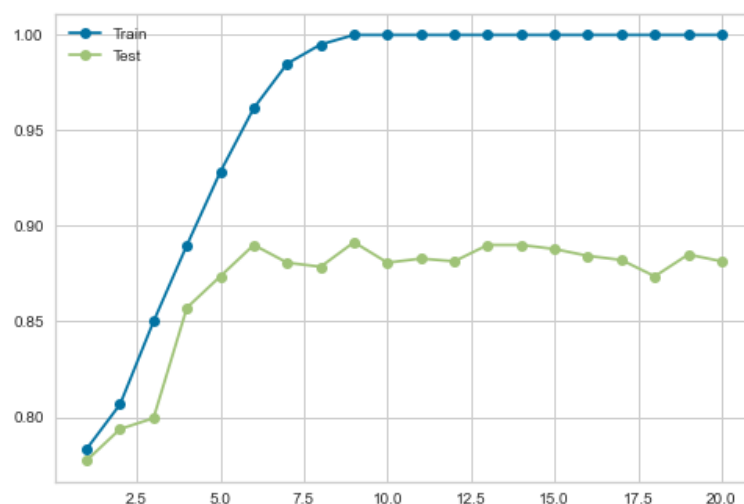


Figure 25: Plot of the learning curves over max depth

5.2.2. kNN

The grid search algorithm identified the following as the optimal hyperparameters for the kNN classifier:

- Algorithm: *auto*
- Number of Neighbors (k): 15
- Power parameter for the Minkowski metric (p): 1
- Weights: *distance*

Considering the findings from the grid search, the chosen hyperparameters align well with the dataset at hand, given the specific characteristics of the kNN algorithm. The *auto* algorithm enables the model to adapt to the training data's structure and select the most effective method for finding nearest neighbors. As revealed upon accessing the chosen algorithm (as explained in 2.2.2.2), the *kd_tree* was selected for our specific dataset. The *kd_tree* algorithm, as previously explained, is generally more efficient than the *brute* method when dealing with larger datasets. Conversely, the *ball_tree* method is known to not perform as well with low-dimensional data like ours. Therefore, it is reasonable that *kd_tree* was chosen as the preferred method for this specific case.

The Minkowski distance metric with a power parameter of 1, equivalent to Manhattan distance, proves beneficial in handling varied feature scales in our data. As the model aims to predict failure modes in RC beams, such versatility helps accommodate the wide range of cross section specifications present in the dataset.

The weight parameter has been set to *distance*, implying that nearer neighbors contribute more heavily to the prediction of a new instance. Given our task's multiclass nature and apparent class imbalance, this strategy amplifies the influence of minority classes in the vicinity of the new instance, potentially boosting our model's performance on these underrepresented failure modes.

However, the number of nearest neighbors set to 15 raises some questions given its relatively high value. A larger k can smooth out the decision boundary, reducing the risk of overfitting. Nevertheless, it may also oversimplify the model and reduce the precision of predictions, especially in complex or noisy datasets. Since our data does exhibit some degree of noise and class overlap, as evidenced by the scatterplot analysis, the choice of k warrants further investigation.

In order to make a more informed decision on the optimal k -value, we will use the elbow method, which involves plotting the misclassification error against different values of k and visually identifying the “elbow”. This point corresponds to the optimal k -value where the misclassification error is minimized.

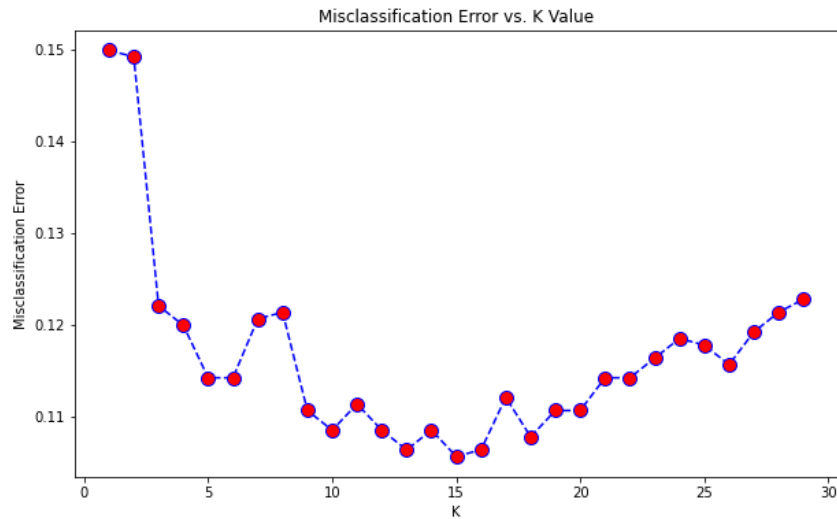


Figure 26: Plot of misclassification error over k

As seen in the figure above, the best parameter for k coincides with the one found by *GridsearchCV* and is 15, with the lowest misclassification error of 0.106. As the k increases or decreases, the misclassification error has a slight increase. Therefore, the optimal value in this case for k is indeed 15. As explained in Chapter 2.2.2.2 we also do avoid ties as the selected k is an odd number.

5.2.3. MLP

For the MLP classifier, due to the extensive number of parameters and vast search space, a random search approach was employed to identify the optimal hyperparameters. The results are as follows:

- First layer (Input): 200 neurons with *tanh* activation function
- Second layer: 100 neurons with *tanh* activation function
- Third layer: (Output): 3 neurons with *softmax* activation function

In addition, the following settings were applied:

- Optimizer: *lbfgs*
- Learning rate: 0.001.

- Loss function: *log_loss*
- Training Configuration: The model was trained for 300 epochs with a batch size of 256.
- Alpha = 0.1

Some things to note: For multi-class classification tasks, the MLP classifier automatically uses the *softmax* function as the output activation. It also uses *log-loss* (or cross-entropy loss) as the loss function when the output activation is *softmax*. Lastly, the MLPClassifier implements L2 regularization, which is controlled by the hyperparameter *alpha*. This introduces a penalty on the size of the weights in the network, encouraging the model to keep them small. This can help prevent overfitting by adding a cost to the loss function for complexity, making the model simpler and more robust to noise in the training data. These behaviors are hard-coded and cannot be directly changed, but they follow best practices for the given task.

The chosen hyperparameters for the MLP are consistent with the requirements of our task and dataset. Its architecture, with three layers (first layer with 200 neurons, second layer with 100 neurons, and third layer with 3 neurons), is a considerable number given we have only 5 input features, enabling the network to effectively learn and capture any complex, nonlinear relationships. The *tanh* activation function in the input and hidden layers enables the MLP to model nonlinearities while ensuring a zero-centered output. The output layer uses the *softmax* function, a suitable choice given our three-class classification problem, as it can provide class probabilities. The optimizer, *lbfgs*, is a good choice for our small dataset, allowing for a quick convergence in a single batch due to its memory limitations.

The reason for referring to the dataset as small in this context, as opposed to when using the kNN algorithm above, is because the size consideration of the dataset depends on the specific algorithm in question. When we talk about a "small" dataset in the context of the MLP model and the *lbfgs* optimizer, it is relative to the typical dataset sizes that deep learning models are often trained on, which can range from thousands to millions of examples.

The learning rate of 0.001 was set, ensuring a balance between stable training and convergence speed. The *log_loss* function is appropriate for our multiclass classification task. The MLP's training configuration of 300 epochs and a batch size of 256 ensures multiple passes over the dataset, allowing for a more robust model. Lastly, an *alpha* of 0.1 is chosen to prevent overfitting by adding a penalty to the loss function.

However, further tests on the number of epochs and learning rates might be beneficial. It is possible that different epoch numbers might yield better results due to the potential overfitting or underfitting of our current setup. Similarly, the learning rate affects how quickly or how well our model can find a solution. By conducting further tests on learning rates, we may find a value that allows our model to learn and generalize better from our data.

We see by studying the loss curves in Figure 28 that we do reach convergence, without actually overfitting, which typically will look like displayed in Figure 27. What this means is that we try to make sure that we do not reduce the training error at the expense of the test error.

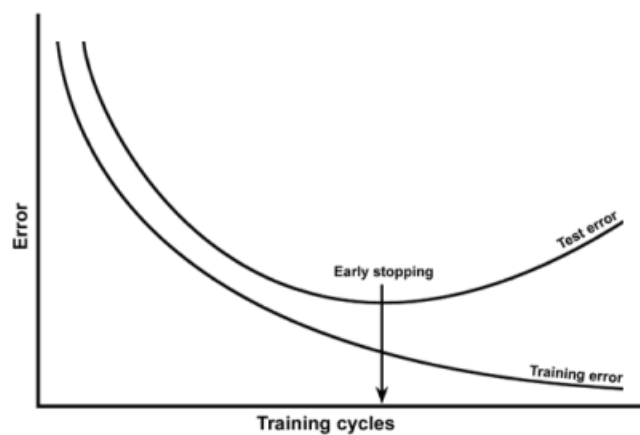


Figure 27: Signs of overfitting [83]

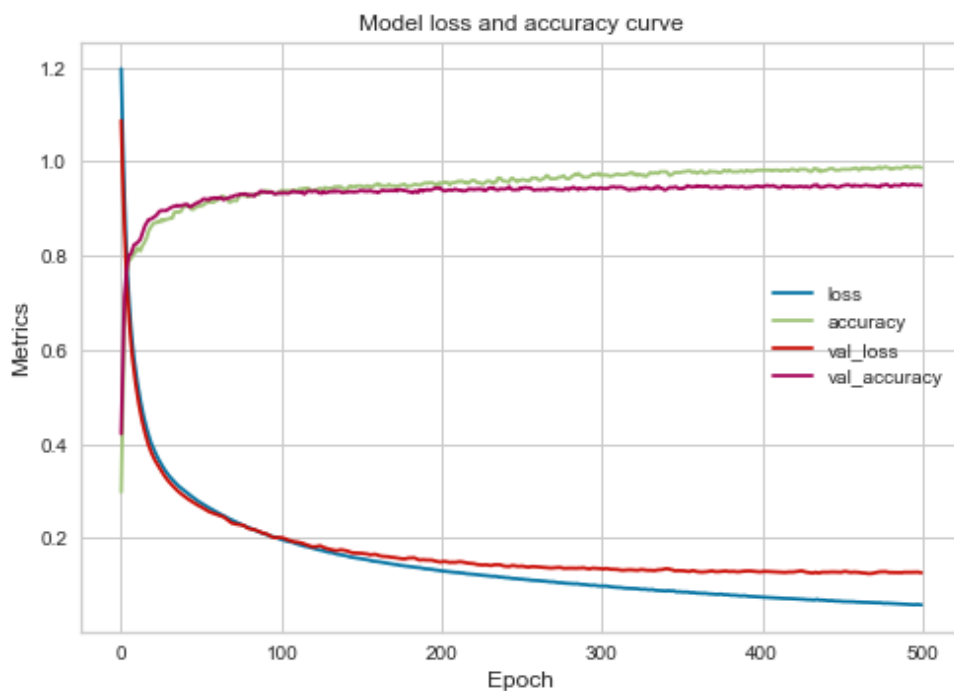


Figure 28: Training and Validation Metrics over Epochs

In our case, we observe quite a long plateau on both the validation accuracy and validation loss, and can therefore stop early, since we effectively gain no more improvements after approximately 300 epochs. We do also see that the loss continues to decrease as the validation loss stays the same, which leads to an increase in the gap between the losses and a potentially higher risk of overfitting after 300 epochs. The plots suggest that the number of hidden layers and neurons selected are effective, as the graphs exhibit a satisfactory rate of increase and decrease, plateau appropriately after a sufficient number of epochs, and display minimal oscillation.

The learning rate parameter is one of the keys to building an optimal NN [84], and can be studied in detail by plotting the training or test score for different learning rates as done in the figure below.

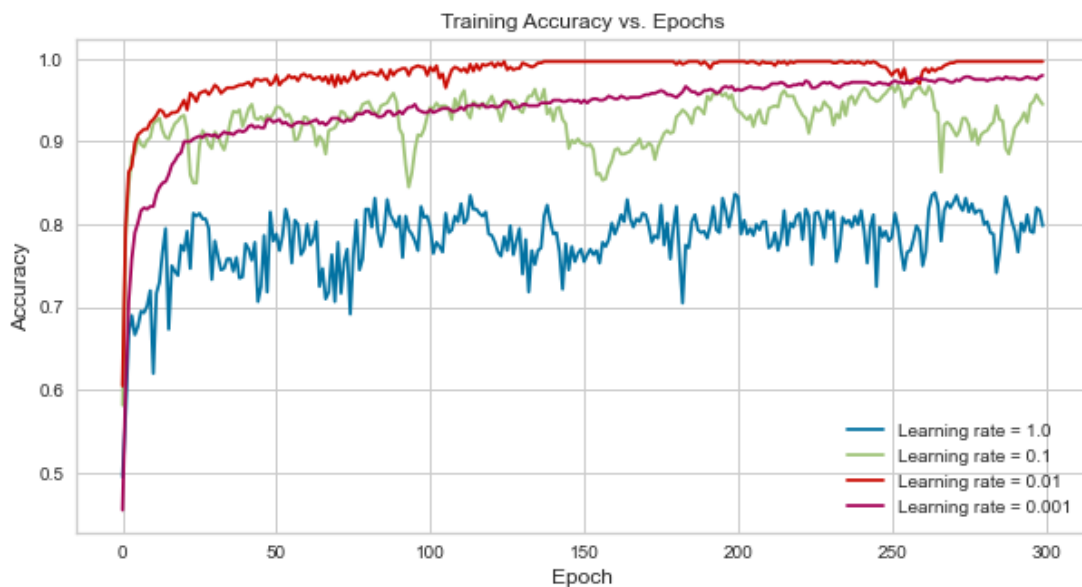


Figure 29: Impact of Learning Rates on Accuracy over Epochs

During training, it is evident that a learning rate of 1.0 is not optimal. If we lower the learning rate to 0.01, the score increases from 0.6 to 0.94, but the problem with this learning rate is that as the epochs increase it becomes almost constant. The choice falls between the learning rate of 0.1 and 0.001, with the latter covering more data points, thus taking more time to achieve the results compared to the first one [84]. After trying both, we found that the learning rate of 0.1 gives the best performance and corresponds with the rate found by the random search. We found similar patterns on the validation set.

5.3. Results

This chapter presents the outcomes of Task 1, which was centered around predicting the anticipated failure mode in the dataset. The goal of this task was to employ various ML techniques to accurately categorize each section according to its corresponding anticipated failure mode. The results provide an in-depth examination of the predictive power and accuracy of these models for this task.

5.3.1. Classification reports

In this section, we present the classification reports for each of our models. These reports focus on essential evaluation metrics, namely precision, recall, and F1-score, which are explained in detail in Chapter 2.4.1. Additionally, we analyze the macro-averaged and weighted-average scores, also discussed in Chapter 2.4.1. Support indicates the number of instances available for each class in the test set, offering insights into the sample size and significance of the results. The color spectrum provides a quick visual indication of model performance, with dark red signaling high performance and pale yellow indicating low performance.

Table 3: *KNeighborsClassifier* Classification Report

	Precision	Recall	F1-score	Support
Moment	0.89	0.96	0.92	926
Deflection	0.80	0.65	0.72	130
Shear	0.95	0.81	0.87	345
Accuracy			0.89	1401
Macro avg	0.88	0.81	0.84	1401
Weighted avg	0.89	0.89	0.89	1401




Table 4: DecisionTreeClassifier Classification Report

	Precision	Recall	F1-score	Support
Moment	0.88	0.90	0.89	926
Deflection	0.76	0.72	0.74	130
Shear	0.81	0.78	0.79	345
Accuracy			0.85	1401
Macro avg	0.82	0.80	0.81	1401
Weighted avg	0.85	0.85	0.85	1401





Table 5: MLPClassifier Classification Report

	Precision	Recall	F1-score	Support
Moment	0.95	0.98	0.96	926
Deflection	0.92	0.88	0.90	130
Shear	0.97	0.91	0.94	345
Accuracy			0.95	1401
Macro avg	0.95	0.92	0.94	1401
Weighted avg	0.95	0.95	0.95	1401



5.3.2. Confusion matrices

In line with the explanations provided in Chapter 2.4.1, we present the confusion matrices for each ML model. These matrices provide a detailed breakdown of the model's classification performance, enabling a closer examination of the predicted and actual class labels. The same color spectrum is used here as in the classification reports, although dark red now represents 1000 instances, while pale yellow represents 1 instance.

Table 6: *KNeighborsClassifier* Confusion Matrix

True class	Moment	889	21	16
	Deflection	45	85	0
	Shear	66	0	279
		Moment	Deflection	Shear
		Predicted classes		

Table 7: *DecisionTreeClassifier* Confusion Matrix

True class	Moment	833	27	66
	Deflection	28	102	0
	Shear	43	0	302
		Moment	Deflection	Shear
		Predicted classes		

Table 8: *MLPClassifier* Confusion Matrix

True class	Moment	905	10	11
	Deflection	15	115	0
	Shear	30	0	315
		Moment	Deflection	Shear
		Predicted classes		

5.3.3. Decision boundaries

Visualizing decision boundaries helps us understand how classifiers distinguish between classes in a feature space, revealing their complexity and decision-making strategies. However, it becomes challenging when the number of features exceeds two, limiting visual representation. To address this, we reduced the dataset's dimensionality by fitting the classifier to a single pair of features at a time. We focus our analysis on two specific pairs: shear reinforcement over the length (upper row, Figure 30), and length over the height (lower row, Figure 30).

It is important to note that optimal parameters found in Chapter 5.2 for the full dataset, may not be suitable for the reduced dataset. Additionally, visualizing decision boundaries for feature pairs does not fully capture the model's complexity on the entire test set, as other features contribute to improved performance.

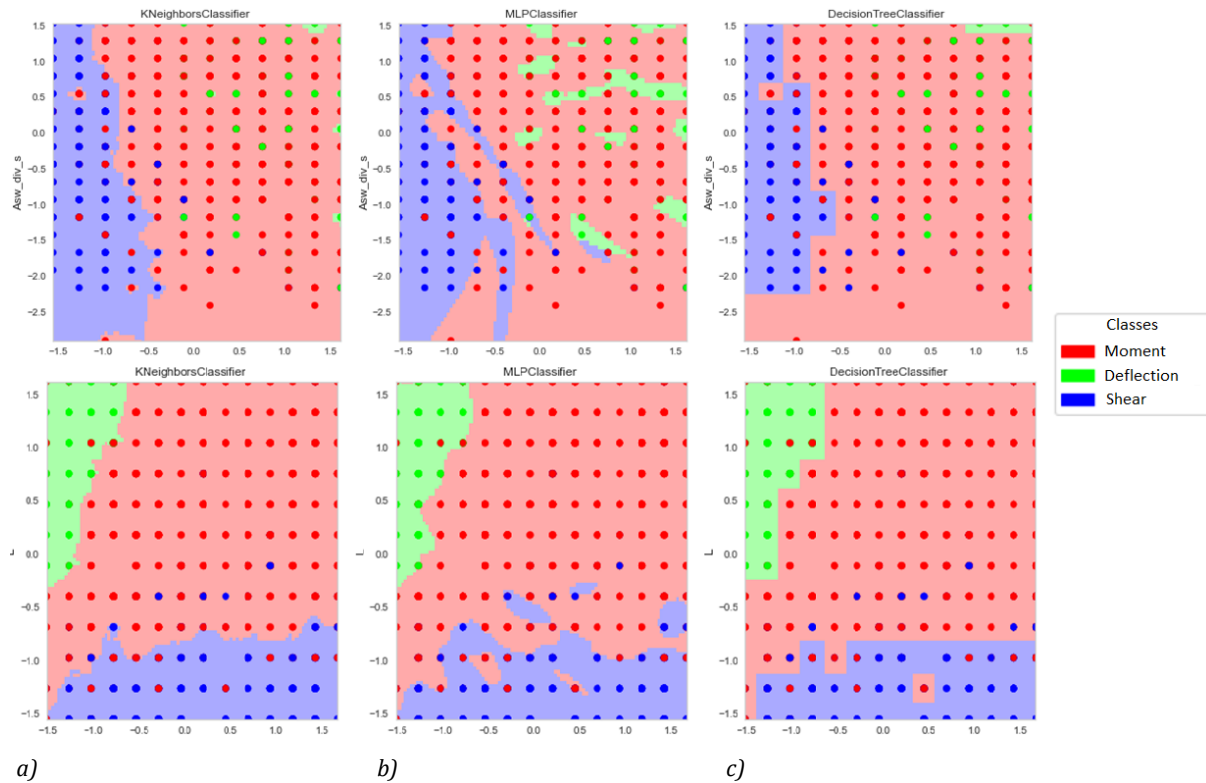


Figure 30: Decision boundaries for two pairs of features for a)kNN; b)MLP; c)DT

5.4. Results after balancing with under-sampling

For the implementation of the *RandomUnderSampler* (RUS) we used our own sampling strategies, which allowed us to tailor the under-sampling process to the specific characteristics of our dataset. By using a more targeted approach, we were able to preserve important instances in the majority class while still balancing the class distribution. This resulted in an increase in overall performance compared to using the default sampling strategies provided by scikit-learn. We experimented with different sampling strategies for each model through trial and error, refining the optimal hyperparameters after each alteration. By doing so, we were able to achieve better performance and generalize better to new and unseen data. The specific sampling strategies used on the training set for each model are listed below:

Table 9: Sampling strategies for RUS

Model	Sampling strategy (instances)		
	Moment	Deflection	Shear
kNN	250	67	153
DT	140	67	100
MLP	100	67	110

5.4.1. Confusion Matrices

In this section we present the updated confusion matrices, which showcase the results of applying RUS to the dataset. By examining these matrices, we can easily observe any changes in the models' ability to accurately predict class labels, and gain a clear understanding of the impact of RUS on their performance.

Table 10: KNeighborsClassifier Confusion Matrix after RUS

True class	Moment	849	30	47
	Deflection	25	104	1
	Shear	34	0	311
		Moment	Deflection	Shear
		Predicted classes		

Table 11: DecisionTreeClassifier Confusion Matrix after RUS

True class	Moment	833	27	66
	Deflection	28	102	0
	Shear	43	0	302
		Moment	Deflection	Shear
		Predicted classes		

Table 12: MLPClassifier Confusion Matrix after RUS

True class	Moment	825	39	62
	Deflection	4	126	0
	Shear	14	0	331
		Moment	Deflection	Shear
		Predicted classes		

5.5. Discussions

In Task 1, we first examined the performance of *KNeighborsClassifier*, *DecisionTreeClassifier*, and *MLPClassifier* on an imbalanced dataset for predicting the anticipated failure mode in RC beams. Given the risk of model bias towards the moment class as the majority class with 380 training instances, compared to deflection class with 67 instances and shear class with 153 instances, we used strategies to mitigate this imbalance. Utilizing the macro average f1-score as the metric for hyperparameter tuning proved effective in enhancing model performance across all classes, as showcased by the promising results in Chapter 5.3. This, coupled with metrics such as individual class f1-scores and confusion matrices, provided a well-rounded assessment of the classifiers' performance.

The classification reports reveal that among the three models evaluated, the MLP classifier outperforms the rest with the highest macro average f1-score of 0.94. This confirms our assumptions about the robustness of MLPs expressed in the preprocessing stage and is consistent with the theoretical advantages of MLPs. Their ability to model complex nonlinear relationships through their nonlinear activation functions, makes them suitable for complex datasets, reflecting their superior performance in balancing precision and recall across all classes, even in the presence of an imbalanced dataset.

The kNN classifier follows with a score of 0.84, and the DT classifier trails with a score of 0.81. This finding is somewhat surprising since we expected the kNN classifier to perform worse overall due to its sensitivity to inconsistencies and class overlap.

Despite the kNN classifier performing better overall than the DT classifier, the DT shows a higher f1-score for the deflection class, suggesting a more balanced precision-recall performance for this

specific class. A closer examination of the confusion matrices of the DT and kNN classifiers does indeed uncover a larger number of deflection class misclassifications in the kNN model as opposed to the DT (45 vs. 28). This highlights the need to consider multiple evaluation metrics when assessing classifiers, as each offers a unique perspective on performance. The kNN classifier, despite a higher misclassification rate for the deflection class, outperforms the DT classifier on the shear and moment classes, making it overall perform better in terms of classifying across all classes, and is more effective in predicting the anticipated failure mode in the dataset.

However, it is important to note that the performance of the two algorithms may vary depending on the dataset. Because their performance is quite similar, it is difficult to conclusively determine which of them will perform better on new unseen data. We can, however, draw some conclusions on the algorithm's performance by looking at their decision boundaries.

The MLP classifier stands out in its ability to decipher complex, nonlinear decision boundaries, as reflected in the smooth boundaries we observe in Figure 30b. This is a testament to its ability to leverage the *tanh* activation function that introduces nonlinearity into the learning process. Crucially, its capacity to discern class instances intermingled within clusters of another class showcases its superior skill in unravelling intricate feature relationships, outperforming both the kNN and DT classifiers.

Contrastingly, the kNN classifier's decision boundaries (Figure 30a), while smooth, closely follow the data's local structure. This highlights the algorithm's reliance on proximate training instances for new sample classification, as mentioned in the disadvantages of the kNN classifier. This becomes a challenge when handling deflection class instances that are scattered among other classes, especially in moment, as they do have some structural similarities. In response, the algorithm classifies them as moment, thereby boosting its performance for that class. However, this compensation leads to more misclassifications of the scattered minority class.

On the other hand, the DT classifier takes a different approach by partitioning the feature space into regions to build a global model. Each decision (i.e., each node) in the tree corresponds to a binary decision based on a single feature, resulting in an "axis-aligned" decision boundary that aligns with the axes of the feature space.

However, this axis-aligned nature of decision boundaries in the DT model may restrict its ability to identify complex nonlinear relationships compared to the kNN and MLP [12]. In cases where

the relationship between features and labels is not well-represented by a series of "if-then" decisions along the feature axes, a DT may struggle to accurately model that relationship. While this partitioning technique can handle scattered instances within the deflection class, it falls short in decoding the complex, nonlinear relationships of this set. This explains why kNN outperforms DT in classifying shear and moment classes, being able to distinguish these two classes more effectively.

Drawing from the data processing findings in Chapter 5.1, it is noteworthy to revisit the interplay between different features and their respective bearing on failure modes. The scatterplot observations suggest that width and shear reinforcement combinations lead to class mixing, indicating a weak or indirect correlation between these two features and the failure modes when paired with most other features.

When looking at the significance of these features in terms of structural mechanics, we notice that height has a direct proportionality to the shear capacity of a beam. Conversely, shear reinforcement and width each contribute to a separate aspect of the shear capacity – the shear reinforcement capacity and the concrete shear capacity, respectively. This further complicates the interpretation of the role of shear reinforcement and width in determining the shear capacity, as their relative contributions can vary depending on the specific conditions and characteristics of the beam.

The moment capacity, on the other hand, directly correlates with width and the square of effective depth. As a result, the effective depth has an exponential impact on moment capacity, whereas width affects it linearly. The shear reinforcement does not influence the moment capacity at all.

In the case of deflection capacity, it is solely dependent on beam length, thereby excluding the role of width and shear reinforcement. When assessing the acting forces, as seen in Eq. (18) and (24), beam length emerges as the only contributing factor to both the shear and moment. For deflection, although both width and effective depth are involved, the latter usually has a more significant impact due to its cubic relationship in the second moment of area calculation.

Reflecting on this, we find that the role of shear reinforcement is quite limited, contributing solely to the shear reinforcement capacity. Conversely, width presents a more substantial role, but due to the complexity of the formulas in which it appears, its contribution may not correlate strongly with specific failure modes, resulting in class mixing in the scatterplots. Furthermore, it is evident that other factors such as beam length and height can often dominate the influence of width and

shear reinforcement, making it harder to discern clear correlations between these features and failure modes when other variables are at play.

The MLP classifier's proficiency in discerning complex relationships, as discussed in Chapter 2.2.4, complements the intricate feature correlations observed in the scatterplot analysis exceptionally well. Its ability to classify instances where the correlation between features and failure modes is weak or indirect, such as the combination of width and shear reinforcement, contributes to its superior performance.

In contrast, the kNN classifier, which relies heavily on local feature relationships for decision making, may struggle when such relationships are not straightforward, as is the case for width and shear reinforcement.

Lastly, the DT classifier's method of partitioning the feature space aligns with the significant impact of features like beam length and effective depth on different failure modes. Its performance, while trailing behind MLP and kNN overall, is indicative of its ability to leverage these clearer relationships. However, the scatterplots underline the existence of complex feature interactions which might be challenging for DT to capture accurately, hence the lower performance for shear and moment classes.

The RUS technique was employed to address the class imbalance issue of the dataset. After applying RUS, the confusion matrices, as seen in Chapter 5.4.1, for all three models showed notable improvements in classification performance, especially for the deflection and shear classes. The number of TP for these classes increased, demonstrating that the classifiers were better equipped to handle the previously underrepresented classes. While the RUS technique successfully improved the classification performance for deflection and shear classes, the moment class's performance remained relatively stable or slightly decreased across the models. However, this trade-off is considered acceptable as the primary goal was to enhance the model's performance for the underrepresented classes, which was achieved.

The results are as expected, since removing information from the majority class would make it harder for the models to predict. Another alternative was using oversampling of the minority class. In our situation however, the class boundaries of the deflection class were not clear, so applying oversampling might have been less effective compared to under-sampling the majority class as done here. The reason for this is that oversampling such as *SMOTE*, generates synthetic samples by interpolating between instances of the minority class, and when class boundaries are

not well-defined, this interpolation process may generate samples that are very close to or within the regions of other classes, resulting in further class mixing and potentially poorer classifier performance.

In the context of this problem, the practical comparison of the three algorithms reveals the potential of ML as a tool for anticipating capacity failure in an RC beam. DT provided a fast and easily interpretable model, with the distinct advantage of being able to visualize decision-making processes as demonstrated in Appendix E. This transparency not only aids in understanding the model's predictions, but also helps to build trust in the model's outcomes, which is crucial given the high stakes associated with structural engineering decisions. Visualization aids in understanding the interrelationships and relative importance of different attributes, thus enabling a more intuitive grasp of complex engineering scenarios. This can facilitate the communication to individuals without technical expertise. They also need less data cleaning, which can make the overall process of anticipating failure modes more time efficient.

However, an observed instability in the model performance scores when running it multiple times, even with the same fixed random seed, raises concerns regarding its consistent application. This could be due to one of the key disadvantages of the DT in which small variations can produce drastically different trees. This inconsistency can be problematic in structural engineering, as we do rely heavily on accuracy and predictability to ensure safety and functionality. In addition, the DT model's instability might make it more challenging to explain and justify decisions to stakeholders.

Therefore, while the interpretability and visual decision-making process of DT are advantages in the structural engineering context, its inconsistency in performance could hinder its wider acceptance and application. Careful considerations and supplementary stability-enhancing methods like the ones mentioned in the drawbacks of the DT model in Chapter 2.2.2.1, will be required to address this issue.

The kNN classifier demonstrated considerable potential. Its relatively high score in our multiclass problem, coupled with the intuitive simplicity of its implementation and minimum need for hyperparameter tuning, makes it a strong contender for certain applications within this field. It could, for example, provide a quick and efficient method for preliminary estimations, or serve as an accessible entry point for professionals looking to integrate ML methods into their work.

However, despite its simplicity and high performance, the kNN classifier has a number of limitations that may reduce its suitability for some structural engineering applications. The local nature of the classifier's decision-making process can make it less effective in scenarios where relationships between features and outcomes are complex, nonlinear, or otherwise not easily captured by proximity alone. This was evident in the way the kNN classifier handled instances from the deflection class that were scattered among instances from other classes, especially moment. Additionally, the varying scales and units of features necessitate careful normalization, adding complexity to kNN's implementation. Performance can also degrade with high-dimensional data, common in this field, due to the "curse of dimensionality" and data sparsity. Additionally, kNN's reliance on a training instance dataset might lead to high computational costs, especially with large datasets typical in structural engineering, potentially making it impractical in these situations.

While the MLP classifier necessitated greater and more complex hyperparameter tuning, its robust performance substantiated its reputation as a versatile tool adept at handling complex data patterns. This robustness makes it suitable for structural engineering, even when dealing with complex datasets and intricate feature relationships, as it can effectively balance precision and recall across all classes. However, its "black-box" nature could pose interpretability and explanatory challenges, particularly for non-experts.

6. Task 2: Predicting capacities

In this section, our objective is to evaluate the performance of various ML algorithms in predicting the moment capacity, shear capacity, and load capacity for a simply supported RC beam using unseen data on its properties. This task is framed as a regression problem, as it involves predicting continuous variables, i.e., the capacity values. The goal of the ML model is to map the properties of the beam (input features) to the respective capacities (target variables).

This problem is suited for supervised learning, given that the dataset is labeled, comprising the properties of an RC beam and the corresponding capacities. The evaluation metrics being used are explained in Chapter 2.4.2.

6.1. Data preprocessing

The quality of datasets directly influences the performance of the algorithms, making their assessment a critical aspect of this thesis. If the datasets are found to be lacking, appropriate measures must be taken to resolve this.

For the first and second subtasks of Task 2, where the moment and shear capacity serve as the dependent variables, the datasets seem to be well-suited. Despite the broad range of values in the dataset, the data exhibits logical relationships that align with our understanding of structural mechanics, which is crucial for the validity of our analysis. For instance, we observe that cross sections with larger heights and greater amounts of longitudinal reinforcement tend to have higher moment capacity. This relationship is consistent with the principles of structural engineering, where larger cross-sectional areas and increased reinforcement contribute to higher load-bearing capacity. Moreover, the datasets do not contain any apparent anomalies that would suggest data entry errors or other issues.

In the third subtask, where the load capacity is predicted, the dataset presents a challenge due to the presence of instances where the self-weight of the beam is enough to exceed its load capacity. In these cases, the external load is effectively zero. This can pose a challenge for the MLR algorithm, as it assumes a linear relationship between the independent and dependent variables. When the live load is zero, this relationship may not hold, leading to inaccurate predictions.

On the other hand, SVR is more flexible and can handle nonlinear relationships better. SVR uses a technique called the kernel trick, explained in Chapter 2.2.3.2, that allows it to fit a linear

regression into a transformed space, making it more robust and capable at modeling complex, nonlinear relationships.

Zero values in the dataset can also pose other challenges for both models. These issues arise due to the sparse nature of the data, skewed distribution, scaling issues, and potential impact on model interpretability. However, we expect SVR to handle these issues better than MLR due to its ability mentioned above and its robustness to outliers, as presented in 2.2.3.2.

Addressing this issue is not straightforward. One potential solution could be to remove entries with a load value of 0, but this approach could introduce bias to the dataset. Specifically, it would exclude cross sections that fail solely due to their self-weight, which are valid and important scenarios in the context of structural engineering. These instances represent real-world situations where a structure fails under its own weight without any additional load, and excluding them from the dataset would limit the model's ability to accurately predict these cases. An alternative approach to address this issue could be to transform the problem into a binary classification task, where 0 signifies that the beam is incapable of supporting a given load, and 1 indicates that it can. This could potentially be expanded to include multiple load levels.

Despite the potential challenges that this dataset weakness could pose, it has been decided to maintain the current state of the data. The primary reason for this decision is the realistic representation of the cross sections.

6.1.1. Feature correlation

To gain a deeper understanding of how each feature in the datasets influences the outcome, we have developed plots that illustrate the relationship between each independent and dependent variable across the entire dataset. These plots are crucial for identifying trends, assessing linearity, and determining the significance of each independent variable. It is important to note that these plots are based on the actual dependent variables, not the predicted ones. Therefore, the relationships depicted are entirely accurate, making these plots an important analytical tool for data analysis.

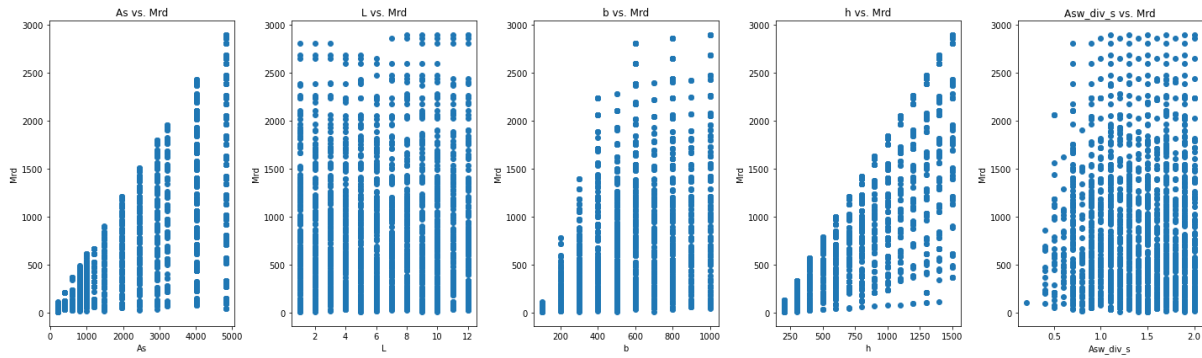


Figure 31: Relationship between the independent variables and the moment capacity

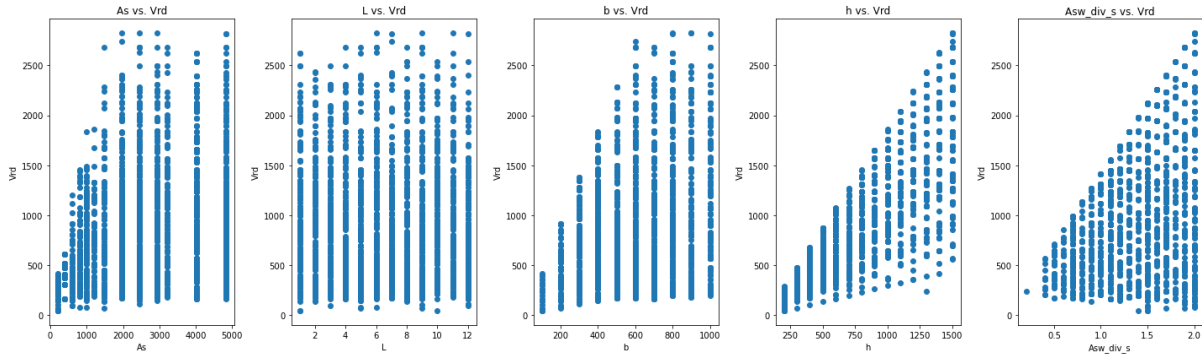


Figure 32: Relationship between the independent variables and the shear capacity

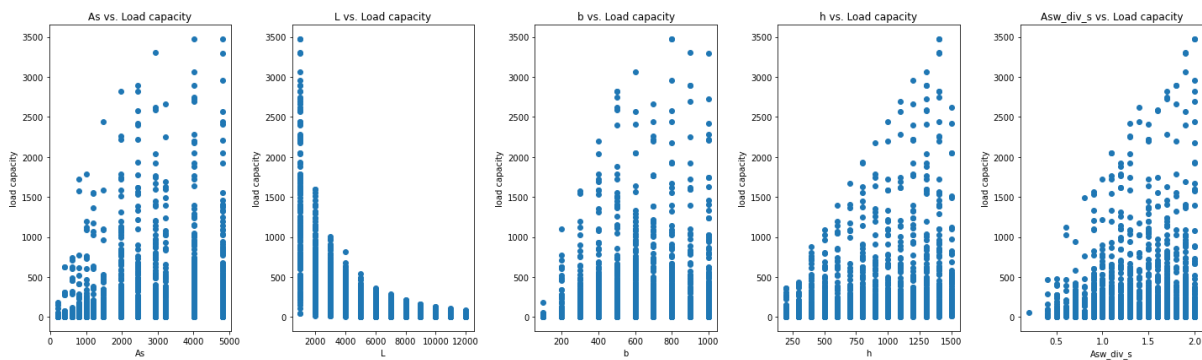


Figure 33: Relationship between the independent variables and the load capacity

Upon examining the influence of independent variables on moment capacity in Figure 31, it becomes evident that the amount of longitudinal reinforcement and height exhibit a nearly perfect linear relationship with moment capacity. Conversely, the width demonstrates a somewhat nonlinear behavior, while the length and shear reinforcement amount seem to have negligible impact.

In Figure 32, which explores the effect of independent variables on shear capacity, the shear reinforcement amount and height display the anticipated linear relationship. However, the width and amount of longitudinal reinforcement exhibit nonlinear behavior.

Figure 33 clearly illustrates the relationship between independent variables and load capacity. The length demonstrates a strong relationship with a decreasing trend, while the longitudinal reinforcement amount and width display a pronounced nonlinear behavior with increasing values. Interestingly, the shear reinforcement seems to maintain a weak linear relationship with load capacity, while the height exhibits a slight stronger linear relationship.

The above assertions are further substantiated by examining the correlation matrix for the independent variables and moment capacity, as depicted in Figure 34. In this matrix, 1 and -1 represent perfect linear and inverse linear correlation, respectively. The height and amount of longitudinal reinforcement show strong positive correlations with moment capacity, with correlation values of 0.77 and 0.61 respectively, although there may be some nonlinearities as the values are not exactly 1. The length and shear reinforcement, with values close to 0, suggests negligible impact on the moment capacity. Despite contributing less than height and longitudinal reinforcement, the width still demonstrates a certain degree of influence on the moment capacity.

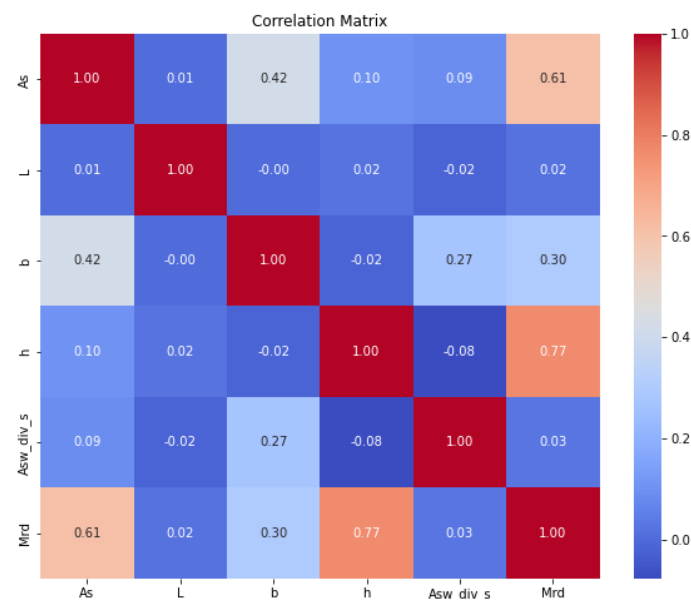


Figure 34: Correlation matrix for the independent variables and the moment capacity

When analyzing the correlation matrix for prediction of the shear capacity (Figure 35), it is clear that height has a strong linear behavior towards the dependent variable, with a score of 0.8. Surprisingly, the shear reinforcement only gets a score of 0.38, which is lower than first

anticipated when looking at the relation between the independent variables and the outcome in Figure 32. Similar to the prediction of moment capacity, the width also exhibits a modest linear correlation of 0.32 when predicting shear capacity.

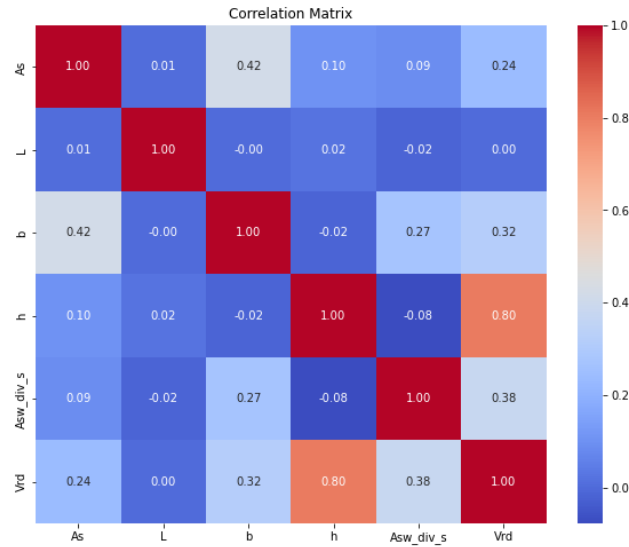


Figure 35: Correlation matrix for the independent variables and the shear capacity

Examining the correlation matrix for the independent variables and load capacities (Figure 36), the results range from 0.16 to 0.30, with the exception of length, which presents a value of -0.6. This suggests that length exhibits a degree of inverse linear behavior, while the other variables predominantly demonstrate nonlinear behavior. Notably, the height, with a value of 0.3, appears to exhibit modest linear behavior.

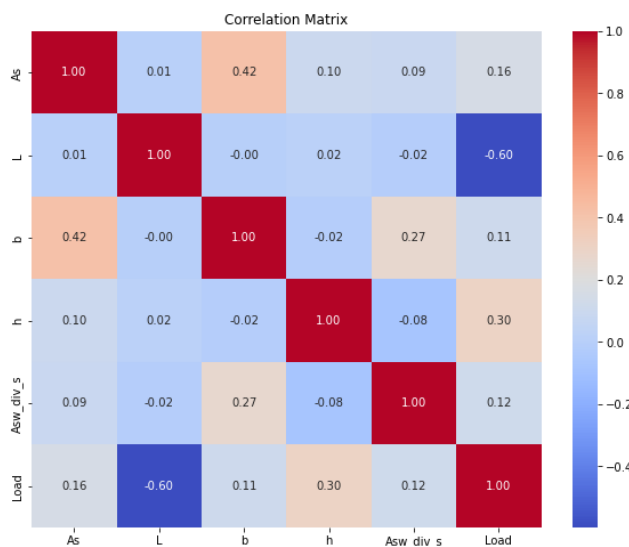


Figure 36: Correlation matrix for the independent variables and the load capacity

Another important observation from examining the correlation matrices is the absence of strong correlations between the independent variables in any of the datasets. This characteristic aligns well with the MLR, which, as discussed in 2.2.3.1, inherently assumes minimal correlation between independent variables.

Lastly, as explained in Chapter 2.2.3.2, SVR is distance-based and uses a kernel function, which is sensitive to the range of the input data. Therefore, we are going to standardize the features when using the algorithm. Although MLR theoretically does not need feature scaling, it is generally advisable to perform it in regression analysis.

6.2. Hyperparameter tuning

In Task 2, we carefully adjusted the hyperparameters of our predictive models, utilizing *RandomSearchCV* for a comprehensive search.

The model predicting moment capacity was optimally tuned with the following parameters:

- Kernel type: Radial basis function (*rbf*)
- Gamma: 0.1
- Epsilon: 0.01
- Regularization parameter (*C*): 200

For shear capacity prediction, *RandomSearchCV* determined the following hyperparameters as optimal:

- Kernel type: Radial basis function (*rbf*)
- Gamma: 0.1
- Epsilon: 0.01
- Regularization parameter (*C*): 600

Finally, in terms of load capacity prediction, the model demonstrated its best performance with these parameters:

- Kernel type: Radial basis function (*rbf*)
- Gamma: *scale*
- Epsilon: 0.01
- Regularization parameter (*C*): 800

The chosen hyperparameters for each model, as tuned through random search, are well-suited for this task for a number of reasons. In all models, the *rbf* kernel was selected. As we discussed in Chapter 2.2.3.2, the *rbf* kernel allows for complex, nonlinear decision boundaries, making it well-suited for our task given some of the complex relationships we have observed in the data.

Additionally, the chosen values for the γ parameter, which controls the "reach" of each training instance in the *rbf* kernel, indicate a balance between bias and variance. In the cases of moment and shear capacity prediction, a γ of 0.1 was chosen. This suggests that the models benefit from a moderate level of complexity, which aligns with our understanding of these structural mechanics phenomena. However, in the case of load capacity prediction, the γ was set to *scale*, adjusting its value based on the variability of the dataset. This could be an indication of a more intricate nonlinear relationship between features and the target variable, potentially reinforcing our suspicions of some of the challenges attached to predicting the load capacity discussed in the preprocessing stage.

The ϵ parameter, set to 0.01 in all cases, denotes a narrow margin of tolerance for errors, suggesting a high level of precision is necessary in our capacity predictions.

The regularization parameter C , which controls the trade-off between achieving a low training error and a low testing error, was set differently for each task. This indicates that the balance between model complexity and generalization varies across the tasks, possibly reflecting the different degrees of complexity in the relationships that these models are trying to capture. Looking at the selected values for C on a general basis, it can be inferred that these values are relatively high. This suggests that the models for these tasks are complex and fit the training data quite closely. It is therefore important to further assess the potential for overfitting.

To do so, we plot the learning curves for each model. These curves are particularly important in SVR algorithms as they provide insights into how well the model is learning and generalizing to unseen data. A typical learning curve for the SVR with a *rbf* kernel shows that the training score remains high regardless of the training set's size, while the test score increases with the size of the training dataset, eventually plateauing. This plateau indicates that adding more training data may not significantly improve the model's generalization performance [85].

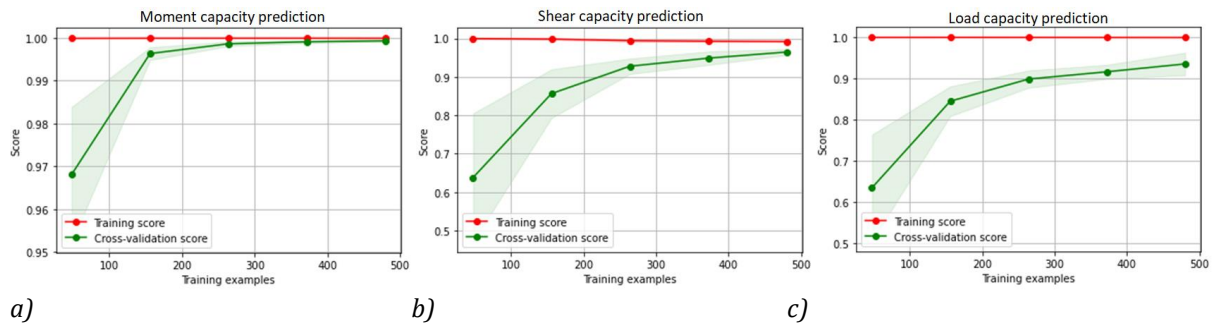


Figure 37: Learning curves for a) Moment capacity, b) Shear capacity, c) Load capacity

In our models, the moment capacity prediction plateaus, while both shear and load capacity continue to rise. This suggests that we could potentially improve the shear and load capacity models' performance by increasing the number of samples. Regarding the proximity of the cross-validation score to the training score, the M_{Rd} ends nearly at the same point, while there is a slight gap for V_{Rd} and a larger one for load. This gap, however, does not compromise the models' performance as the cross-validation scores continue to increase while the training scores remain relatively constant. Thus, our models demonstrate a good balance between bias and variance, effectively handling both underfitting and overfitting scenarios.

6.3. Results

This chapter presents the findings of Task 2, focused on predicting moment capacity, shear capacity, and load capacity using MLR and SVR models. Each capacity prediction will be evaluated using tables that feature the metrics MSE, MAE, and R-squared, which are described in detail in Chapter 2.4.2. Alongside, plots will be provided that illustrate the predicted versus actual values for both models. Furthermore, residual plots for each model will be depicted to assess the accuracy and consistency of the predictions. This comprehensive analysis aims to showcase the potential of these regression models in accurately predicting the capacities of RC rectangular beams.

6.3.1. Moment capacity prediction

In this section, we will present the results derived from the prediction of the moment capacity, alongside an evaluation of the performance of the regression models used in this task.

Table 13: Values of different measurements of the model's accuracy in predicting moment capacity

	MLR	SVR
Mean squared error (MSE)	53731.24	244.15
Mean absolute error (MAE)	172.72	10.56
R-squared	0.87	0.99

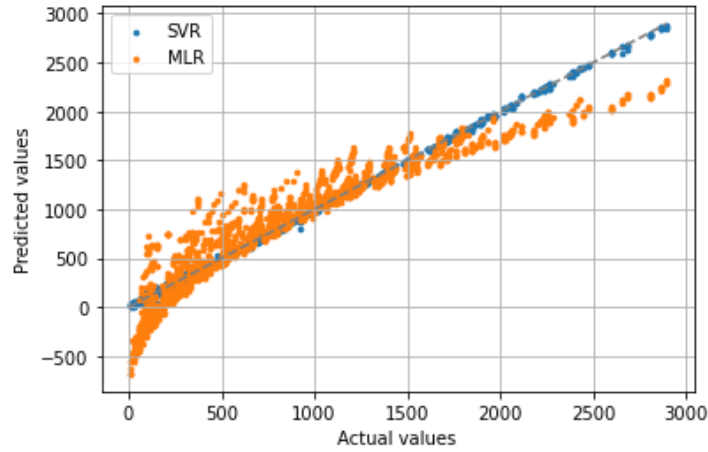


Figure 38: Illustration of the actual values and the predicted moment capacities for both the MLR and the SVR model

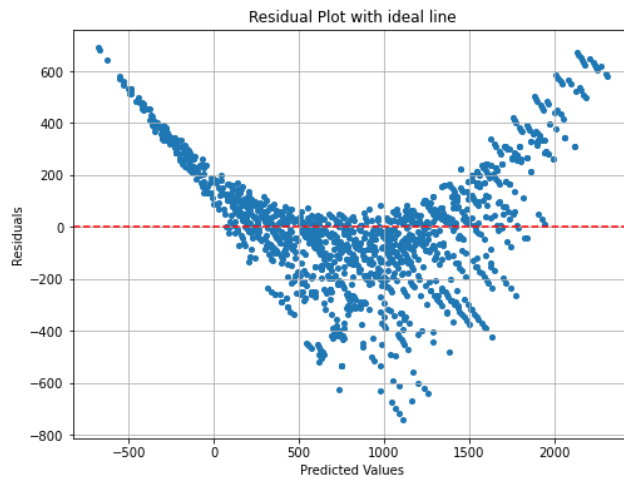


Figure 39: Residual plots for the moment capacity predictions for the MLR model

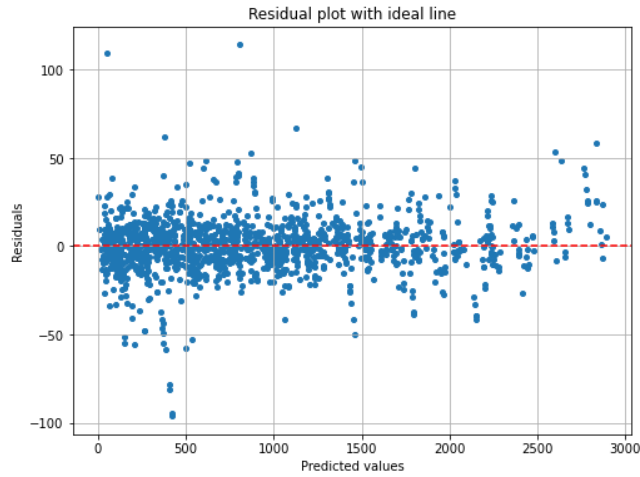


Figure 40: Residual plots for the moment capacity predictions for the SVR model

6.3.2. Shear capacity prediction

This section will present the results from the shear capacity predictions, along with an analysis of the performance of the regression models.

Table 14: Values of different measurements of the model's accuracy in predicting shear capacity

Metric	MLR	SVR
Mean squared error (MSE)	39514.95	6391.16
Mean absolute error (MAE)	153.34	36.46
R-squared	0.88	0.97

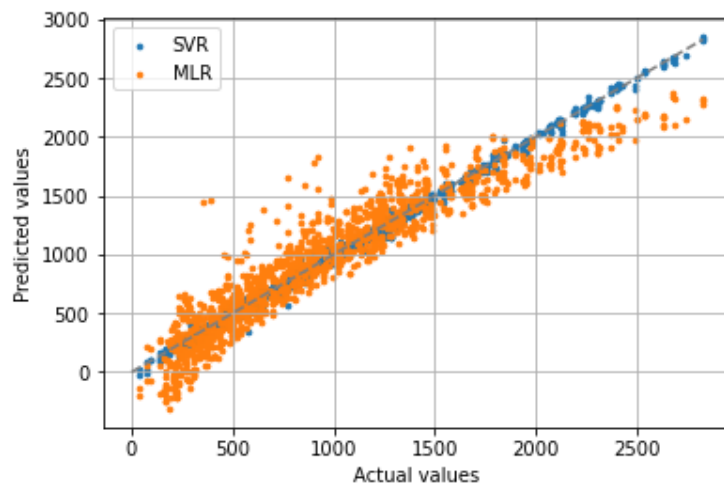


Figure 41: Illustration of the actual values and the predicted shear capacities for both the MLR and the SVR model

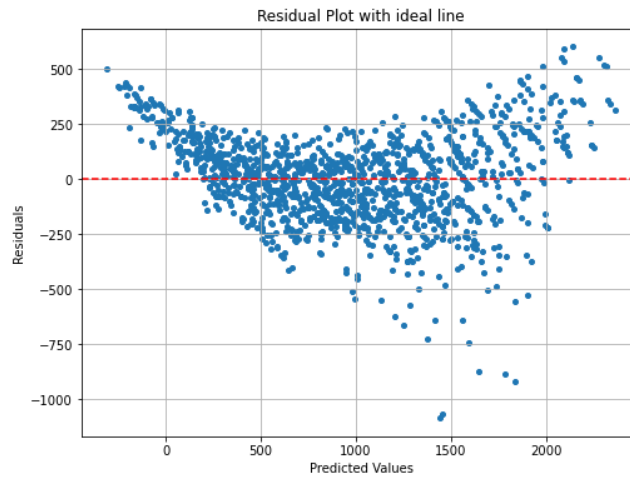


Figure 42: Residual plots for the shear capacity predictions for the MLR model

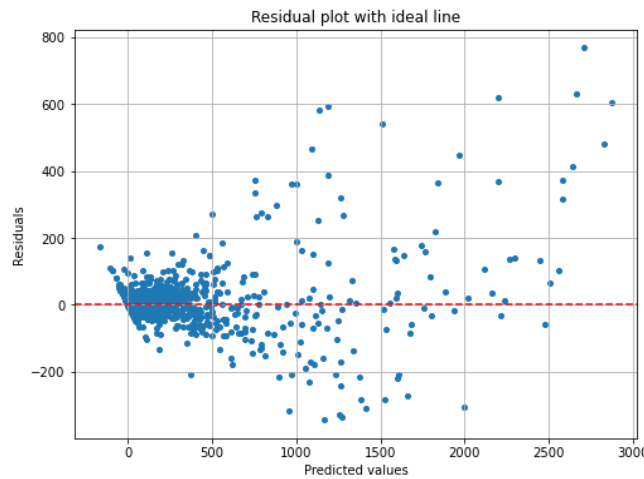


Figure 43: Residual plots for the shear capacity predictions for the SVR model

6.3.3. Load capacity

This section will present the results from the load capacity predictions and the performance of the regression models.

Table 15: Values of different measurements of the model's accuracy in predicting moment capacity

Metric	MLR	SVR
Mean squared error (MSE)	107575.9153	7852.494
Mean absolute error (MAE)	209.73	47.98
R-squared	0.48	0.96

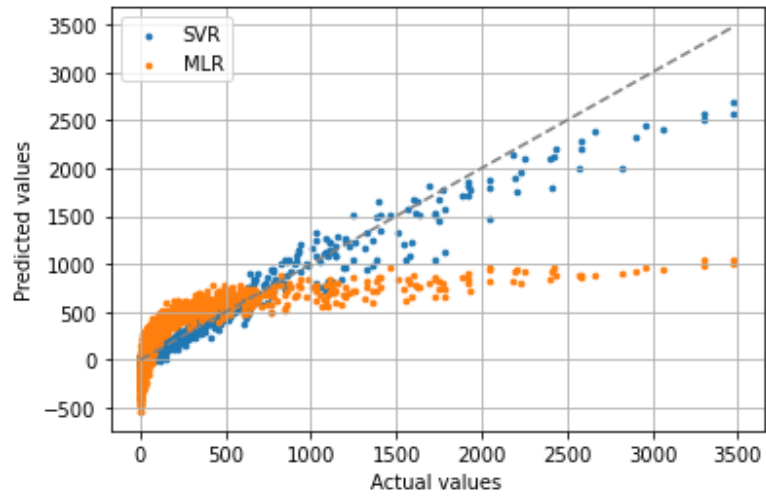


Figure 44: Illustration of the actual values and the predicted load capacities for both the MLR and the SVR model

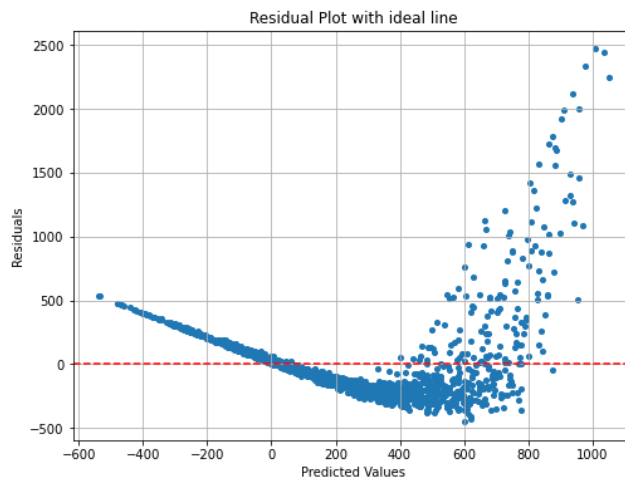


Figure 45: Residual plots for the load capacity predictions for the MLR model

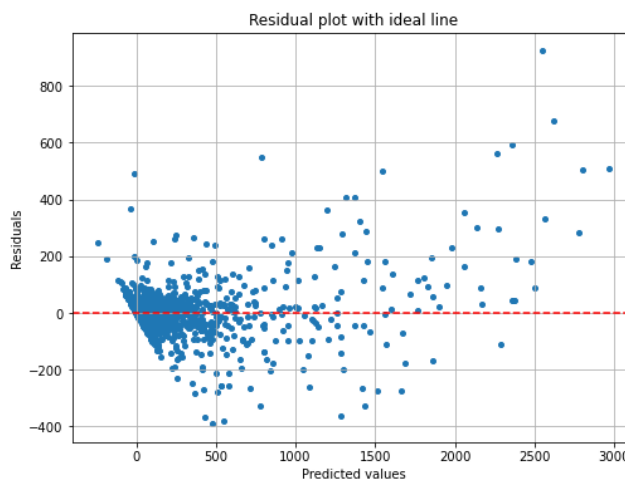


Figure 46: Residual plots for the load capacity predictions for the SVR model

6.4. Discussions

In Task 2, we evaluated the performance of MLR and SVR models for predicting the moment, shear, and load capacities in RC beams. Regarding the predictions for the moment capacity, there are some discrepancies in quality between the two models. The SVR model, with an impressive R-squared score of 99%, outperforms the MLR model with a score of 87%. As random errors are expected to occur, this high R-squared score suggests that the SVR model has very good predictive power, able to explain a large proportion of the variance in the moment capacity using the beams properties. Additionally, despite the MLR model having a much higher MAE value of 173 (indicating larger residuals compared to a value of 11), its R-squared score is still within acceptable limits.

Given that the MSE squares the residuals, the metric tends to inflate larger errors, leading to high MSE values as explained in Chapter 2.4.2.1. These values might not intuitively align with the model performance, particularly when examining the residual plots (Figure 39 and Figure 40), which show outliers but not of an extremely large scale. Consequently, the MSE is considered a less desirable metric in this scenario. Despite this, it is notable that the SVR model has a significantly lower MSE than the MLR model, which suggests better management of larger discrepancies.

The U-shaped distribution observed in the MLR residual plot reinforces our understanding of nonlinearity, as discussed in Chapter 6.1, during the preprocessing stage. The inherent linear assumption in MLR's algorithm restricts its capability to effectively capture and represent these nonlinear trends between the dependent and independent variables of the dataset. Contrastingly, the SVR model, a more flexible and adaptable approach, is designed to handle nonlinear interactions via its kernel function, as explained in Chapter 2.2.3.2. This design proves to be advantageous in our case, as is reflected in superior R-squared and MAE values when compared to the MLR model. Further reinforcing our assertion of non-linearity is the correlation matrix (Figure 34) and the successful application of the *rbf* kernel, as discovered through our random search.

The independent variables' influence on the moment capacity, as showed in Figure 31, can be used to provide some clarity on the almost linear behavior of variables like height and longitudinal reinforcement amount. An increase in either of these parameters invariably leads to a larger capacity.

Conversely, explaining the more nonlinear behavior of width presents more of a challenge. One way is to look at its interactions with other independent variables. For instance, the correlation between width and longitudinal reinforcement amount is 0.42, and between width and shear reinforcement is 0.27. However, a more intuitive explanation emerges when we look at formula (23) in Chapter 4.2, used for calculating moment capacity. Initially, an increase in width contributes to a larger capacity, reflecting a linear relationship. However, beyond a certain point, further widening does not enhance capacity. This could be due to the self-weight of the beam increasing with width, counterbalancing the increased capacity.

The squared d term in the moment capacity formula of Eq. (23) might be why the same phenomenon is not observed with height. This squaring results in the capacity increasing significantly more than the self-weight with rising height, explaining the linear relationship between height and capacity. The correlation matrix also displays this, where the height and the width have a correlation value of 0.77 and 0.3, respectively, towards the moment capacity.

Regardless of these interactions, it is essential to recognize that calculating moment capacity is inherently a nonlinear problem. This explains why the SVR consistently outperforms the MLR in our analyses. It is also important to note that the residual plots for the SVR model during the calculation of moment capacity paint a rather satisfactory picture. Here, the data points are mostly randomly scattered along the zero-line, indicative of a well-fitted model, although a few outliers are present (see Figure 40). We can view this as a healthy model characteristic, showing that the SVR model can indeed handle the nonlinear aspects of this problem efficiently as expected.

Analysis of the evaluation metrics for the shear capacity prediction in Chapter 6.3.2 reveals a similar narrative. The R-squared value of the MLR model in this case is 88%, while the R-squared value of the SVR model is 97%. Additionally, the MAE is 153 as opposed to 36, which denotes larger residuals for the MLR model. It is plausible that the prediction of the shear capacity is a less nonlinear problem than the prediction of the moment capacity, given that the difference in performance for the two algorithms is smaller here than when predicting moment capacity.

Inspecting the predicted vs. actual graph for both models in Figure 41 reveals the SVR model adhering more closely to the perfect-prediction line, compared to the MLR model. This discrepancy becomes more noticeable after 1500 kN, where the MLR model starts to underpredict values significantly. Upon further examination of the residual plots shown in Figure 42 and Figure 43, the limitations of the MLR model are highlighted. At the same time, the SVR

model shows reliable predictions within the 0-600 kN range, encompassing the majority of the cross sections. Beyond this range, the residuals increase proportionally with the shear capacity, implying a slight degree of heteroscedasticity.

Heteroskedasticity is a complicated issue that can surface in regression models, particularly when there is a substantial difference between the smallest and largest values in a dataset [86] - just like in our case, due to the broad ranges we have set. This condition violates the assumption of homoscedasticity, or constant variance, a prerequisite for LRs using ordinary least squares. When the variability of errors is high in certain areas, the model might struggle to provide a good fit. Despite SVR not assuming homoscedasticity, it may still be influenced by these high-error regions during the training process. As a result, the model tends to perform more accurately for cross-sections with lower shear capacities. However, as the capacity increases, the errors may also grow, leading to heteroskedastic residuals, which can explain the lower performance here compared to moment capacity prediction.

Despite the high residuals for larger shear capacities, the good R-squared value is likely due to these extremes pertaining to a small number of cross sections. Therefore, it is sensible to propose the SVR as a tool because of the model's reliable performance within the 0-600 kN range where most cross sections are found, but acknowledge it struggles beyond that.

The U-shaped distribution in the MLR residual plot once again indicates the model's failure to capture the expected nonlinear patterns, as seen in Figure 42. However, this U-shaped trend is less pronounced than in the moment predictions, which strengthens the assumption that this problem involves a higher degree of linearity, and could explain why the MLR model performs marginally better when computing shear capacity as opposed to moment capacity.

In our analysis of feature correlation, we noted a modest linearity in the relationship between shear reinforcement amounts and beam width towards shear capacity, reflected by the correlation values of 0.38 and 0.32, respectively. Upon examination, this may be attributed to the intricate interactions of variables in shear capacity calculation. While shear reinforcement and width indeed enhance a beam's shear resistance, they do not always linearly increase shear capacity. This could be due to the presence of other significant variables such as the beam height, which demonstrates a high correlation of 0.8 and can dilute the effect of other parameters. Furthermore, the transition between the two mechanisms governing shear capacity (Eq. (25) and (27)) could contribute to the observed nonlinearity. The lower correlation of the longitudinal reinforcement (0.28) indicates that its direct impact on shear resistance is less pronounced. This

is logical as the primary role of longitudinal reinforcement is to counteract the tensile forces produced by bending moments in a beam, not necessarily to resist shear forces. Moreover, an increase in longitudinal reinforcement might lead to a shift in the shear failure mode of the beam, introducing a more complex interaction. Conversely, length does not directly impact the outcome as it mainly affects load distribution rather than shear capacity itself.

Predicting load capacity is a complex task that involves taking into account bending moments, shear forces, deflections, and capacities. This additional layer of complexity makes it more challenging than predicting shear and moment capacity alone. Furthermore, the task requires accounting for varying lengths, which were not previously used in calculating moment and shear capacity.

The MLR model, due to its assumption of linear relationships, shows high-amplitude negative values that are not present in the dataset, that has a minimum live load of 0. This confirms our concerns regarding the 0 live load present in the dataset, as discussed in the preprocessing step. The MLR model attempts to fit a straight line and fails to account for scenarios where the self-weight of the beam exceeds its load capacity, resulting in negative predictions. There is considerable variation in the residuals, suggesting that the model tries to accommodate outliers, leading to a regression line that does not accurately represent the majority of data points. Outliers can significantly affect the slope and y-intercept of the line, leading to a lower R-squared value. Figure 45 further demonstrates the inability of the MLR model to capture all nonlinearities, as evidenced by the formation of a steep U-curve.

On the other hand, the SVR model exhibits a high R-squared value of 96%, indicating accurate load prediction capability. However, Figure 44 reveals some discrepancies in its predictions. While the majority of load capacity predictions fall within the range of 0 to 1000 kN (similar to shear capacity), the SVR model struggles to accurately predict values beyond this range. Examining the residual plot in Figure 46, it becomes clearer that the SVR model performs well up to a value of approximately 600 kN, but shows greater deviation and heteroscedasticity for the same reasons as for shear, beyond that range. Some instances of negative load predictions can be observed in the SVR model, although not as prevalent as in the MLR model.

The correlation matrix in Figure 36 highlights the significant role of the length feature, showing an inverse correlation with load capacity. Other independent variables exhibit nonlinear behavior. The substantial degree of nonlinearity in the problem explains the poor performance

of the MLR model and the superior performance of the SVR model, as indicated by the R-squared values.

Outliers were not specifically addressed in this task, but considering the high precision of the SVR model, the influence of outliers on the model's performance seems relatively small. This is consistent with the inherent benefits of SVR models, as discussed in Chapter 2.2.3.2, where they are noted for their resilience against outliers, also reducing the effects of heteroscedasticity.

To summarize, the SVR model performed admirably in all subtasks of Task 2, achieving an R-squared value of 96% or higher. However, it should be used cautiously, as its performance may falter within specific ranges for shear and load capacity, as evidenced by the residual plots and comparison graphs. On the other hand, the MLR model exhibits reasonable accuracy in predicting moment and shear capacity but is currently unsuitable for estimating load capacity.

Ultimately, the selection of the algorithm in Task 2 hinged on the nonlinear nature of the problems at hand, making the SVR model the superior choice overall. Especially when just one of the models can handle nonlinearities, it is simpler to understand the distinctions between the models than it was for Task 1.

Given its performance metrics, MLR might serve as a simpler, more interpretable model for initial insights or for cases where computational resources are limited. However, for complex, nonlinear structural engineering problems like the ones discussed, the SVR model, despite being harder to interpret due to its complexity and less transparent reports, stands out. Its robust performance provides a promising tool for predicting moment, shear, and load capacities of RC beams. This advantage, in context of this task, tends to overshadow its limitations, marking it as a good choice for predicting capacities of RC beams. However, careful interpretation and validation of results are crucial, especially for predictions in the higher range of shear and load capacities.

7. Task 3: Cost optimization

This section aims to explore the potential of ML as a tool for minimizing the cost of simply supported, reinforced ordinary and high-strength concrete T-beams. The cost of construction materials is predominantly determined by the cross-sectional dimensions of the T-beam, the formwork, and the unit costs of concrete and steel reinforcement.

Based on the nature of the dataset, our problem can be described as a multivariate regression problem. We are trying to predict multiple continuous target variables (b , bw , h , d , A_s , hf , $cost$) using a set of input features (fck , fyk , q , g , L , VEd , MEd , $delta_lim$). As proven in the classification Task 1, a NN, specifically a MLP, can handle complex, nonlinear relationships between input and output variables, and is what we will be using for this task. Additionally, it would be interesting to explore how this NN model handles regression problems. For this task, TensorFlow was utilized with the API Keras, as it offers more flexibility and user configuration options compared to scikit-learn when it comes to building and customizing MLP models.

7.1. Data preprocessing

One crucial step in data processing is the identification and management of missing or corrupted data, formatting inconsistencies, and data errors.

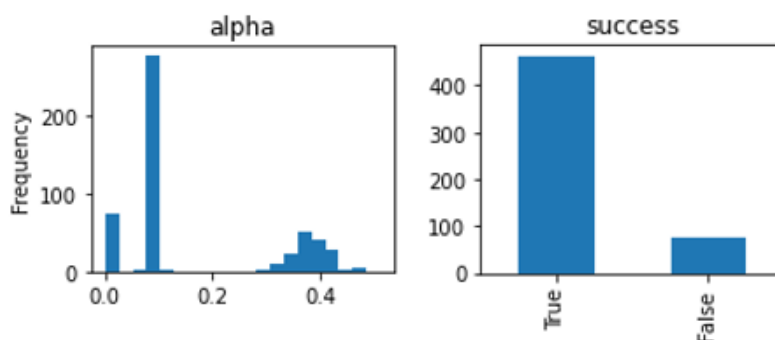


Figure 47: Frequency and value of α -parameter and successful optimization convergences

Upon examining the dataset, it becomes evident in Figure 47 that 76 instances were marked as unsuccessful, which is not surprising given the numerous parameter combinations. The optimization algorithm failed to find a feasible solution for some sections that satisfy all the constraints, resulting in an unsuccessful optimization.

Another reason for the algorithm not converging for some sections could be the termination criteria. The algorithm will stop after 5000 iterations even if a better solution might still be available. When the tolerances are met, the algorithm will assume that it has converged to a solution and therefore also stop, leading potentially to premature termination. Lower tolerance and higher iteration values mean that the algorithm will continue searching for a better solution. However, it is important to balance the need for accuracy with the need for computational efficiency, ensuring that the algorithm does not waste too much time searching for a solution when it already has found one, or when the search for one is unlikely to improve upon further iterations.

Some sections also although converging, assigned the value 0.0 to α . This would imply that there is no concrete in compression, which is not realistic for RC concrete beams under loading. It is important to note that the α value had a lower bound of 0.1 defined. This can happen due to numerical imprecisions, meaning that rounding errors could lead the algorithm to converge for an α value close to 0 even though the lower bound was set to 0.1. Another reason might be that it converged to a suboptimal solution that satisfies the termination criteria but not all the constraints. In this case the algorithm posts success, even though the returning α is incorrect or not optimal.

In addressing the issues above, the decision to remove them from the dataset resulted in a reduction from 540 to 402 instances. By eliminating instances that do not meet the optimization or structural criteria, the remaining data becomes more representative of feasible design solutions. The drawback to this approach is that it may lead to a narrower scope of analysis, excluding potentially some design scenarios that could have provided some valuable insight to the ML model's training.

Lastly, looking at the height of the flange, we see that the optimization algorithm used the lower bound, which in this case was set to 150 mm in all but one optimized section. One possible explanation is that the objective function is designed to minimize cost. A smaller flange height corresponds to reduced material usage and subsequently a lower cost, making it a more attractive choice for the cost function. Finally, the initial guess which also was set to 150mm, might inadvertently have biased the optimization process towards this lower bound solution.

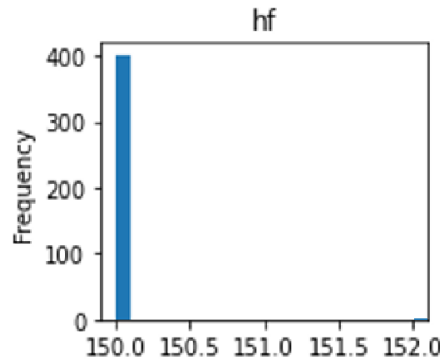


Figure 48: Frequency and value of h_f -parameter in dataset

The lack of diversity in the feature can result in the MLP being unable to capture meaningful patterns or relationships between the feature and other features or target variables, leading to poorer generalization. It will also increase the complexity of the model, without providing any additional information that may improve the performance. Because of its redundancy, it may be beneficial in this case to remove it from the input features from a ML or feature selection perspective. However, in the context of our problem, doing so would remove one of the design variables.

Lastly, as described in the preprocessing of Task 1, the features of the dataset are standardized to ensure that all features contribute on a similar scale.

7.1.1. Dealing with outliers

Box plots are a type of chart that visualizes the distribution of numerical data, including quartiles (or percentiles) and averages, highlighting skewness and dispersion in the data. The five-number summary displayed by box plots includes the minimum score, lower quartile (Q1), median, upper quartile (Q3), and maximum score. Additionally, box plots depict the interquartile range (IQR), which represents the middle 50% of scores. These can be seen in Figure 49a. In our case the box plots are particularly useful for showing the outliers within our dataset. When reviewing a box plot, outliers can be identified as data points that lie outside the whiskers of the box plot, as seen in Figure 49b.

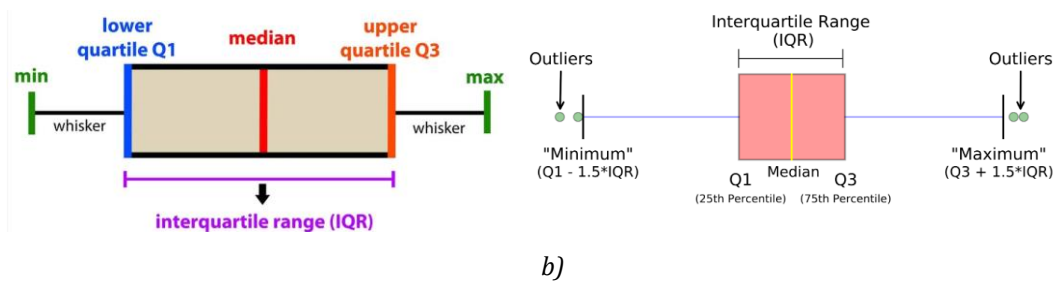


Figure 49: a) Summary; b) Outliers that are outside of the upper and lower quartiles by 1.5 times the interquartile range [87]

The boxplots in Figure 50 reveal the presence of outliers in some of the features within the dataset, especially in V_{Ed} , M_{Ed} , A_s and $cost$. Outliers can negatively impact the performance of a machine learning model, as the model may attempt to fit these outliers, leading to overly complex models and overfitting to the training set. Consequently, this results in reduced performance and weaker generalization capabilities when encountering unseen data. To mitigate this issue, a viable approach could involve defining an appropriate whisker region and eliminating data points that fall outside this region. Subsequently, comparing the model's generalizability before and after the removal of outliers can help assess the effectiveness of this solution.

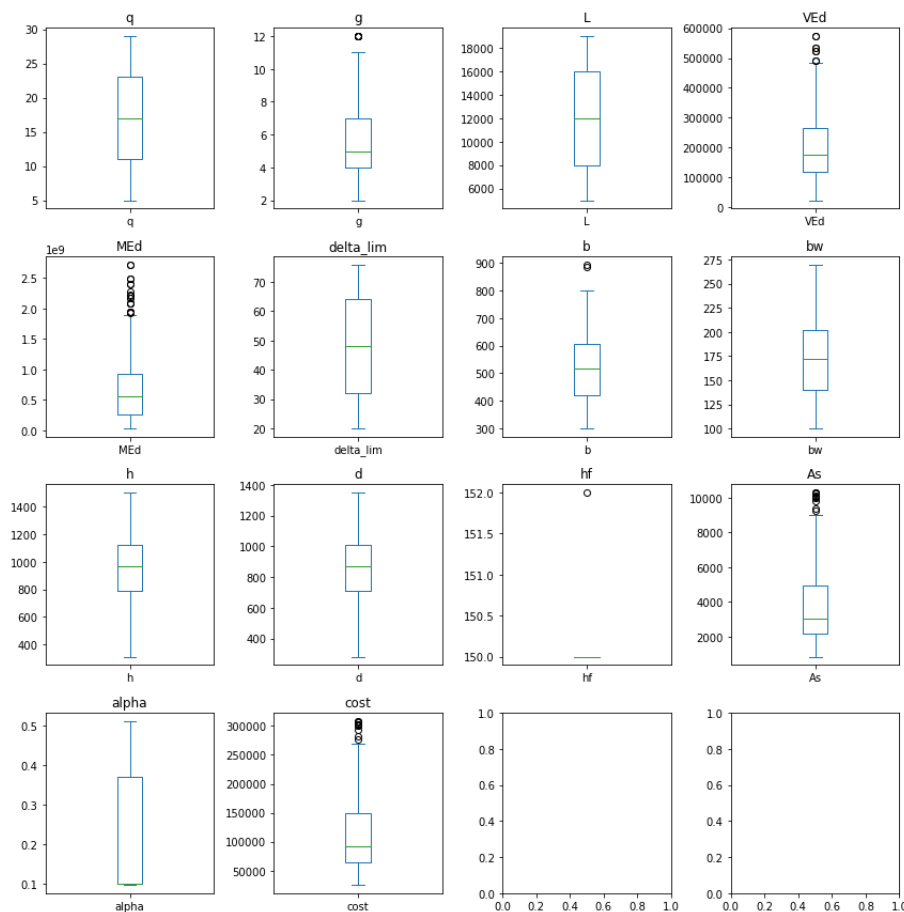


Figure 50: Box plots showing outliers for the entire dataset

It is important to note that since the dataset was built on values that were optimized through a series of constraints before being passed to the dataset, the outliers are characterized as natural, meaning they are most likely not artificial (due to error) and removing them would remove valuable information.

7.1.2. Feature correlation

In the context of regression analysis, it is also important to examine multicollinearity. Multicollinearity occurs when two or more independent variables are highly correlated, making it challenging to estimate the individual effects of each variable on the target variable [88]. This issue can result in inflated standard errors, complicating the interpretation of coefficients and their significance [89]. Although in the case of nonlinear models, such as NNs, multicollinearity tends to not be problematic [90], it remains essential to investigate the relationships between input features to ensure optimal model performance and interpretability.

To identify the presence of multicollinearity in our data, we can examine the correlation matrix, a table that presents the correlation coefficients between every pair of variables in our multivariate dataset. The higher the correlation coefficient between one input variable and another, the more multicollinearity exists. High correlations (close to 1 or -1) indicate strong associations and suggest that these two variables convey similar information, introducing overlapping information to the analysis [89].

	fck	fyk	q	g	L	MEd	VEd	delta_lim
fck	1.00	0.11	-0.03	-0.23	0.03	-0.05	-0.06	0.03
fyk	0.11	1.00	0.01	-0.09	-0.11	-0.08	-0.07	-0.11
q	-0.03	0.01	1.00	0.22	-0.07	0.41	0.62	-0.07
g	-0.23	-0.09	0.22	1.00	0.73	0.78	0.76	0.73
L	0.03	-0.11	-0.07	0.73	1.00	0.80	0.69	1.00
MEd	-0.05	-0.08	0.41	0.78	0.80	1.00	0.96	0.80
VEd	-0.06	-0.07	0.62	0.76	0.69	0.96	1.00	0.69
delta_lim	0.03	-0.11	-0.07	0.73	1.00	0.80	0.69	1.00

Figure 51: Correlation matrix

Upon analyzing the correlation matrix in the figure above, we observe a perfect correlation between L (span of the member) and $delta_lim$ (deflection limit). This is because the two variables have an exact linear relationship ($delta_lim = L/250$). To mitigate the effects of multicollinearity between these two variables, one could consider removing $delta_lim$. Another pair of variables with high correlation are M_{Ed} (bending moment) and V_{Ed} (shear force). Their high correlation can be attributed to their shared dependence on the same variables (q , g , and L) and the similarity of their formulas. In this case, we could consider removing one, or both of these variables.

In the context of optimizing T-sections, these high correlations can hinder our ability to discern the individual contribution of each variable to the overall performance of the structure.

Addressing multicollinearity issues can enhance the interpretability and reliability of the regression model, resulting in a more comprehensive understanding of how each variable affects the optimization process. Furthermore, it is essential to consider the primary goal of the analysis. If the main objective is to predict or forecast the response variable, as in our study, high multicollinearity will not pose a significant problem, as multicollinearity does not reduce the predictive power or reliability of the model, at least within the sample dataset [91]. However, if the goal is to understand the individual effects of each independent variable on the response variable, addressing multicollinearity becomes crucial.

7.2. Hyperparameter tuning

In Chapter 2.2.4, a detailed explanation of the hyperparameters can be found. In the hyperparameter tuning process, the random search method aided in determining the following optimal parameters for the MLP:

- First layer (Input): 1500 neurons with *relu* activation function
- Second layer: 400 neurons with *relu* activation function
- Third layer: 600 neurons with *relu* activation function
- Fourth layer (Output): 6 neurons with *linear* activation function

In addition, the following settings were applied:

- Optimizer *adam*
- Learning rate: 0.0001.
- Loss Function: *mean_absolute_error*
- Training Configuration: The model was trained for 300 epochs with a batch size of 64.

The chosen hyperparameters for the MLP model seem to be quite suitable for the task at hand. First, the structure of the model, with three hidden layers and 1500, 400, and 600 neurons respectively, is a good choice given the number of features and the complexity of the task. The *relu* activation function for the hidden layers is beneficial as it introduces nonlinearity into the model, enabling it to capture complex relationships between the features and target variables.

The linear activation function at the output layer is appropriate for a regression problem like this, where we are trying to predict continuous target variables.

The *adam* optimizer is often a solid choice for training MLPs due to its efficiency and low memory requirements, particularly when dealing with large datasets or many parameters. The chosen learning rate of 0.0001, although on the lower end, can ensure stable training but might also slow down the learning process.

In the context of a design problem, using MAE as both the loss function and evaluation metric might be more appropriate. This is because MAE offers a more interpretable value and is less sensitive to extreme values or outliers, which is advantageous considering the presence of some outliers in our dataset. Alternative methods that exhibit robustness against outliers include the Median Absolute Error, Huber Loss, Quantile Loss, and Tukey's Biweight Loss. Although these methods are not explored in this thesis, their employment is recommended when dealing with the presence of outliers in datasets.

The model was trained for 300 epochs with a batch size of 64. These are reasonable choices given the size of the dataset, but it could be beneficial to further investigate the impact of the number of epochs on the training and validation loss, as the model could potentially benefit from more epochs. Similarly, adjusting the learning rate and observing its impact on the model's performance could also be informative.

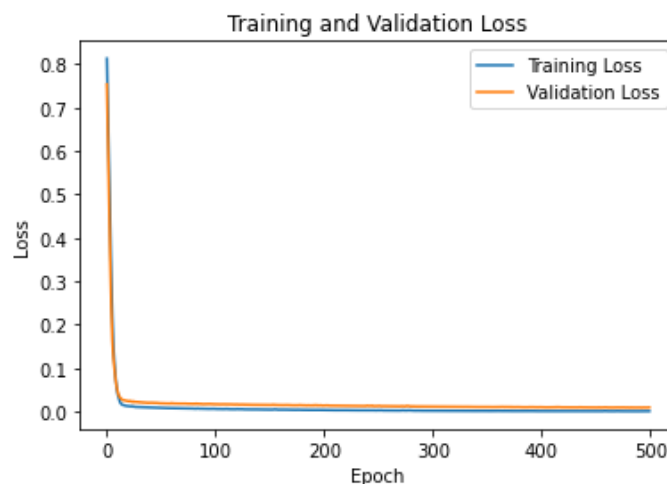


Figure 52: Loss curves over Epochs

Based on the observations of the loss curves in the figure above, it appears that the model is learning effectively during the initial few epochs, as evidenced by the steep decrease in both

training and validation losses. The fact that the training loss is ever so slightly lower than the validation loss suggests that the model is slightly overfitting to the training data, albeit to a very small extent. This is a common phenomenon in ML since the model learns to perform on the training data. The gap seems also to decrease with the number of epochs which indicates that the model is still improving its generalization performance as the training progresses. This as well as the constant small gap is a positive sign, as it suggests that the model is not overfitting significantly and is able to perform well on unseen data. After trying different combinations, 300 epochs gave the best results when accounting for computational time and performance.

After adjusting the learning rates, we also found that the learning rate value of 0.0001 gave the smoothest curve, while higher learning rates made the loss curves oscillate as large weight updates caused the model to overshoot optimal values, as explained in Figure 14.

7.3. Results

This section presents the findings of Task 3, which focused on optimizing the cost of T-sections in the dataset. The objective of this task was to utilize a range of ML approaches to accurately predict the design variables and minimize the cost associated with each T-section. The results offer a detailed analysis of the effectiveness and precision of MLP models in optimizing T-sections based on cost, while maintaining the required structural integrity and standards.

7.3.1. Results on the preliminary dataset

First, we will present the immediate results of our model's predictions focusing on the comparison between the predicted and actual values for each design variable. Graphical representations of these comparisons will be provided, alongside residual graphs, to give a clear visualization of the model's prediction accuracy, and to highlight any potential discrepancies in the data. We will also examine key performance metrics such as MSE, MAE and R-squared (Appendix J4) to evaluate the accuracy and reliability of our models. These metrics provide both a numerical and graphical representation of how closely our predictions align with the actual data. All these metrics, along with the rationale behind their use, are thoroughly explained in Chapter 2.4.2.

This raw look at the model's performance will help to establish a foundational understanding of its initial predictive capabilities before any optimizations are implemented.

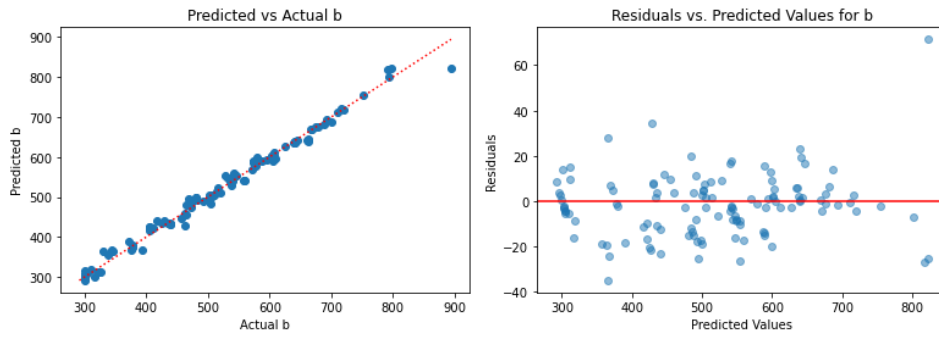


Figure 53: Predicted vs. Actual and residual plot for b

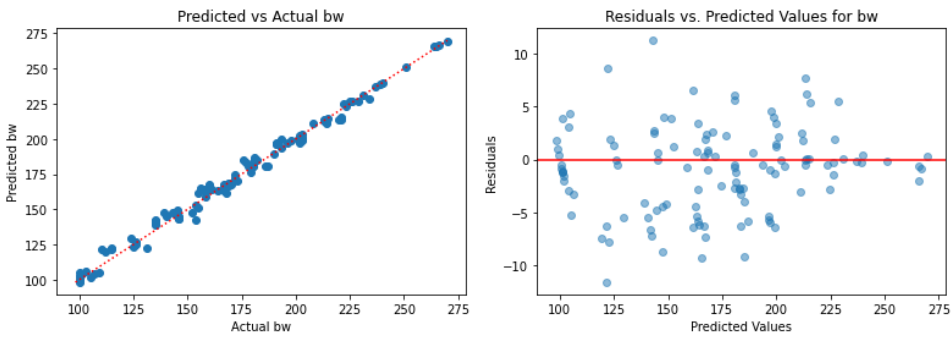


Figure 54: Predicted vs. Actual and residual plot for b_w

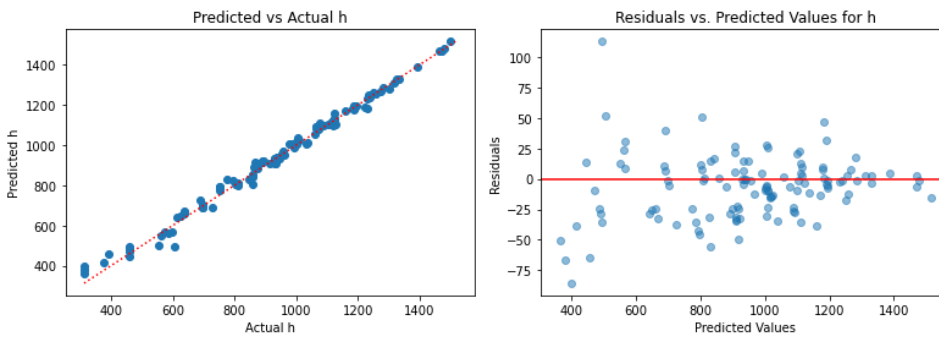


Figure 55: Predicted vs. Actual and residual plot for h

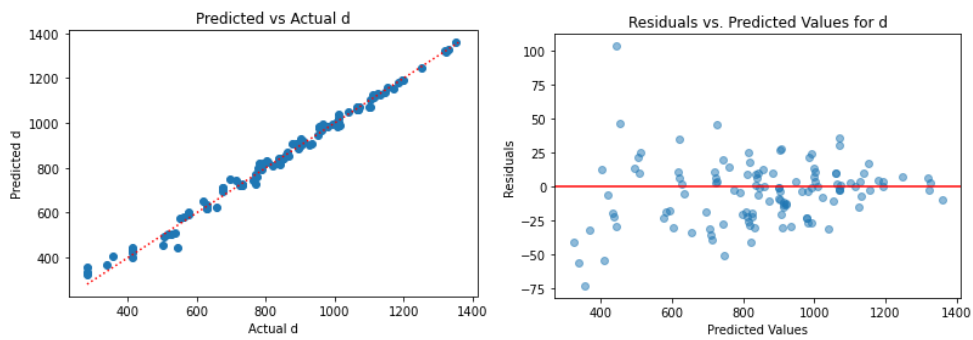


Figure 56: Predicted vs. Actual and residual plot for d

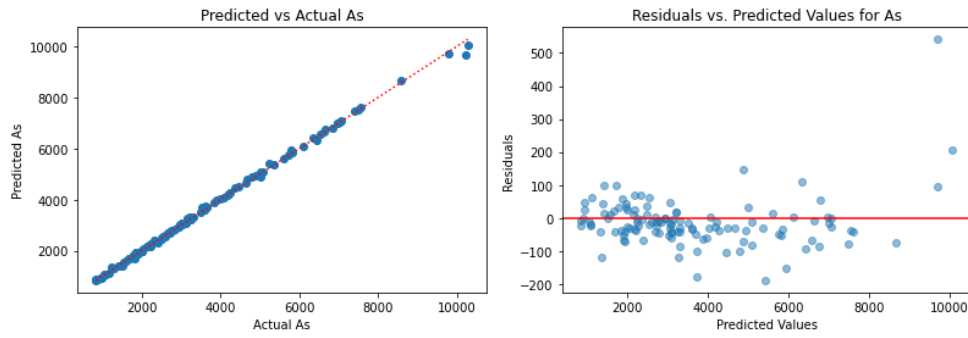


Figure 57: Predicted vs. Actual and residual plot for A_s

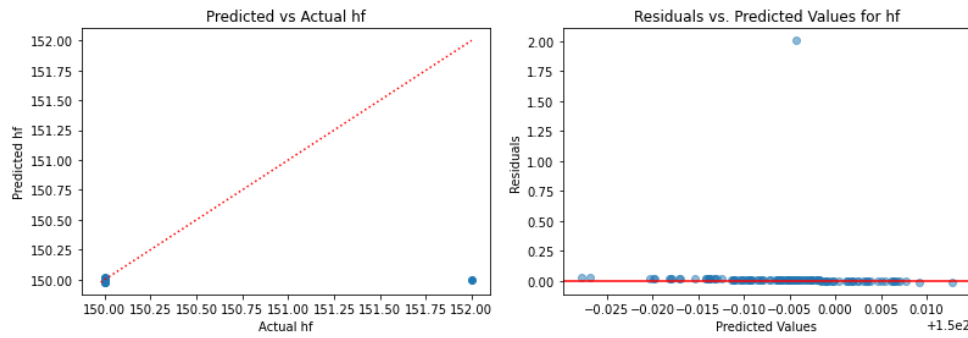


Figure 58: Predicted vs. Actual and residual plot for h_f

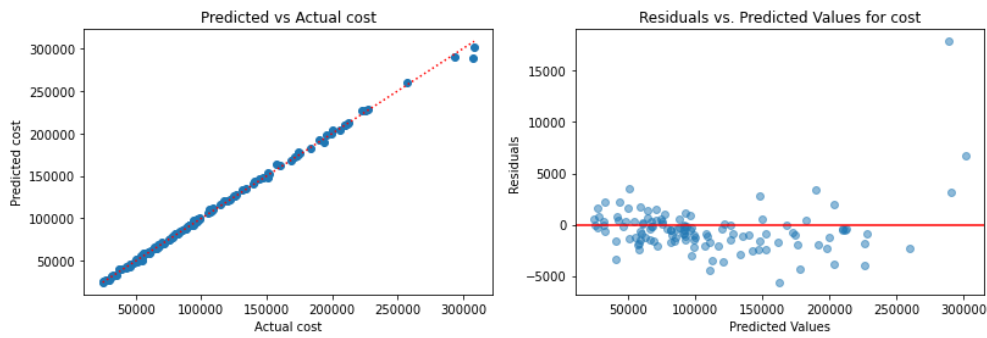


Figure 59: Predicted vs. Actual and residual plot for cost

Table 16: Evaluation metrics of MLP

Metric	MLP
Mean squared error (MSE)	853975.64
Mean absolute error (MAE)	229.88
R-squared	0.84

7.3.2. Results after the removal of `delta_lim` and outliers

By removing the outliers (Appendix J3), as well as the feature `delta_lim`, we aimed to improve the reliability of our model's predictions and mitigate the impact of multicollinearity on our model's stability and interpretability. The IQR multiplier was set to 1.8 to determine the acceptable range for each variable, and we filtered the data by removing any datapoints that fall outside that range. Being able to adjust whiskers by choosing a higher multiplier helped retain more data points, as they do represent valuable information. These approaches are anticipated to enhance the overall performance of the model, ensuring more robust and consistent results in capturing and predicting the underlying patterns within the data. The dataset now has 382 samples.

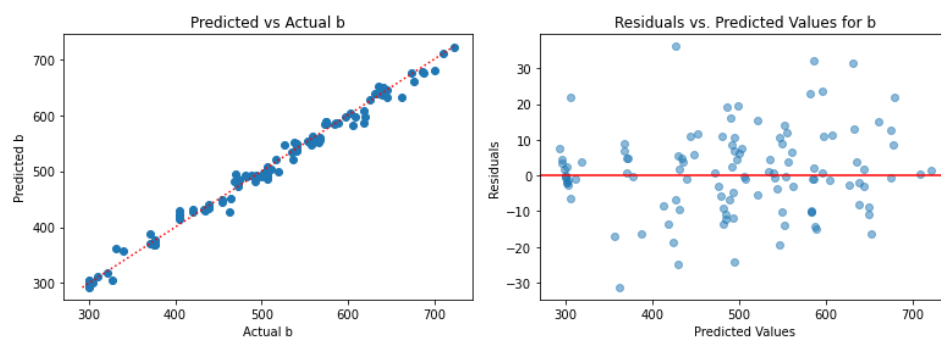


Figure 60: Predicted vs. Actual and residual plot for b after preprocessing

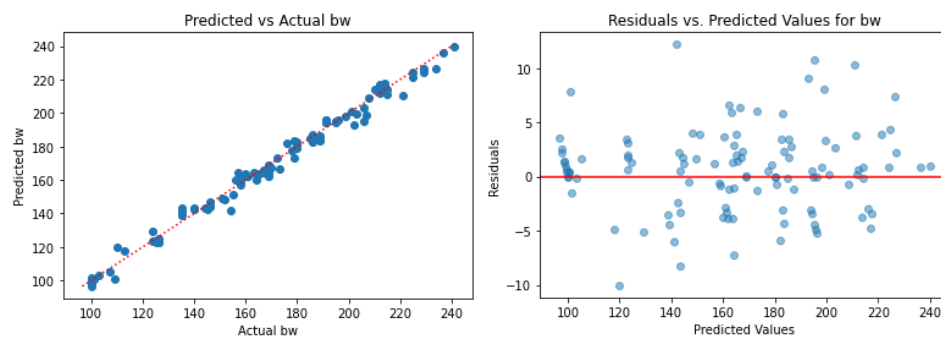


Figure 61: Predicted vs. Actual and residual plot for b_w after preprocessing

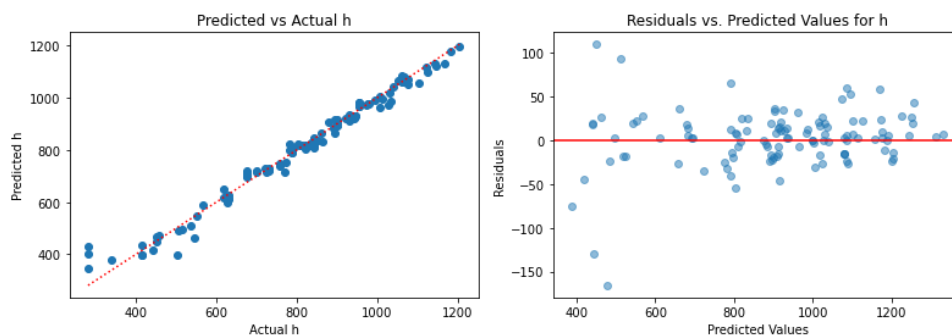


Figure 62: Predicted vs. Actual and residual plot for h after preprocessing

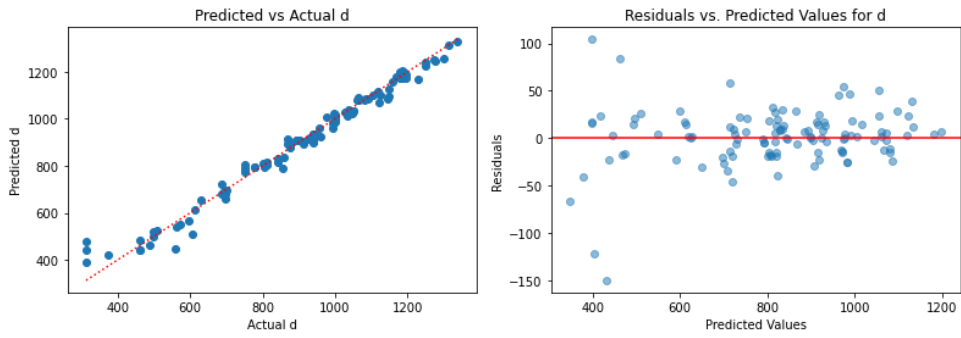


Figure 63: Predicted vs. Actual and residual plot for d after preprocessing

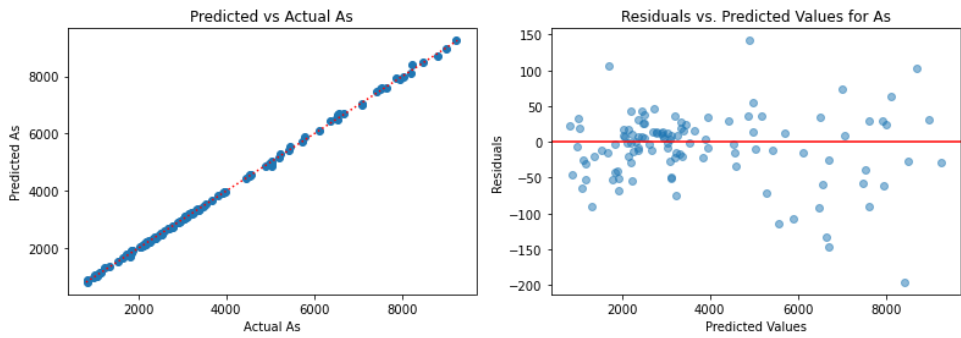


Figure 64: Predicted vs. Actual and residual plot for A_s after preprocessing

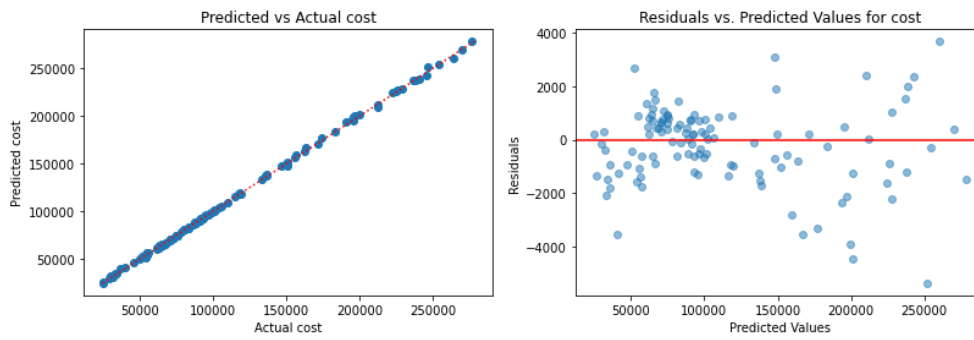


Figure 65: Predicted vs. Actual and residual plot for cost after preprocessing

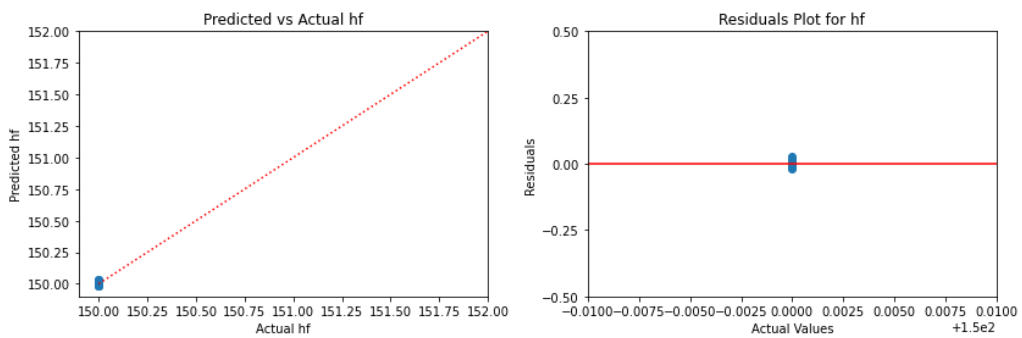


Figure 66: Predicted vs. Actual and residual plot for h_f after preprocessing

Table 17: Evaluation of MLP after preprocessing

Metric	MLP
Mean squared error (MSE)	315433.96
Mean absolute error (MAE)	170.73
R-squared	0.85

7.4. Discussions

In Task 3, we created an MLP model, which demonstrated solid performance in predicting T-section design variables as shown in Chapter 7.3.1, efficiently capturing the complex relationships between input features and targets. The predicted vs. actual plots above highlight this competency, with most predictions aligning closely with the dotted line, signaling an accurate reflection of actual values. Further confirmation of the model's effectiveness comes from the residual plots above, where the majority of residuals are randomly distributed around the zero line, implying random model errors. Upon evaluating the model's performance metrics, we observed a MAE of 230 and an R-squared of 0.84, which, considering the scales of the target variables (Table 2), are indicative of a satisfactory predictive performance.

However, the presence of outliers in the input features and target variables as seen in Chapter 7.1.1, and the residuals in Chapter 7.3.1, signifies the potential for a disproportionate influence on the model's outcomes due to these extreme values, otherwise known as "leverage points". It is noteworthy, though, that these outliers do not significantly deviate from the general trend in the predicted vs. actual plots, suggesting that the model adequately predicts design variables for these datapoints too.

Despite the reassuring model performance, certain residual plots exhibit a slight skewness and a small majority of datapoints located below the zero line, particularly for A_s , b , and $cost$. This indicates a tendency of the model to overestimate the design variables slightly [92] [93], possibly due to the influence of outliers. From a design perspective, erring on the side of caution is common when designing structures. However, given our objective to maximize prediction precision in this scenario, we proceeded to remove outliers, an approach aimed at enhancing the generalizability of our model. This resulted in improved MSE and MAE metrics, while maintaining similar R-squared values, affirming model performance enhancement.

Mild heteroscedasticity was noticeable in the data, particularly in residual plots for b_w , h , and d , with data points initially having a larger range and then converging. As explained in the

discussion part of Task 2, heteroskedasticity is common when there is a broad range of values in the model, as is the case in our task due to the large span of design variable values and its high-dimensionality. Furthermore, the data was produced using an optimization algorithm, which might carry certain biases. These biases could affect the distribution of data points, possibly leading to heteroskedasticity. Despite potentially impacting the MLP model performance, MLP models are typically robust to heteroscedasticity, given their ability to learn complex, nonlinear relationships and adjust input variable weights accordingly, as explained in Chapter 2.2.4.

Nevertheless, heteroscedasticity may influence the training process and model generalizability, warranting the use of robust evaluation metrics like MAE and R-squared, as discussed in Chapter 2.4.2. We attempted to mitigate heteroscedasticity using log and Box-Cox transformations, but found outlier removal more effective in reducing its effect, as demonstrated in Chapter 7.3.2's residual plots.

Correlation between the features h and d , illustrated in their similar plots, is expected due to their inherent relationship in the optimization process. Multicollinearity, a topic broached in Chapter 7.1.2, led us to experiment with removing M_{Ed} , V_{Ed} , and $delta_lim$ in various combinations. Interestingly, omitting only $delta_lim$ yielded superior MLP model performance, suggesting that despite their high correlation, M_{Ed} and V_{Ed} still contribute valuable information to the model's predictive power. Upon removing $delta_lim$ and outliers via the IQR method, we achieved the lowest MSE and MAE values, thereby enhancing prediction accuracy.

As discussed in Chapter 7.1, the optimization algorithm led to a lack of diversity in the flange height feature (h_f), resulting in the lower bound value being used in most cases. Consequently, the model was not able to predict the h_f value of 152 mm, skewing the residual plot and underestimating it. Due to the limiting variability in the feature, the model is not able to generalize to new data points with different h_f values. We therefore can assume that the model has a limited understanding of the effect of the input features on the h_f , resulting in a decrease of accuracy. It would therefore be appropriate to gather additional data points with a wider range of flange heights to improve the model's performance, while maintaining the design significance of the h_f parameter. To achieve a more accurate and reliable model that maintains its relevance to real-world T-section design, it is recommended to revisit the optimization, collect more diverse data points with different flange heights and retrain the model.

Our goal was to maximize prediction precision for optimized design variables of new T-sections, necessitating the minimization of average prediction error magnitudes. While the final model

displayed strong performance in this regard, interpretation of the results should be made with caution, owing to the multicollinearity presence and limited diversity of the h_f target variable in the training set.

A more extensive dataset would dilute outlier impact, enhancing model robustness. Yet, this would not inherently mitigate collinearity, as many variables are intrinsically related. Further collinearity mitigation could benefit from exploring alternatives like dimensionality reduction techniques (e.g., PCA) or regularization techniques (e.g., Ridge or Lasso).

The MLP model exhibits suitability for structural optimization due to its ability to effectively capture complex relationships between variables and provide robust performance, even with challenging characteristics of the dataset, such as heteroscedasticity and outliers. Its ability to learn nonlinear models allows it to handle real-world problems where the underlying relationships are not linear. Moreover, its real-time learning capability, where the model parameters can be incrementally updated as it receives new data, is a significant advantage in situations where the model needs to adapt continuously to new information.

The application of MLP in these types of tasks, while promising, comes with its set of complexities. Firstly, tuning hyperparameters such as the number of hidden layers, neurons, and iterations can pose a significant challenge. These hyperparameters greatly influence the model's performance, but finding the optimal values often requires substantial computational resources and expertise, potentially slowing down the optimization process. Secondly, MLP's sensitivity to the scale and distribution of input features requires careful feature scaling for optimal performance, necessitating an additional preprocessing step. Additionally, MLP's non-convex loss function introduces unpredictability due to possible convergence to sub-optimal solutions, potentially requiring multiple training runs or advanced optimization techniques.

8. Conclusions

This thesis explored the application of ML techniques, specifically supervised learning, within the realm of structural engineering, focusing on the design of RC beams. The investigation was organized around three distinct tasks, each selected to necessitate different key elements of ML for their resolution. This allowed for a comprehensive exploration of ML's various facets, providing valuable insights into the benefits and implications of incorporating ML in structural engineering practice.

Task 1 utilized ML classification techniques - MLP, kNN, and DT classifiers - to identify the anticipated capacity failure mode in RC beams. The MLP classifier demonstrated superior performance in balancing precision and recall across all classes in an imbalanced dataset. The kNN and DT classifiers also offered promising results, with unique strengths and weaknesses in handling specific classes. Despite the challenges posed by the imbalanced dataset, the strategic use of the macro-average f1-score for hyperparameter tuning proved effective. This task underscored the importance of considering multiple evaluation metrics, understanding structural significance of features, and addressing class imbalance.

In Task 2, the focus shifted to regression models for predicting moment, shear, and load capacity in RC beams. The SVR model excelled in precision, even in the presence of outliers and heteroscedasticity. On the other hand, the MLR model performed well for predicting moment and shear capacity but struggled with load capacity due to its inability to handle nonlinearities. The SVR model's superior performance can be attributed to its ability to handle nonlinear interactions. However, it should be used with caution in certain ranges of shear and load capacity. This task showed how regression models can augment traditional prediction methods in structural engineering. It also highlighted the need to choose the right ML algorithm based on the specific problem, demonstrating how versatile ML can be in tackling various issues within structural engineering.

Task 3 delved into the realm of optimization in structural engineering, leveraging an optimized dataset to train a ML model for predicting cost-effective designs. Despite challenges such as outliers, mild heteroscedasticity, and multicollinearity, the MLP model demonstrated robust performance in predicting T-section design variables. The task highlighted the need for diverse data points, particularly for the flange height feature h_f , to improve model generalizability. This task underscored the potential of ML in optimizing structural design and demonstrated how ML

can be used to automate and streamline complex design processes, reducing the time and effort required while maintaining sufficient accuracy.

At the heart of these tasks were digitized adaptations of the EC2 design rules concerning moment, shear, deflection capacities, as well as common practices. This transformation facilitated the generation of comprehensive datasets, effectively guiding the ML models to navigate and learn from a myriad of design scenarios. Consequently, the derived ML models underscored the importance of digital transformation in structural engineering, paving the way for automated design checks and optimization.

Despite the promising results, our models are not without their limitations. However, the thorough documentation presents an opportunity for further refinement. Our research aids the choice of algorithm for tasks that fall into the explored ML categories, through a comprehensive literature review and practical application of ML algorithms. The developed ML models demonstrated their potential as tools for structural engineers, as they provide a starting point for optimization and can serve as potential indicators of structural failure. Moving forward, the focus would involve selecting a specific model and enhancing it over time, through continual adjustments and exposure to new, unseen data. Consistent performance of the model can deem it a reliable tool for structural engineering purposes.

We wish to highlight the potential of the MLP model, which we employed in both multivariate regression and multiclass classification tasks, in this regard. Its adaptability, precision, and real-time learning capability hold considerable promise for structural optimization. MLP's ability to adjust to dynamic environments, where structural conditions can change over time and new findings in structural engineering need to be incorporated, is valuable.

The effectiveness of ML-based prediction models is heavily influenced by the attributes of the training datasets utilized. While assembling data for these datasets, we focused on structural significance, but learned that it is crucial to understand how dataset characteristics, such as multicollinearity, heteroscedasticity, and outliers, can impact various ML models. As such, ML methods should be employed cautiously as tools for structural analysis and design, by individuals that are familiar with ML, given the paramount importance of structural safety in these tasks. This necessitates that structural engineers and practitioners acquire a foundational understanding of ML to effectively develop and validate their ML-based predictive models. In the long term, integrating basic ML-related subjects into civil and structural engineering university curricula could foster and advance the application of ML within the structural engineering community.

8.1. Limitations

Though the approach presented in this paper shows potential, there also exist several limitations, which are listed below:

- **Data limitations:** The research relies on synthetic data generated from known principles and Eurocode design rules. While this ensures control and consistency, it may not fully capture the variability present in real-world, experimental data. Additionally, there is no widely accepted rule to determine the sample size of the training set, and while the number of training samples used in this study was deemed sufficient based on the guidelines presented in Chapter 4, a larger sample size could potentially lead to increased model accuracy, especially for Task 3.
- **Scope of the problem:** The study focuses on rectangular and T-sectioned RC-beams, primarily dealing with uniaxial bending. It also concentrates on a limited set of checks, namely deflection, shear, and moment capacity. Furthermore, the applicability of the specific models developed in this study to more complex problems, such as members subjected to biaxial bending or torsional forces, remains untested.
- **Computational limitations:** The computational resources available for this study have restricted the size of the datasets and the complexity of the models trained. This could have impacted the models' performance and the generalizability of the results.
- **Choice of algorithm:** In this study, we selected and evaluated a range of renowned ML algorithms and found that, for our specific tasks, the MLP yielded the best results for the classification task, and SVM excelled in the regression Task 2. Nevertheless, it is important to acknowledge that there are numerous other algorithms such as Ridge and Lasso Regression, or Gradient Boosting Regressors for regression tasks, and Random Forests for classification tasks that were not explored in this study. These untested algorithms may potentially offer superior performance or provide additional insights. Therefore, the chosen algorithms in this thesis should not be regarded as definitively the best options; there may exist other algorithms better suited to the problem that were not considered in this study.
- **Model interpretability:** One inherent limitation of many ML models, including those used in this study, is a lack of transparency or interpretability. Often, these models function as "black boxes", proficient at predicting outcomes but providing little insight into the underlying processes that lead to these predictions. Despite our best efforts to visualize and interpret the results, this lack of explicability can impede the broader acceptance and application of these models, especially in fields like structural engineering where a deeper understanding of the underlying mechanisms is often essential.

8.2. Future work

Considering the limitations discussed above, the successful application of machine learning in structural engineering will benefit from further research. Potential areas for exploration include the following:

- **Data:** A worthwhile future direction would be to incorporate actual experimental data into the training sets. This would also give a more realistic picture of the number of data points that can be produced in practice, to train a model.
- **Problem:** To take the most advantage of the ML models, the methodologies developed in this thesis could be applied in some not well-resolved problems of structural engineering, like RC members subjected to biaxial bending and torsional forces. The mechanisms of these problems are rather complex, so ML could prove to be a powerful tool in solving them, as they avoid the mechanism-driven numerical models and directly give the results.
- **Model explanation:** To make ML models more persuasive, it is important to find ways to enhance the model explanation further, perhaps through the development and integration of explainable AI techniques, to foster greater trust and acceptance among professionals in the field.
- **Computational power:** As the models become more complex and the datasets larger, the need for increased computational power grows. Future work could investigate the use of higher-capacity hardware, distributed computing, or cloud-based solutions for data storage and processing. This would not only enable the handling of larger datasets, but also potentially improve the efficiency of the model training process.
- **Algorithm selection and optimization:** As previously noted, the choice of learning algorithms for ML applications is abundant, and the optimal choice tends to be dependent on the specific task at hand. This presents a challenge in identifying the most suitable algorithm for a given task, particularly within the domain of structural engineering. Therefore, a significant direction for future work would be to propose a unified methodology for identifying the best-suited algorithm for specific applications in structural engineering. Our research has laid a broad foundation for future exploration in this field. Going forward, it would be useful to select a specific task or problem from this domain, and concentrate on developing and refining the ML algorithm that we have pinpointed as the most suitable for that particular task.

9. References

- [1] H.-T. Thai, "Machine learning for structural engineering: A state-of-the-art review," *Structures*, vol. 38, pp. 448-491, 2022.
- [2] European Committee for Standardization, Eurocode 2: Design of concrete structures - Part 1-1: General rules and rules for buildings, Brussels, Belgium: CEN, 2004.
- [3] E. K. Chong and S. H. Zak, An introduction to optimization, vol. Fourth edition, New Jersey: OHN WILEY & SONS, INC., PUBLICATION, 2013, p. 642.
- [4] Y.-x. Yuan, "A Review of Trust Region Algorithms for Optimization," *ICM99: Proceedings of the Fourth International Congress on Industrial and Applied Mathematics*, 09 1999.
- [5] J. Nocedal and S. J. Wright, Numerical optimization, vol. Second edition, New York: Springer Science+Business Media, LLC, 2006, p. chapter 4.
- [6] scipy, "docs.scipy," 20 12 2022. [Online]. Available: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html#:~:text=and%20lower%20bounds,-,Constrained%20Minimization,-Method%20COBYLA%20uses>. [Accessed 14 05 2023].
- [7] M. Lalee, J. Nocedal and T. Plantenga, "On the Implementation of an Algorithm for Large-Scale Equality Constrained Optimization," *SIAM Journal on Optimization*, vol. 8, no. 3, pp. 682-706, 1998.
- [8] R. H. Byrd, J. C. Gilbert and J. Nocedal, "trust region method based on interior point techniques for nonlinear programming," *Math. Program*, vol. 89, p. 149-185, 2000.
- [9] R. H. Byrd, M. E. Hribar and J. Nocedal, "An Interior Point Algorithm for Large-Scale Nonlinear Programming," *IAM Journal on Optimization*, vol. 9, no. 4, pp. 877-900, 1999.
- [10] T. Hastie, R. Tibshirani and J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, vol. 2nd ed, New York: Springer, 2009.
- [11] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed ed., MIT Press, 2018.
- [12] C. M. Bishop, Pattern Recognition and Machine Learning, New York: Springer, 2006.
- [13] K. Sotiris, "Supervised Machine Learning: A Review of Classification Techniques," *Informatika*, vol. 31, pp. 249-268, 2007.
- [14] JavaTpoint, "JavaTpoint," [Online]. Available: <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>. [Accessed 02 2023].

- [15] O. M. Lior Rokach, "Decision Trees," *The Data Mining and Knowledge Discovery Handbook*, vol. 6, pp. 165-192, 01 2005.
- [16] scikit-learn developers, "scikit-learn," 2023. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>. [Accessed 02 2023].
- [17] J. Brownlee, "machinelearningmastery," 27 November 2020. [Online]. Available: <https://machinelearningmastery.com/overfitting-machine-learning-models/>. [Accessed 02 2023].
- [18] R. G. Mantovani, T. Horváth, R. Cerri, S. B. Junior, J. Vanschoren and A. C. P. d. L. F. d. Carvalho, "An empirical study on hyperparameter tuning of decision trees," 2018.
- [19] O. M. Lior Rokach, "Classification Trees," in *The Data Mining and Knowledge Discovery Handbook*, New York, Springer, 2005.
- [20] L. E. Raileanu and K. Stoffel, "Theoretical comparison between the Gini Index and Information Gain criteria," *Annals of Mathematics and Artificial Intelligence*, vol. 41, p. 77/93, 2004.
- [21] D. Kaplan, "enjoymachinelearning," EML, 10 2022. [Online]. Available: <https://enjoymachinelearning.com/blog/gini-index-vs-entropy/>. [Accessed 06 2023].
- [22] S. Kamperis, "ekamperi.github," 13 04 2021. [Online]. Available: <https://ekamperi.github.io/machine%20learning/2021/04/13/gini-index-vs-entropy-decision-trees.html>. [Accessed 01 2023].
- [23] S. S. Mullick, S. Datta and a. S. Das, "Adaptive Learning-Based k-Nearest Neighbor Classifiers With Resilience to Class Imbalance," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1-13, 2018.
- [24] Z. Shi, "Improving k-Nearest Neighbors Algorithm for Imbalanced Data Classification," *IOP Conference Series: Materials Science and Engineering*, vol. 719, no. 1, 2020.
- [25] scikit-learn developers, "scikit-learn," 2023. [Online]. Available: <https://scikit-learn.org/stable/modules/neighbors.html>. [Accessed 03 2023].
- [26] A. AnAj, Artist, *KnnClassification.svg*. [Art]. Wikipedia commons, 2007.
- [27] i. V. Degtyarev and K. D. Tsavdaridis, "Buckling and ultimate load prediction models for perforated steel beams using machine learning algorithms," *Journal of Building Engineering*, vol. Volume 51, 2022.
- [28] S. Vijendra and P. Shivani, "Robust Outlier Detection Technique in Data Mining: A Univariate Approach," *Computer Vision and Pattern Recognition*, 2014.

- [29] C. Sima and E. R. Dougherty, "The peaking phenomenon in the presence of feature-selection," *Pattern Recognition Letters*, vol. 29, no. 11, pp. 1667-1674, 2008.
- [30] scikit-learn developers, "scikit-learn," 2023. [Online]. Available: <https://scikit-learn.org/stable/modules/neighbors.html#>. [Accessed 03 2023].
- [31] V. Kanade, "spiceworks," 3 april 2023. [Online]. Available: <https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-linear-regression/>. [Accessed 02 2023].
- [32] A. Gupta, A. Sharma and A. Goel, "Review of Regression Analysis Models," *International Journal of Engineering Research & Technology*, vol. 6, no. 8, 2017.
- [33] M. A. Iqbal, "Application of Regression Techniques with their Advantages and Disadvantages," p. 17, september 2021.
- [34] scikit-learn developers, "scikit-learn," 2023. [Online]. Available: https://scikit-learn.org/stable/tutorial/statistical_inference/supervised_learning.html#linear-regression. [Accessed 04 2023].
- [35] "jvatpoint," [Online]. Available: <https://www.jvatpoint.com/machine-learning-support-vector-machine-algorithm>. [Accessed 02 2023].
- [36] A. Sethi, 24 april 2023. [Online]. Available: <https://www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/>. [Accessed february 2023].
- [37] N. C. Berke Akkaya, "Comparison of Multi-class Classification Algorithms on Early Diagnosis of Heart Diseases," in *y-BIS 2019 Conference: ISBIS Young Business and Industrial Statisticians Workshop on Recent Advances in Data Science and Business Analytics*, Istanbul, Turkey, 2019.
- [38] scikit-learn developers, 2023. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html>. [Accessed april 2023].
- [39] scikit-learn developers, 2023. [Online]. Available: https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html#:~:text=Intuitively%20C%20the%20gamma%20parameter%20defines,the%20model%20as%20support%20vectors. [Accessed april 2023].
- [40] s.-l. developers, "scikit-learn," 2023. [Online]. Available: https://scikit-learn.org/stable/modules/neural_networks_supervised.html. [Accessed 03 2023].
- [41] C. M. Bishop and G. Hinton, *Neural Networks for Pattern Recognition*, New York: Oxford University Press, 1995, p. 92.

- [42] Keras developers, "Keras," 2023. [Online]. Available: <https://keras.io/about/#:~:text=Keras%20is%20the%20high%2Dlevel,solutions%20with%20high%20iteration%20velocity..> [Accessed 05 2023].
- [43] J. Lederer, "Activation Functions in Artificial Neural Networks: A Systematic Overview," *Computing Research Repository*, 2021.
- [44] L. Ven and J. Lederer, "Regularization and Reparameterization Avoid Vanishing Gradients in Sigmoid-Type Networks," *CoRR*, 2021.
- [45] J. Brownlee, "Machine Learning Mastery," 22 01 2021. [Online]. Available: <https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/>. [Accessed 04 2023].
- [46] C. E. Nwankpa, W. Ijomah, A. Gachagan and S. Marshall, "Activation Functions: Comparison of Trends in Practice and Research for Deep Learning," *CoRR*, 2018.
- [47] R. Pascanu, T. Mikolov and Y. Bengio, "On the difficulty of training Recurrent Neural Networks," *Computing Research Repository*, 2012.
- [48] TensorFlow Authors, "keras," 2015. [Online]. Available: <https://keras.io/api/layers/activations/#softmax-function>. [Accessed 05 2023].
- [49] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016.
- [50] J. Jordan, "jeremyjordan," 01 03 2018. [Online]. Available: <https://www.jeremyjordan.me/nn-learning-rate/>. [Accessed 03 2023].
- [51] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1, p. 503–528, 1989.
- [52] O. B. Leon Bottou, *The Tradeoffs of Large Scale Learning*, Red Hook, NY: Curran Associates Inc., 2007, p. 161–168.
- [53] S. Ruder, "An overview of gradient descent optimization algorithms," 2017.
- [54] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *International Conference on Learning Representations*, 2014.
- [55] J. Brownlee, "Machinelearningmastery," 10 08 2022. [Online]. Available: <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/>. [Accessed 05 2023].
- [56] TensorFlow, "tensorflow," 20 05 2023. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/Model?version=nightly#fit. [Accessed 05 2023].

- [57] A. Devarakonda, M. Naumov and M. Garland, "AdaBatch: Adaptive Batch Sizes for Training Deep Neural Networks," *CoRR*, 2017.
- [58] scikit-learn developers, "scikit-learn," 2023. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. [Accessed 02 2023].
- [59] scikit-learn developers, "scikit-learn," 2023. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html. [Accessed 02 2023].
- [60] K. Zakka, "github," 13 July 2016. [Online]. Available: <https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/#more-on-k>. [Accessed 03 2023].
- [61] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *The Journal of Machine Learning Research*, vol. 13, p. 281–305, 2012.
- [62] A. Kumar, "vitalflux," Data Analytics, 17 March 2023. [Online]. Available: https://vitalflux.com/accuracy-precision-recall-f1-score-python-example/#What_is_Precision_Score?utm_content=cmp-true. [Accessed 02 March 2023].
- [63] A. Suresh, "Analytics Vidhya," 17 November 2020. [Online]. Available: <https://medium.com/analytics-vidhya/what-is-a-confusion-matrix-d1c0f8feda5>. [Accessed March 2023].
- [64] A. Hooshmand, "Accurate Diagnosis of Cancer by Machine Learning Classification of the Whole Genome Sequencing Data," 2020.
- [65] V. Plevris, G. Solorzano, N. P. Bakas and M. E. A. Ben Seghier, "Investigation of performance metrics in regression analysis and machine learning-based prediction models," in *8th European Congress on Computational Methods in Applied Sciences and Engineering*, Oslo, Norway.
- [66] A. Botchkarev, "Evaluating Performance of Regression Machine Learning Models Using Multiple Error Metrics in Azure Machine Learning Studio," *SSRN Electronic Journal*, 2018.
- [67] scikit-learn developers, "scikit-learn," 2023. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html. [Accessed 05 2023].
- [68] wikipedia, "Wikipedia," 03 04 2023. [Online]. Available: https://en.wikipedia.org/wiki/Coefficient_of_determination#Adjusted_R2. [Accessed 05 2023].

- [69] "stackoverflow," april 2021. [Online]. Available: <https://stackoverflow.com/questions/39208718/predicted-vs-actual-plot>. [Accessed 02 2023].
- [70] J. Jobson, Applied multivariate data analysis, New York, NY: Springer, 1999.
- [71] U. Gohar, "towardsdatascience," 05 03 2020. [Online]. Available: <https://towardsdatascience.com/how-to-use-residual-plots-for-regression-model-validation-c3c70e8ab378>. [Accessed 04 2023].
- [72] European Committee for Standardization, EN 1990: Eurocode - Basis of structural design, Brussels, Belgium: CEN, 2002.
- [73] spyder, [Online]. Available: <https://www.spyder-ide.org/>.
- [74] scikitt-learn, [Online]. Available: <https://scikit-learn.org/stable/>.
- [75] SciPy.optimize, [Online]. Available: <https://docs.scipy.org/doc/scipy/reference/optimize.html>.
- [76] F. Ferhat, "Design Optimization of Reinforced Ordinary and High-Strength Concrete Beams with Eurocode2 (EC-2)," in *Optimum Composite Structures*, Rijeka, IntechOpen, 2018.
- [77] A. Bryman, Social Research Methods. 5th ed, London, England: Oxford University Press, 2015, p. p. 144.
- [78] D. J. Mundfrom, D. G. Shaw and T. L. Ke, "Minimum Sample Size Recommendations for Conducting Factor Analyses," *International Journal of Testing*, vol. 05, no. 02, pp. 159-168, 2005.
- [79] S. I. Sørensen, betongkonstruksjoner, trondheim: fagbokforlaget, 2013.
- [80] SciPy. [Online]. Available: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html#:~:text=and%20lower%20bounds.-,Constrained%20Minimization,-Method%20COBYLA%20uses>.
- [81] N. Andrei, "Sequential Quadratic Programming. In: Modern Numerical Nonlinear Optimization," in *Modern Numerical Nonlinear Optimization*, Cham, Springer International Publishing, 2022, pp. 521--567.
- [82] A. Kumar, "vitaflux," 31 07 2022. [Online]. Available: <https://vitalflux.com/how-know-data-linear-non-linear/>. [Accessed 05 2023].
- [83] F. DERNONCOURT, "datascience.stackexchange," 2017.

- [84] S. Mehta, "analyticsindiamag," 26 July 2022. [Online]. Available: <https://analyticsindiamag.com/why-should-the-learning-rate-always-be-low/>. [Accessed March 2023].
- [85] scikit-learn developers, "scikit-learn," 2023. [Online]. Available: https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html. [Accessed 05 2023].
- [86] CFI Team, "corporatefinanceinstitute," 07 05 2023. [Online]. Available: <https://corporatefinanceinstitute.com/resources/data-science/heteroskedasticity/>. [Accessed 06 2023].
- [87] S. Mcleod, "simplypsychology," 17 05 2023. [Online]. Available: <https://www.simplypsychology.org/boxplots.html>. [Accessed 04 2023].
- [88] A. HAYES, "investopedia," 25 02 2023. [Online]. Available: <https://www.investopedia.com/terms/m/multicollinearity.asp>. [Accessed 05 2023].
- [89] A. Siegel, Practical Business Statistics, vol. 7th, Academic Press, 2016.
- [90] R. D. D. Veaux and L. H. Ungar, "Selecting models from data. Lecture Notes in Statistics, vol 89," in *Multicollinearity: A tale of two nonparametric regressions*, New York, Springer, 1994, p. 393–402.
- [91] "Wikipedia," 05 06 2023. [Online]. Available: <https://en.wikipedia.org/wiki/Multicollinearity>. [Accessed 05 2023].
- [92] Khan Academy, "khanacademy," 2022. [Online]. Available: <https://www.khanacademy.org/math/statistics-probability/describing-relationships-quantitative-data/regression-library/a/introduction-to-residuals>. [Accessed 05 2023].
- [93] Khan Academy, "khanacademy," 2022. [Online]. Available: <https://www.khanacademy.org/math/ap-statistics/bivariate-data-ap/xfb5d8e68:residuals/e/residual-plots>. [Accessed 04 2023].

10. Appendices

The accompanying appendices to this thesis are available digitally and can be accessed upon request. Please feel free to contact either the authors or the Norwegian University of Science and Technology (NTNU) for access to these materials.

A1 - Functions used for dataset generation for Task 1&2

A2 - Dataset creation script for Task 1&2

A3 - CSV-file script for Task 1&2

A4 - CSV-file shorten script for Task 1&2

B1 - kNN model for Task 1

B2 - Hyperparameter tuning of kNN

C1 - DT model for Task 1

C2 - Hyperparameter tuning of DT

D1 - MLP model for Task 1

D2 - Hyperparameter tuning of MLP

D3 - Random Under Sampler for Task 1

E - Visualization & Metrics for classification

F - MLR model for Task 2

G1 - SVR model for Task 2

G2 - Hyperparameter tuning of SVR

H - Visualization & Metrics for Regression

I - Code related to dataset generation for Task 3

J1 - MLP model for Task 3

J2 - Hyperparameter tuning of MLP

J3 - Removal of outliers for Task 3

J4 - Metrics for regression Task 3