DESIGN SPACE EXPLORATION AND OPTIMIZATION OF MECHANICAL COMPONENTS USING MACHINE-LEARNING TECHNIQUES

AKLILU TEKLEMARIAM

HAWASSA UNIVERSITY, HAWASSA, ETHIOPIA

October 2023

DESIGN SPACE EXPLORATION AND OPTIMIZATION OF MECHANICAL COMPONENTS USING MACHINE-LEARNING TECHNIQUES

AKLILU TEKELMARIAM

MAJOR ADVISOR: Dr. TINSAE GEBRECHRISTOS

CO-ADVISOR: SOLOMON DAMENA

A THESIS SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE

HAWASSA UNIVERSITY

INSTITUTE OF TECHNOLOGY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE

HAWASSA, ETHIOPIA

October 2023

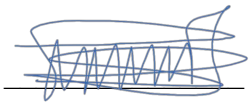
APPROVAL SHEET-I

This is to certify that the thesis entitled “Design Space Exploration and Optimization of Mechanical Components Using Machine-Learning Techniques” Submitted in partial fulfillment of the requirements for the degree of Master's with specialization in Computer Science, the Graduate Program of the Department/School of Informatics, and has been carried out by Aklilu Teklemariam. Therefore, we recommend that the student has fulfilled the requirements and hence hereby can submit the thesis to the department.

Tinsae Gebechristos	_____	13/10/2023
Name of major advisor	Signature	Date
_____	_____	_____
Name of co-advisor	Signature	Date

APPROVAL SHEET-II

We, the undersigned, members of the Board of Examiners of the final open defense by Aklilu Teklemariam have read and evaluated his/her thesis entitled "Design Space Exploration and Optimization of Mechanical Components Using Machine-Learning Techniques" and examined the candidate. This is, therefore, to certify that the thesis has been accepted in partial fulfillment of the requirements for the degree.

Tinsae Gebrechristos		
Name of Major Advisor	Signature	Date
Name of Internal Examiner-I	Signature	Date
Name of Internal Examiner-II	Signature	Date
Michael Melese (PhD)		19-Dec-2023
Name of External Examiner	Signature	Date
SGS Approval	Signature	Date

Final approval and acceptance of the thesis is contingent upon the submission of the final copy of the thesis to the School of Graduate Studies (SGS) through the Department/School Graduate Committee (DGC/SGC) of the candidate's department.

Stamp of SGS Date: _____

Remark

- Use this form to submit the thesis with ***MINOR CORRECTION*** suggested by the examining board.
- 3 copies

Acknowledgment

First, thank the Almighty God for helping me complete this research work. Next, I would like to express my sincere gratitude to my advisor, Dr. Tinsae Gebrechristos, for his valuable guidance, motivation and encouragement during this research work. His encouragement to pursue the direction I want with my freedom has been incredible and inspired me to reach my goal. Working with him, I have also developed professionally and continue to be driven for further research in the future. I also thank my co-advisor, Mr. Solomon Damena, for his motivation and encouragement. Finally, I thank all computer science department staff who taught me courses for their effort and motivation to provide knowledge and skills in computer science. I specially acknowledge Dr. Anitha for her valuable teaching and for providing information about active research areas during class lessons, which helped me with this thesis work. Finally, I thank all my classmates for sharing information during the course and research and actively participating in group work.

Abstract

In today's highly competitive market, rapid product development with minimal resource utilization is crucial. This requires thoroughly evaluating all available design solutions to identify the most efficient options. While various techniques exist to explore design spaces and achieve optimal results, continuous research is dedicated to discovering more effective approaches that can be adapted to diverse design challenges.

This thesis examines how supervised machine-learning techniques can be used to explore and optimize mechanical component design problems. Specifically, it focuses on three mechanical component design problems: pressure vessel design, helical coil spring design, and belt pulley drive design. Four machine learning classification models (support vector machine, random forest, gaussian naïve Bayes, and neural network) are tested.

We prepared three different dataset sizes for both binary and multiclass classification using simulation-based design of experiments to investigate how dataset size affects model performance. We used the Latin hypercube sampling method to effectively sample points from the available design space. Additionally, hyperparameter tuning was performed to improve the performance of the evaluated models.

Based on our findings, the random forest and support vector machine models outperform the others. Specifically, the random forest model excels in all three design problems for binary and multiclass classifications across various dataset sizes, even with default parameters. However, the support vector machine and neural network model can surpass the random forest's performance when hyperparameters are fine-tuned. On the other hand, the Gaussian naïve Bayes model exhibits the lowest accuracy in all three design problems. Interestingly, regardless of dataset size, there are no significant variations in the classifiers' performance for both binary and multiclass classifications. This suggests that the classifiers' effectiveness relies more on the dataset's representation of the original distribution than its size. This implies that reducing the sampling budget is possible using a small number of data points that accurately represent the design space. This study shows how machine learning classifiers efficiently solve mechanical component design issues, particularly in exploring design spaces and finding optimal values.

STATEMENT OF THE AUTHOR

I hereby declare that this MSc thesis is my original work and has not been presented for a degree in any other university, and all sources of material used for this thesis / have been duly acknowledged.

Name: ……...……. Signature: …….……

Place: Institute of Technology, Hawassa University, Hawassa

Date of Submission: …….

Table of Contents

LIST OF FIGURES

LIST OF TABLES

Acronyms and Abbreviations

DOE	Design of Experiment
NN	Neural Network
SVM	Support Vector Machine
GNB	Gaussian Naïve Bayes
RF	Random Forest
BPD	Belt Pulley Drive
PVD	Pressure Vessel Design
HSD	Helical Spring Design
GPU	Graphical Processing Unit
BNN	Backpropagation Neural Network
BSD	Berkeley Software Distribution
ASME	American Society of Mechanical Engineers

CHAPTER ONE

INTRODUCTION

Engineers and designers need fast and efficient methods to explore alternative design solutions and find the optimal design by considering different factors to market a product quickly. These days market competition is high, so to outshine in the current market, a product should be optimized to have high quality, low price, compact size, light weight and high performance. To achieve these design requirements of products, robust data-driven design tools are essential.

Different methods have been developed to tackle design exploration and optimization problems in many fields [1-6]. The first approach takes function values by minimizing or maximizing an objective function by making derivatives equal to zero. Still, it does not always work because of the inflection point of some functions. The second approach that can solve problems with significant design variables uses function values and gradient information. These methods locate only local optima, have difficulty solving discrete optimization problems, and are complex to implement efficiently. The third method uses nature-inspired evolutionary algorithms capable of robustly solving various problems. These methods overcome the local optimum of other approaches, but their efficiency is problem-dependent. The future trend will be data-driven approaches, i.e., using data created either experimentally or using simulation, making a machine learn from these data to predict the new data in the future, which alleviates expensive experiments and simulation and helps to produce results within a short period.

Supervised machine learning classification methods are used to identify feasible design spaces for a given threshold value of a design problem [6]. This approach is inverse mapping, finding a set of design variables values that satisfy a performance requirement. It contrasts with the conventional method in engineering design, which is a forward mapping approach that uses a surrogate model that accepts unique design variable values and predicts the performance. Even though Surrogate models are often used to establish computationally efficient forward mappings, they have limited applicability to inverse mappings. Because in inverse mappings, a specific level of performance is associated with more than one candidate design and disjoint regions of the design space may provide similar performance levels.

In the design of mechanical components, a design space can be manipulated to get alternative design variable values to solve the design problem. However, specifying these design variable values is challenging when the problem has many design variables to produce alternative solutions or optimal design. Classification algorithms can help designers to improve the efficiency of design exploration and optimization tasks, especially in complex problems that need high simulations and experimentations, by using resources effectively.

Classification methods have been employed in reliability-based analysis and design, additive manufacturing, and machinery fault classifications [7-9]. For example, artificial neural networks, support vector machines, and Naïve Byes algorithms are applied in structural reliability analysis. These are complex and costly numerical problems to deal with the complexity and increase accuracy[10]. For additive manufacturing application, Bayesian network classifiers (BNC) has been used to design negative stiffness metamaterials to incorporate manufacturing variation into the design exploration process and identify designs that reliably meet performance requirements when this variation is considered [9].

From the literature, knowing that supervised classification methods have promising results in design space exploration and optimizations of engineering design problems, this study aimed to uncover some mechanical component design problems which are not solved by previous research by using this approach and provide more insight into the selection and application of supervised learning for design of mechanical components to the engineering design community.

1.1 Statement of the Problem

Design space exploration is the process of searching design variables, values, or parameters of a design problem that have the potential to satisfy requirements. It is a quantitative method of discovering which design variable values will impact the product's performance most. The most helpful method for design space exploration is performing design of experiments (DOE). In the DOE study, an analysis model is automatically evaluated multiple times by taking different values to cover several design variable values within the given range [11]. Since the number of design variable values is infinite, a statistical sampling method should be used to yield maximum information about the problem characteristics with the least experimental (computational) effort and with confidence that the set of points sampled gives a representative picture of the entire design space.

Among the discovered design variables values, there are some values that give the maximum performance. Hence, searching those values is achieved through the use of design optimization methods. Design optimization is a process of finding optimal values of design variable values that yield the maximum performance from the available design space. In general, an optimization problem consists of design variables, objective function and constraints [1]. Design space gives freedom to search combinations of design variable values, objective function specifies goals we wish to maximize or minimize, and constraints specify design limits to stay within. Depending on the type of objective functions, design variables and constraints, an optimization problem may have different characteristics that make the process more challenging to solve.

Instead of performing computationally expensive simulations or experiments to evaluate all the available design variable values, it is possible to evaluate some sample design points and use these points with their response to prepare datasets that have class labels based on threshold values and constraints and train a classification algorithm to classify the combinations of new design variables values whether they satisfy the performance requirements or not. The requirements can be mathematically formulated as follows:

$$Decide\ c = c_1\ if\ \ f(x) \leq f_{thresh};\ else\ decide\ c = c_2 \qquad (1.1)$$

where x is a candidate design, f is the performance function of interest, and c_1 and c_2 represent two classes of interest, satisfy and not satisfy, respectively.

Using probabilistic decision criteria for classification algorithms, performance requirements can be expressed as follows:

$$Decide\ c = c_1\ if\ \ \mathrm{p}(c_1|x) > \mathrm{p}(c_2|x);\ else\ decide\ c = c_2 \qquad (1.2)$$

where p(c|x) is the conditional probability of the class given the candidate design, x.

Data-driven method for design space exploration and optimization is not much investigated in the area of solving mechanical component design problems [13]. Design space exploration and optimization using traditional methods is not effective and efficient in addressing complex engineering design problems because it requires high computational resources and many iterations to get feasible design spaces and optimal values. Most of the time, in practical situations, the number of design variables is high and the influence of these variables to optimize the objective function will be complicated (nonconvex) with nonlinear character. To address such problems

machine learning is a promising approach since it learns from previous data to predict current situations and avoids expensive simulations.

1.2 Research Questions

Is machine learning, mainly supervised classification methods, a feasible approach in exploring design spaces and finding optimal values of design variables for mechanical components design problems, and if so, to what extent and degree of accuracy?

Specifically, our research questions are as follows:

1. Where and how can the required raw data be collected or generated for the study?
2. How to build reliable datasets for training and testing of the classification models?
3. How to develop supervised machine learning classification models suitable for solving mechanical component design problems.
4. Which models perform well for which problems, and what could be concluded from the comparative evaluation results?

1.3 Objectives

1.3.1 General Objective

The primary objective of the research is to develop machine learning classification models that can effectively enhance the design space exploration and optimization of mechanical component designs.

1.3.2 Specific Objectives

1. To perform design of experiments study for the three selected mechanical component design problems.
2. To build reliable datasets from the design of experiment study results for the selected problems.
3. To build a machine learning classification model for design space exploration and optimization.
4. To perform comparative evaluation with different dataset sizes and number of classes.

1.4 Scope of the Study

This research focuses on the design space exploration and optimization of selected mechanical component design problems. The research evaluated the performance of supervised machine learning models in searching feasible design space and optimal values of the optimization problem. Latin hypercube sampling, an efficient statistical method, has been employed to generate sample design points in the given interval to preparing reliable datasets using Simulation-based DoE methods. Small, medium, and large dataset sizes for binary and multiclass classification have been prepared to observe the effect of sample size on the performance of classification models. A cross-validation technique was employed to tune the hyperparameters of the classification models to improve classification performance.

1.5 Significance of the Study

This research will contribute for understanding the utilization of machine learning techniques in solving mechanical design problems. It may help designers and engineers to select an appropriate machine-learning algorithm for a specific problem that works more efficiently and effectively. It also provides designers insight into the amount of data required to get good accuracy depending on problem complexity and dimensionality. This knowledge aids in planning experiments in the early design stages. By applying appropriate machine learning models, designers can produce cost-effective and tailored designs within a short period.

1.6 Limitations of the Study

This study has the following limitations. First, the number of classification models we have tested is only four due to the time constraint. Second, we have studied the applicability of classifier models only on three mechanical component design problems because of time limitation. Covering more design problems with variety of characteristics can enhance to get more understanding on the performance of classification models. Third, the sampling method used to select design points from available design space is space filling which may not provide superior information for the classification models to small dataset sizes especially for imbalanced datasets. Finally, even if the datasets are collected from high-fidelity simulations using simulation-based design of experiments (DOE), the DOE does not consist of replication as the classical experiment-based DOE which includes replications. Replications in experiments may reduce variability and increase confidence

in the results. Simulation-based experiments use a probabilistic sampling method to select design points.

1.7 Structure of the Thesis

Chapter 1 provides an introduction and some background information. Chapter 2 reviews literature on design space exploration and optimization methods applied to mechanical component design problems. It also elaborates on the application of machine learning methods in engineering design and material science. In addition, sampling techniques that will be used in the research are discussed. Chapter 3 presents the methodology, starting from problem identification to model evaluation. It describes the mathematical formulations of mechanical component design problems, make them suitable for performing design of experiments. It also discusses dataset preparation, model fitting and model evaluation metrics. Moreover, it explains the theoretical background of the selected classification methods used for the research. Chapter 4 provides the findings of the research. It shows the performance of four machine learning classification models with different dataset sizes on three design problems with binary and multiclass classification. Chapter 5 concludes the thesis by summarizing the main points and suggesting future work for extension.

CHAPTER TWO

LITERATURE REVIEW

This chapter delves into the extensive literature surrounding engineering design space exploration and optimization issues. Over the years, various techniques have been employed to tackle these problems, comprising conventional and contemporary methods and machine learning tactics. These methodologies have progressed in tandem with the ever-growing intricacy of design, dating back to the dawn of the Industrial Revolution.

2.1 Overview of Design Space Exploration and Optimization Approaches

In the design decision-making processes, a correct understanding of the relationships between design variables values and product performance is essential to making cost-effective products. However, creating the relationship is difficult because modern design problems have many design variables and their interactions become more complicated and unclear. Design space exploration and optimization strategies give designers insight into the design process to obtain the desired results of design outputs.

Since experimental or empirical approaches are not efficient and costly, much research is performed on model-based approaches to improve design space exploration and optimization at different design stages [14]. Most approaches are used for embodiment design and detail design stages but rarely used in the conceptual design because of the following reasons: (1) The tasks in the conceptual design phase are to find engineering principles and solutions for high-level functional models rather than low-level physical models and (2) the design decision is made with enumerative approach rather than optimization and exploration approach.

Data-driven approaches, surrogate models, or metamodels are recently used approaches that offer a rapid means of design space exploration and optimization. The models substitute expensive high-fidelity simulation models by resembling input-output relations with training datasets. For engineering design, input–output behavior refers explicitly to the mapping between design variables values and their performance. The data-driven method encompasses three steps: dataset collection, training, model fitting, and model evaluation.

2.2 Traditional Methods

In the past, designers used experimental and empirical methods for design space exploration and optimization. They would rely on their own judgment, experience, analytical models, and opinions of others to make decisions in the hopes of achieving optimal design. However, this experience-based optimization may not identify the best design when dealing with many variables and conflicting objectives and/or constraints. The interactions are too complex and the variables too numerous to determine the optimum design intuitively.

When using analytical methods, the designers use a mathematical function containing one or more independent variables as an objective function that can be minimized or maximized depending on the problem. The concept of derivatives can be used to find an objective function's minimum or maximum values. The optimal point of a function is a point that makes the derivative function equal to zero but this does not always guarantee an optimum solution because there exists a saddle point or inflection point, which is neither a maximum nor a minimum point.

In traditional optimization methods, first, the algorithm decides the search direction and step length parameters. The algorithm finds the optimum solution iteratively, starting from a randomly chosen initial solution. Many traditional optimization methods can be categorized into direct search and gradient-based methods [1].

2.2.1 Direct Search Methods

Direct search methods, such as random search, univariate search, pattern search, etc., are optimization techniques that do not explicitly use derivatives. They work directly with values of the objective function to drive the search for an optimal value. Direct search methods were formally introduced in the late 1950s and early 1960s. These methods have remained popular due to their simplicity and practical success across many problems [15]. Recently, there has been renewed interest in direct search methods due to new mathematical analysis, their ability to work well with parallel and distributed computing, and their usefulness in solving optimization problems involving complex computer simulations.

Random search (RS) is an approach that can be used for functions that are not continuous or differentiable. The technique relies on randomly selecting a sequence of points within the feasible region of the problem using a predetermined probability distribution or a sequence of probability distributions. RS is useful for many problems and has been shown to achieve optimal performance under certain conditions. As a result, RS has gained popularity in the last decade for its ability to tackle complex problems that are challenging or impossible to analyze mathematically.

Uni-variate optimization is a non-linear optimization with no constraint and only one decision variable must be optimized. In machine learning, finding a model's coefficient to fit a training dataset is useful. It can also be used to find the value of a single hyperparameter that results in the best model performance.

Pattern search (PS) is a numerical optimization method that doesn't rely on the optimized problem's gradient. This makes PS useful for functions that are not continuous or differentiable. It's particularly helpful for low-dimensional optimization problems that are challenging or impossible to calculate derivatives [16]. PS is also referred to as direct-search, derivative-free, or black-box methods. It's a straightforward and effective optimization technique that can optimize a broad range of objective functions.

2.2.2 Gradient-Based Methods

Gradient-based optimization techniques use gradient information and function values to find the optimum solution efficiently. They are widely used for solving various optimization problems in engineering because they efficiently solve a problem with large numbers of design variables and require little problem-specific parameter tuning. However, these methods also have several drawbacks. They can only locate a local optimum, have difficulty solving discrete optimization problems, are complex algorithms that are difficult to implement efficiently, and may be susceptible to numerical noise [17]. The gradient-based algorithms differ primarily in the logic used to determine the search direction. Some of the more popular gradient-based methods are the steepest descent method, conjugate method, and quasi-newton method.

The Steepest Descent Method is an iterative method in which, at every iteration, the derivative is computed at the current point and a unidirectional search is performed in the negative to this derivative direction to find the minimum point along that direction. The search continues from the minimum point until a point with a small enough gradient vector is found. The conjugate gradient method is commonly used for solving large linear systems of equations and nonlinear optimization problems. The quasi-Newton method avoids high computational costs; it adapts the inverse of the Hessian matrix of the objective function to compute the minimizer, unlike the Newton method, where the inverse of the Hessian matrix is calculated at each iteration. The Quasi-Newton method computes the minimizer by adapting the inverse of the Hessian matrix of the objective function, avoiding the high computational costs of calculating the inverse at each iteration, unlike the Newton method.

2.2.3 Drawbacks of Traditional Methods

Traditional optimization tools have the following drawbacks[1]:

1. The final solution is dependent on the initially chosen random solution. There is no guarantee that the obtained solution will be globally optimal.
2. Optimization problems involving discontinuous objective functions cannot be tackled using gradient-based methods. Moreover, the solutions of gradient-based methods may get stuck at local optimum points.
3. There exist a variety of optimization problems. A particular traditional optimization method may be suitable for solving only one type of problem. Thus, no versatile optimization method can be used to solve various problems.

2.3 Non-Traditional Methods

These methods are developed to overcome the problem of the traditional methods, which need gradient information and are unable to solve nonlinear and non-differentiable objective functions. Even though the performance of these methods is problem-dependent, they can solve many problems, including linear, nonlinear, differentiable, and non-differentiable optimization problems [1], [18]. This section discusses the more popular methods, such as Genetic Algorithm, Particle Swarm Optimization, and Simulated Annealing methods.

2.3.1 Genetic Algorithm

Genetic algorithm is a nature-inspired method based on natural genetics and Darwin's principle of natural selection, which Holland and his collaborators first developed in the 1960s and 1970s [18]. It is a population-based search and optimization algorithm capable of solving various problems.

The first step is to create a random initial population. Typically, the population size stays the same during the optimization study. The population is then ranked based on the fitness (objective function) of each individual, and parents are randomly selected for reproduction. The parent designs are selected so that the higher-ranked (fitter) individuals are more likely to be selected. The next generation is created by randomly mixing selected parent designs in a process called cross-over. The new generation is again ranked, and the process is repeated until convergence [19].

2.3.2 Particle Swarm Optimization (PSO)

The PSO algorithm is a stochastic search method that draws inspiration from the social behavior of birds looking for food [20]. It iteratively updates generations to discover the optimal solutions. The algorithm is modeled after a swarm of bees searching for food, where each particle contributes to the overall information of the swarm. The process begins with an initial population randomly distributed throughout the design space, gradually converging into the best possible solution. The position of each particle is then updated from one design iteration to the next using the following update formula,

$$x_i^{q+1} = x_i^q + v_i^q \Delta t \qquad (2.1)$$

where i refers to the i^{th} individual in the swarm, q refers to the q^{th} iteration and v_i^q refers to the velocity vector of the i^{th} individual at the q^{th} iteration. The time increment Δt is typically taken as unity. Initially, each particle is assigned a random velocity vector that is updated at each iteration using

$$v_i^{q+1} = \omega v_i^q + c_1 r_1 \frac{(p_i - x_i^q)}{\Delta t} + c_2 r_2 \frac{(p^g - x_i^q)}{\Delta t} \qquad (2.2)$$

where ω is known as the inertia parameter, r_1 and r_2 are random numbers between 0 and 1, and c_1 and c_2 are known as trust parameters. Additionally, p_i is the best point found by the i^{th} particle, while p^g is the best point y by the swarm. The inertia parameter ω controls the search behavior of the algorithm, with larger values resulting in a more global search and smaller values resulting in a more local search. When utilizing particle swarm optimization, the trust of the particle by itself

and the group is reflected through the c_1 and c_2 parameters. It is recommended to set both parameters to 2. Moreover, the p^g parameter can be customized to local or global, depending on whether the best point is obtained from a small subset of particles or the entire swarm. The user must adjust the ω, c1 and c2 values to determine the number of particles in the swarm and the number of iterations to perform. Furthermore, there are five problem-dependent parameters that must be tailored by the user, even for the basic version of the algorithm. It is important to note that there are numerous variations of the algorithm that are more intricate. Many evolutionary methods necessitate specific parameter tuning tailored to the problem, which poses a significant drawback [21]. Like many other evolutionary algorithms, the PSO algorithm is an unconstrained optimization algorithm, which presents another major limitation.

2.3.3 Simulated Annealing

The simulated annealing algorithm mimics the gradual cooling of molten metal to find the minimum function value in a minimization problem [22]. A temperature-like parameter is controlled using Boltzmann probability distribution to simulate the cooling process. According to the Boltzmann probability distribution, a system in thermal equilibrium at a temperature T has its energy distributed probabilistically according to $P(E) = \exp\left(-\frac{E}{kT}\right)$, where k is the Boltzmann constant. When a system is at a high temperature, there is an equal chance of it being at any energy state. However, when the temperature is low, it is less likely to be at a high energy state. The algorithm's convergence can be managed by regulating the temperature (T) and using the Boltzmann probability distribution. The simulated annealing procedure is utilized to locate the global optimum even for ill-conditioned functions with many local minima, increasing the likelihood of success.

A general simulated annealing algorithm is as follows:

1. Choose an initial point x_o, a termination criterion ε. Set T a sufficiently high value, number of iterations to be performed at a particular temperature n, and set time t=0.
2. Calculate neighboring point $x^{(t-1)} = N(x^{t+1})$. Usually, a random point in the neighborhood is created.
3. If $\Delta E = E\left(x^{(t+1)}\right) - x^{(t)} < 0$, set t = t+1; Else create a random number (r) in the range (0,1).

4. If $r \leq exp\left(-\frac{E}{T}\right)$ set t = t+1; Else go to Step 2.
5. If $\left|x^{(t+1)} - x^{(t)}\right| < \varepsilon$ and T is small, Terminate; Else if (t mod n) = 0 then lower T according to the schedule. Go to Step 2; Else go to Step 2.

2.4 Surrogate Models

Surrogate models are a widely used tool in engineering, particularly when measuring or calculating desired outcomes is difficult [23]. By creating an estimated mathematical model, engineers can assess design objectives and constraints without requiring time-consuming simulations or experiments.

However, simulations remain a vital component in many engineering design problems and can be highly time-consuming, sometimes taking minutes, hours, or even days. This can present a significant challenge for tasks such as design optimization, sensitivity analysis, design space exploration, and what-if analysis, which require numerous simulation evaluations, often running into the millions.

A helpful strategy for reducing the workload is to create surrogate models that imitate the actions of a simulation model. These models, called emulators or metamodels, are less computationally intensive to assess. The construction of these models employs a bottom-up, data-driven technique where solely the input-output behavior of the simulation model is relevant, and the code's exact workings are irrelevant. The model's construction involves simulating the simulator's response to a limited number of intelligently chosen data points. This approach is also referred to as behavioral or black-box modeling. When there's only one design variable, it's called curve fitting.

Creating an accurate surrogate model with minimal simulation evaluations presents a scientific challenge. The process involves three major steps that can be iteratively interleaved. Firstly, there is sample selection, also known as sequential design, optimal experimental design, or active learning. The second step is constructing the surrogate model while optimizing its parameters, considering the bias-variance trade-off. Lastly, the accuracy of the surrogate is assessed.

The effectiveness of the surrogate relies heavily on the quantity and placement of samples, which may require costly experiments or simulations within the design space. Different design of experiments (DOE) techniques are available to address specific types of errors, such as those caused by data noise or an unsuitable surrogate model.

Several surrogate modeling methods are available, including polynomial response surfaces, kriging, radial basis function, support vector machines, space mapping, artificial neural networks, Bayesian networks, and random forests. However, which surrogate model will provide the most accurate results may not always be clear, especially when the true function is unknown in advance. Additionally, there is no widely accepted approach for determining the most reliable estimates of a surrogate's accuracy. For problems with known physical properties, physics-based surrogates such as space-mapping based models are generally the most efficient option [24].

Surrogate model-based optimization involves creating an initial surrogate using a portion of the available budget for expensive experiments and simulations. The remaining experiments and simulations are then conducted for designs that the surrogate model predicts will have favorable performance. This search and update process typically follows the following procedure.

1. Select the initial sample that serves as a starting point.
2. Create a surrogate model.
3. Thoroughly search the surrogate model using a genetic algorithm or other methods as it is cost-effective to do so.
4. Run and update experiments or simulations at newly discovered locations found through the search and add them to the sample set.
5. Repeat steps 2 to 4 until the design is deemed sufficiently good or time runs out.

The process of converging on an optimum solution, whether local or global or none, depends on the type of surrogate used and the complexity of the problem. When it comes to design space approximation, the focus is not on finding the perfect parameter vector, but rather on understanding the system's overall behavior. To achieve this, a surrogate is utilized to imitate the underlying model as accurately as possible across the entire design space. These surrogates provide an inexpensive and valuable means of gaining insight into the system's global behavior.

2.5 Data-Driven Approaches for Engineering Design and Material Science

The 1950s marked the beginning of computational science and simulations because of the advent of advanced computers. Computer experiments and simulations became possible, with the corresponding results being analyzed and interpreted like measured ones [2]. Data-driven approaches have been used in engineering design, additive manufacturing and material science as a means of prediction and classification mechanism.

The authors [6] presented a comparative study of four popular machine learning classification methods on six engineering design example problems that have different characteristics. In that study, 10,000 sample design points were generated for training using the Hammersley sequence for continuous variables and the randint function in SciPy for discrete variables. To differentiate the testing data from training data they have used the Halton sequence to generate testing data. Grid search with three-fold cross-validation was used to tune the hyperparameters of the algorithms. Threshold values are used for binary classification. The result of the study shows that the accuracy of tested machine-learning algorithms depends on the nature of the problem.

The authors [25] proposed a holistic approach that applies data-driven methods for design search and optimization in the embodiment and detail design stages of a design process for additive manufacturing. Bayesian network classifier was applied for design space exploration in the embodiment design phase and the Gaussian process regression model was used as an evaluation function for optimization to exploit design space in the detail design stages. The process of creating initial datasets involves the use of the Latin hypercube sampling method. These datasets are then further improved using the Markov Chain Monte Carlo sampling method. This two-step approach allows for a combination of design exploration and exploitation, as seen in the design of an ankle brace that includes customized horseshoe structures in specific areas to meet stiffness requirements during rehabilitation. The optimal designs resulting from this approach were evaluated using high-fidelity FEA simulations and outperformed the design identified through the standalone exploitation method.

A study was conducted to investigate the possibility of machine learning methods for the design of non-involute gears[8]. The data for training was collected from finite element simulations. Linear regression, K-nearest neighbor, Support Vector Machine, AdaBoost, neural network, and random forest models were used for numerical prediction. The models were validated with N-fold cross-validation and with new FEM simulations. The best-performing ones were Random Forest and AdaBoost. The proposed models calculate the nominal root stress in gears with a progressive curved path of contact. This model can provide real-time calculation of nominal root stress, serving as an alternative to FEM simulations. It is versatile enough to calculate stress for gears with varying numbers of teeth, widths, modules, paths of contact, materials, and loads.

The study [2] attempts to investigate the usefulness of machine learning methods in material science for the characterization of metallic material properties without performing expensive tests. The performance of manufactured components is affected by many parameters associated with the processing and the structure of the materials. The growth of material data through experiments and simulations aids in identifying structure-property relationships and discovering patterns across various length and time scales using data-driven methods. Conducting material mechanical property tests like tensile, compression, or creep tests can be expensive and time-consuming. However, utilizing machine learning techniques can simplify the process of generating material property information and help save time and cost. The study demonstrates ML methods on small punch test (SPT) data for the determination of the property ultimate tensile strength for various materials. A strong correlation was found between SPT and tensile test data, allowing for the replacement of more costly tests with simpler and faster ones in combination with ML.

The thesis work [26] applies machine learning methods to optimize sintered powder microstructure from two-phase field modeling. Sintering is a process of producing a component from powder by applying external agents such as heat, pressure, or temperature usually the process is performed below the melting point of the material. The problem with this manufacturing technology is the lack of persistent property of the manufactured components. The problem may arise from various sources of uncertainty present during manufacturing process. In this study the sources of uncertainty considered to be the two input parameters surface diffusivity and inter-particle distance. The size of the neck region developed between the two particles is selected as a response quantity. Two different cases with equal and unequal sized particles were studied. The machine-learning algorithm Gaussian Process Regression was used to create the surrogate model of the response quantity. Bayesian Optimization method was used to find optimal values of the input parameters. The results show that surface diffusivity should be higher and inter-particle distance should be lower for achieving larger neck size and better mechanical properties of the material.

2.6 Supervised Machine Learning Classification Techniques

There are two broad categories of classifiers: generative and discriminative [12]. Generative models are models where the focus is the distribution of individual classes in a dataset and the learning algorithms tend to model the underlying patterns/distribution of the data points. These models use the intuition of joint probability in theory, creating instances where a given feature (x)/input and the desired output/label (y) exist at the same time.

Generative models use probability estimates and likelihood to model data points and distinguish between different class labels in a dataset. These models are capable of generating new data instances. However, they also have a major drawback. The presence of outliers affects these models to a significant extent.

Examples of machine learning generative models

- Naive Bayes (and generally Bayesian networks)
- Hidden Markov model
- Linear discriminant analysis (LDA), a dimensionality reduction technique

Discriminative models, also called conditional models, tend to learn the boundary between classes/labels in a dataset. Unlike generative models, the goal here is to find the decision boundary separating one class from another.

Examples of machine learning discriminative models

- Support vector machines
- Random forest
- Logistic regression
- Decision trees
- Boosting (meta-algorithm)
- Conditional random fields

This section will discuss four supervised machine-learning classification algorithms used in this research. Supervised machine learning is a class of machine learning that uses prior labeled data to predict unlabeled new data. According to the authors [7] supervised machine learning has been applied in many focus areas such as healthcare, machinery fault diagnosis, sentiment analysis, text

classification, radar performance, etc. The models have been selected based on their suitability for specific tasks, dataset types, dataset size, training time, and performance.

2.6.1 Support Vector Machines

Support vector machines (SVM) are discriminative machine learning models that use a hyperplane or set of hyperplanes to separate data points into different classes. SVM is a mathematical approach to machine learning based on statistical learning theory. It uses a subset of the training input to construct a solution. SVM is commonly used for classification, regression, novelty detection, and feature reduction tasks [40]. Support Vector Machines have powerful capabilities, good at dealing with high dimensional data, and works well on small dataset, with the exception that compute and storage requirements rapidly increase with the number of training vectors.

SVM uses a hyperplane to separate data points into different classes with the possible largest margin. The margin is the distance between the hyperplane and the closest data points from each class. The class of new data can be specified by determining on which side of the hyperplane it falls. The hyperplane that can classify $n \times p$ data matrix X that consists of n-training observation in p-dimensional space for a class label of y_i is given by

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) > 0 \;\; for\ all\ i = 1, \ldots, n \qquad (2.3)$$

Where $y_i \in \{-1, 1\}$ which -1 represents one class and 1 represents the other class

$\beta_0, \ldots, \beta_p$ are the coefficients of the separating hyperplane

The classifier classifies the test observation x^* based on the sign of $f(x^*) = \beta_0 + \beta_1 x^*{}_1 + \beta_2 x^*{}_2, \ldots, \beta_p x^*{}_p$. If f(x^*) is positive, then it assigns the test observation to class 1, and if f(x^*) is negative, then it assigns to class −1.

Many hyperplanes can separate the training observations into two classes but obtaining the optimum hyperplanes that efficiently classify the data points with little error leads to an optimization problem expressed as follows.

Maximize M

$$\beta_0, \beta_1, \ldots, \beta_p, \epsilon_1, \ldots, \epsilon_n, M$$

Subjected to

$$\sum_{j=1}^{p} \beta_j^2 = 1, \qquad (2.4)$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) > M(1 - \epsilon_i), \qquad (2.5)$$

$$\epsilon_i \geq 0,\ \sum_i^n \epsilon_n \leq C, \qquad (2.6)$$

The constraint equation $y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) > M\ \ \forall\, i = 1, \ldots, n$ guarantees that each observation will be on the correct side of the hyperplane, provided that M the width of the margin is positive. For each observation to be on the correct side of the hyperplane, equation 2.4 should be satisfied. With $\sum_{j=1}^{p} \beta_j^2 = 1$ constraint the distance perpendicular to the ith observation and the hyperplane is given by

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \qquad (2.7)$$

In equation 2.5, $\epsilon_1, \ldots, \epsilon_n$ are slack variables that permit individual observations to be on the wrong side of the margin or the hyperplane. The slack variable ϵ_i indicates where the ith observation is located, relative to the hyperplane and margin. If $\epsilon_i > 0$ then the ith observation is on the wrong side of the margin, i.e., the ith observation has violated the margin. The data point will be on the wrong side of the hyperplane when $\epsilon_i > 1$.

C is a tuning parameter that bounds the sum of the ϵ_i's, and so it determines the number and severity of the violations to the margin or hyperplane that we will tolerate. C can be considered a budget for how much the n observations violate the margin. If C = 0, there is no budget for violations to the margin. For C > 0 , no more than C observations can be on the wrong side of the hyperplane. The reason is if an observation is on the wrong side of the hyperplane, then $\epsilon_i > 1$, and equation 2.6 requires that $\sum_i^n \epsilon_n \leq C$. As the budget C increases, we become more tolerant of violations to the margin, and so the margin will widen. Conversely, as C decreases, it becomes less tolerant of violations to the margin, and so the margin narrows [41].

In practice, C is treated as a tuning parameter that is generally chosen via cross-validation. C controls the bias-variance trade-off of the statistical learning technique. When C is small, we seek narrow margins that are rarely violated; this amounts to a classifier that is highly fit to the data, which may have low bias but high variance. On the other hand, when C is larger, the margin is wider, and we allow more violations to it; this amounts to fitting the data less challenging and obtaining a potentially more biased classifier that may have lower variance.

The approach for linearly non-separable data is to transform it into a higher dimensional feature space where it can be separable. A decision boundary can be applied to separate the classes and make predictions. To optimize the objective function and train a support vector classifier, it is

necessary to perform operations with higher dimensional vectors in the transformed feature space. However, in practice, there might be many features in the data that may make it challenging to apply a transformation involving many polynomial combinations of these features, leading to extremely high and impractical computational costs. To overcome such a problem, the kernel trick is used that represents data through pairwise similarity comparisons between original data observations rather than applying transformation and representing the data by these transformed coordinates in a higher dimensional feature space. It transforms linearly inseparable data into a separable one by mapping it into a higher dimension using a kernel function.

There are two approaches for multiclass classification using SVM: one-to-one approach and one-to-rest. The one-to-one approach breaks the multiclass problem into multiple binary classification problems. The idea is to map data points to high dimensional space to gain mutual linear separation between every two classes. The classifier can use a $\frac{m(m-1)}{2}$ SVMs where m is the number of classes. As shown in Figure 4 a), the hyperplane to separate between every two classes neglects the points of the third class. This means the separation takes into account only the points of the two classes in the current split. For example, the red-blue line tries to maximize the separation only between blue and red points. It has nothing to do with green points.

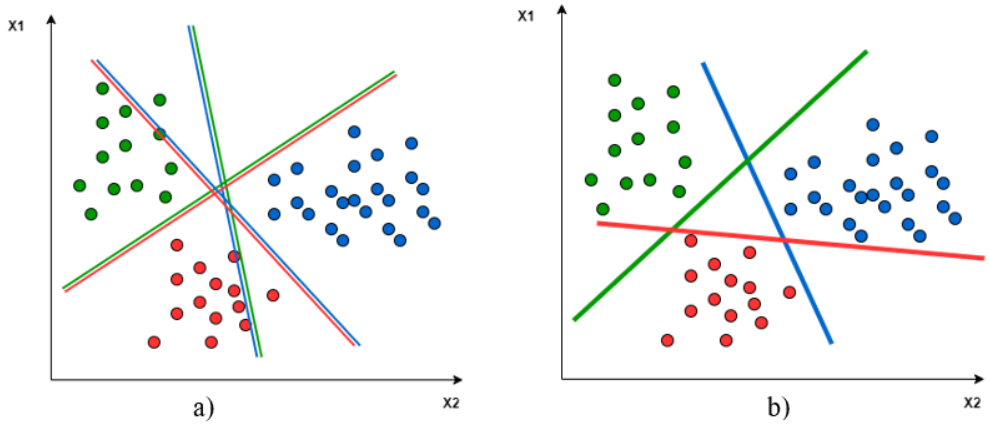


Figure 1 Multiclass classification: a) one to one and b) one to rest

In the One-to-Rest approach, the classifier can use m SVMs. Each SVM would predict membership in one of the m classes. Figure 4 b) shows a hyperplane to separate between a class and all others at once. The separation takes all points into account, dividing them into two groups; a group for the class points and a group for all other points. For example, the green line tries to maximize the separation between green points and all other points at once.

2.6.2 Random Forest

The algorithm creates decision trees on data samples and then gets the prediction for each try until it selects the best solutions [42]. It is an advanced version of decision trees because it reduces the overfitting cons of decision trees by averaging the result. Random forests have some appealing properties, such as they naturally handle both regression and multiclass classification, are relatively fast to train and predict, depend only on one or two tuning parameters, and can be used directly for high-dimensional problems. The working principle of random forests is shown in Figure 5.

The training algorithm applies the general technique of bootstrap aggregating or bagging[41]. Given a training set X = $\{x_1, \ldots, x_n\}$ with responses Y= $\{y_1, \ldots, y_n\}$, bagging selects a random sample with replacement of the training set repeatedly for B times and fits decision trees to these samples. After training, the class prediction for an unseen sample, x′, is determined by taking a majority vote as follows:

$$C(x') = \arg\max_i \sum_{j=1}^{B} w_j I(h_j(x') = i) \qquad (2.8)$$

where $w_1, \ldots, w_B$ are weights that sum to 1, which are usually set to 1/B; I(·) is an indicator function; and $h_j(\cdot)$ is the prediction function for the j^{th} decision tree. In general, employing an RF classifier for classification includes four steps: (1) select random samples from a given dataset, (2) construct a decision tree for each sample and obtain a prediction result from each decision tree, (3) take a vote from each decision tree, and (4) select the prediction with the most votes as the final prediction of class membership. The RF technique is generally very effective in preventing overfitting, even when the ensemble contains thousands of individual trees. The error rate of RF on unseen samples tends to converge slowly to a limiting value when the number of trees grows rapidly.

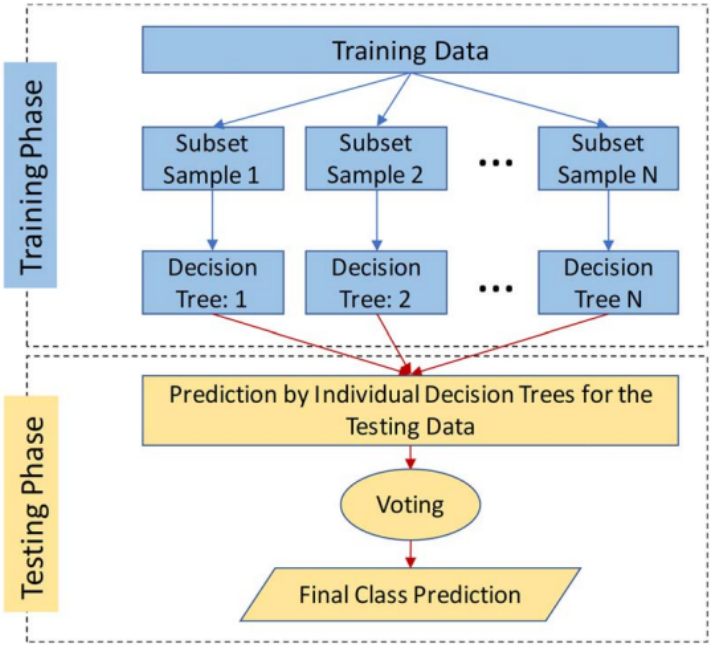


Figure 2 The working principle of the random forest classifier taken from [6]

2.6.3 Gaussian Naïve Bayes

The Bayesian classification represents a supervised learning method as well as a statistical classification method. It assumes an underlying probabilistic model and it allows us to capture uncertainty about the model in a principled way by determining probabilities of the outcomes [43]. In Bayesian classification prior knowledge and observed data can be combined to predict the outcomes.

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There are several algorithms based on a common principle that assumes that the value of a particular feature is independent of the value of any other feature, given the class variable [44]. Naive Bayes is suitable for solving multi-class prediction models. It is quick and easy to save much time and handle complex data.

The Naïve Byes classifier can be formulated as follows using Bayes' theorem:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)} \quad (2.9)$$

Where X = {x_1, x_2, …, x_n} are feature vectors and Y = {y_1, y_2, …, y_n} are class variable.

By putting naïve assumption to Byes theorem, we can split pieces of evidence as independent parts which give,

$$P(y|x_1, \dots, x_n) = \frac{P(x_1|y)P(x_2|y)\dots P(x_n|y)P(y)}{P(x_1)P(x_2)\dots P(x_n)} \quad (2.10)$$

This can also be expressed as,

$$P(y|x_1, \dots, x_n) = \frac{P(y)\pi_{i=1}^{n} P(x_i|y)}{P(x_1)P(x_2)\dots P(x_n)} \quad (2.11)$$

As the denominator remains constant for a given input, we can remove that term:

$$P(y|x_1, \dots, x_n) \propto P(y)\pi_{i=1}^{n} P(x_i|y) \quad (2.12)$$

To create a classifier model, we find the probability of a given set of inputs for all possible values of the class variable y and pick up the output with maximum probability. This can be expressed mathematically as:

$$y = argmax_y P(y)\pi_{i=1}^{n} P(x_i|y) \quad (2.13)$$

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution. A Gaussian distribution is also called a normal distribution. The likelihood of the features is assumed to be Gaussian; hence, conditional probability is given by:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right) \quad (2.14)$$

Where μ_y is the mean of the class variable and σ_y is the variance of the class variable distribution.

2.6.4 Artificial Neural Networks

Artificial neural networks are a widely used method for supervised learning in classification tasks. This technique employs a network structure that comprises an input layer, an output layer, and typically one or more hidden layers. The input layer's dimensions are determined by the problem's input features, while the output layer's size is determined by the number of distinct output classes. The complexity of the problem is reflected in the size and number of hidden layers. Figure 6 provides a three-layer fully connected feed-forward neural network model. The labels show the number of nodes in each layer and the line with arrow heads indicates the weighted connections between nodes of different layers. The output from the network's k^{th} output node can be expressed as

$$y_k = \varphi_0\{\alpha_k + \sum_j w_{jk}\, \varphi_h(\alpha_j + \sum_j w_{ij}\, x_i)\} \qquad (2.15)$$

In a neural network, the weights and biases are represented by w and α respectively. The activation function is denoted by φ. The activation function can be a linear mapping, like the identity function which returns the input value, or a nonlinear mapping like rectified linear units (ReLUs) and logistic functions that can model more complex mappings [45]. When training a neural network, we utilize training samples and an algorithm to determine the weights and biases. Backpropagation neural networks (BNN) are a specific type of artificial neural network that uses backpropagation for supervised learning during network training [46]. During BNN training, inputs are processed through hidden layers to the output layer, while errors are sent back to the input layer. By adjusting synaptic weights and biases, we can reduce errors. To classify data using BNN, we feed new data points into the trained BNN model, and the outputs of the trained BNN provide the conditional probabilities of class membership. We use the decision criterion in Eq. (1.2) to interpret these probabilities and determine the predicted binary class label.

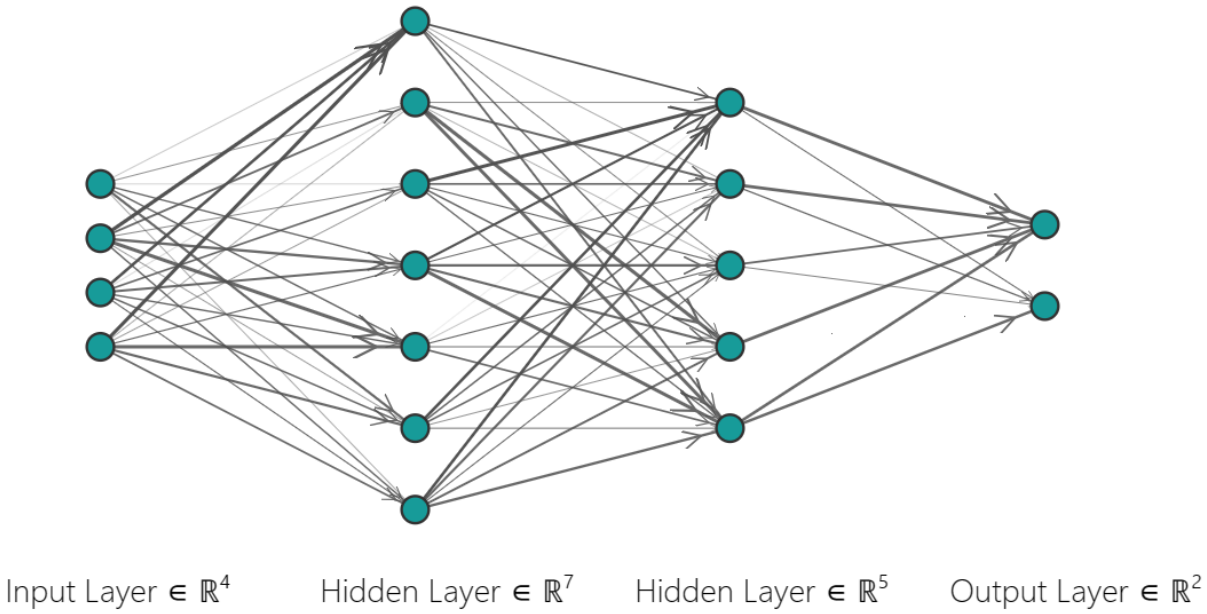


Figure 3 A fully connected three-layer neural network model.

There are multiple ways to structure neural networks. For the sake of simplicity and versatility, this work utilizes a fully connected feed-forward neural network model called the multilayer perceptron, which is depicted in Figure 6. However, convolutional neural networks have become increasingly popular for working with spatially structured data, like images or topologies [47]. As the focus of this study is not on spatially structured data, these alternate network architectures are not implemented here.

When neural networks have a high number of hidden layers, they are referred to as deep learning. In recent years, deep learning has become increasingly popular due to advancements in computing power, like massively parallel GPU architectures, and custom software such as TensorFlow [48]. These developments have allowed deep neural networks to process large datasets with remarkable results in the machine learning and design communities. However, training deep learning networks requires access to vast amounts of training data to fit thousands of associated network parameters. This study, on the other hand, is concerned with training classifiers using only a few hundred to a few thousand samples, which is insufficient for training deep learning networks, as is often the case in engineering design applications. Consequently, this research focuses on neural networks with only one or two hidden layers.

2.6.5 Cost of Computation

When training classifiers, it is crucial to consider the costs of computation. The time it takes to train a classifier increases as the number of training points and design variables grows, and the amount of time varies depending on the algorithm used. In this work, we've summarized the worst-case theoretical time complexity of different algorithms in Table 1. The variables used in the table include the number of training points (n), the number of variables (m), the number of trees in a random forest (t), the number of variables randomly sampled at each node of the random forest decision trees (p), the number of neurons in each layer of a neural network (h), the number of layers in a neural network (k), and the number of backpropagation iterations (i). Out of the algorithms we have considered, Gaussian Naive Bayes (GNB) is the most efficient for training due to its linear time scaling. However, random forests and support vector machines have exponential scaling with the number of training points, making them computationally expensive for larger datasets. For fully connected neural networks, the efficiency depends heavily on the number of layers and neurons present in the network. While some modern networks have many parameters, resulting in greater computational expense, it's important to note that the computational complexity in Table 1 only refers to the cost of training the classifiers. In comparison, all of these classifiers require less computational expense for prediction.

Table 1 Computational complexity of training classifier

Algorithm	***Computational complexity***
Gaussian naive Bayes	$O(nm)$
SVM	$O(n^2m)$ to $O(n^3m)$
NN (perceptron)	$O(nmh^ki)$
Random forest	$O(pn^2t\log(n))$

2.7 Sampling Techniques for the Design of Experiments

The selection strategy of the design point plays a crucial role in determining the accuracy of models. There are many sampling methods for simulation-based DOE which have their own characteristics in covering the given sample space and taking sampled points. The method used for selection depends on the specific problem at hand. This section presents different sampling methods that should be applicable to this study.

2.7.1 Grid Search

Grid search is also called a grid or full factorial sampling. A method for searching a space involves placing a grid of evenly spaced points over it. This method is simple to implement and covers the entire space without relying on randomness. However, it uses a significant number of points and the grid's coarseness may skip over regions of the space where good solutions exist[27]. This problem becomes more severe as the number of inputs or search space dimensions increases.

To generate a grid of samples, one can choose a uniform separation of points and enumerate each variable in turn while incrementing it by the chosen separation. In machine learning models, adjustable parameters can alter how the model learns. One approach is to experiment with different values and select the one that yields the best score, which is called a grid search. If we have to choose values for two or more parameters, we'll evaluate all combinations of the value sets, resulting in a grid of values.

A full factorial experiment involves multiple factors, each with discrete levels, and all possible combinations of these levels are tested on the experimental units[28]. This design is also known as a fully crossed design and allows researchers to analyze the impact of each factor and the interactions between them on the response variable. A factorial design enables the determination of the impact of multiple factors and their interactions with the same number of trials needed to determine a single effect accurately.

2.7.2 Random Search

Random sampling draws m random samples over the design space using a pseudorandom number generator. To generate a random sample x, we can sample each variable independently from a distribution. A random search may be the best strategy when dealing with highly complex problems with discontinuous areas in the search space that can affect gradient-dependent algorithms.

To acquire a sample from a specific domain, we can rely on a pseudorandom number generator. We need to establish the range or limit of each variable and then extract a value from that range uniformly at random. This procedure is computationally simple and does not require significant memory. It can be advantageous to create a sizable input sample and then assess each one separately. Because each sample is distinct, it can be evaluated in parallel to expedite the process

if needed. With random sampling, new sample points are generated without regard to previously generated ones. The predetermined number of sample points is not a requirement.

2.7.3 Latin Hypercube Sampling

Latin Hypercube Sampling (LHS) generates a nearly random selection of parameter values from a multidimensional distribution, making it an efficient statistical technique. When there is just one sample in each row and column, this arrangement of sample positions is referred to as a Latin square [29]. However, a Latin hypercube is a more intricate extension of this concept, allowing for an arbitrary number of dimensions. In a Latin hypercube, each sample is the sole representative in every axis-aligned hyperplane that encompasses it. This technique is widely employed in constructing computer experiments.

To sample a function with N variables, we divide the range of each variable into M equally probable intervals and place M sample points to meet the Latin hypercube requirements. This ensures that the number of divisions, M, is equal for each variable and does not require more samples for more dimensions. One advantage of this sampling scheme is its independence, allowing random samples to be taken one at a time while keeping track of the samples already taken. For Latin hypercube sampling, we must first determine the number of sample points and remember the row and column for each sample point.

When dealing with a function that has multiple variables, we can use the Latin hypercube sampling technique to obtain representative samples. Essentially, we divide the range of each variable into a fixed number of equally probable intervals (M), and then place M sample points in a way that satisfies certain requirements. One of the key benefits of this technique is that the number of intervals remains constant across all variables, which ensures statistical independence. Additionally, we can take random samples one at a time and keep track of which ones we've already taken by noting their respective rows and columns in the sampling grid[30]. To use this method, we must first determine the number of sample points.

2.7.4 Quasi-Monte Carlo Sampling

Monte Carlo methods, or Monte Carlo experiments, are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The concept of randomness is utilized to solve problems that may be deterministic in theory [31]. They are commonly applied in physical and mathematical scenarios and are most effective when other approaches are not feasible

or impossible to use. Monte Carlo methods are mainly used in three problem classes: optimization, numerical integration, and generating draws from a probability distribution.

Quasi-Monte Carlo (QMC) methods provide an $n \times d$ array of numbers in $[0,1]$. They can be used in place of n points from the $U[0,1]^d$ distribution. Compared to random points, QMC points are designed to have fewer gaps and clumps which can be quantified by discrepancy measures.

QMC constructions are often optimized for specific values of n, such as powers of 2 or large primes. Even a small change in the sample size can significantly impact their performance and convergence rate [32]. Certain QMC constructions can be extended in terms of the sample size (n). We can discover another special base for sample size (n'), which is a prime number greater than n, and often an infinite sequence of increasing special sample sizes. Additionally, some QMC constructions can be extended in terms of the dimension (d) without requiring special values of d, possibly up to a certain upper bound [33]. Finally, there are QMC methods that can be extended in both a and d.

2.7.5 Halton Sampling

Halton sampling is a type of sampling method that reduces discrepancies. It generates a set of sample points that are well-distributed but not uniform. The samples are not too close together and there are no excessively large regions of the sample space with no samples [34]. It follows a deterministic process that uses the Halton sequence, a series of numbers created by a set of co-prime bases. This sequence is an extension of the Van der Corput sequence, and each individual Halton sequence is based on an inverse function defined on a prime number. However, it's important to note that the Halton sampling method is only suitable for low-dimensional problems where n is less than or equal to 10. As the dimensions get higher, the effectiveness of the sampling method decreases.

2.7.6 Hammersley Sampling

Hammersley Sampling generates a uniformly distributed and stochastic-looking sampling pattern using a deterministic method, at low computational cost [35]. Hammersley samples are produced using a method like Halton samples. This method involves reversing/flipping the base conversion of numbers using primes. To generate n samples in a p-dimensional space, the first (p-1) prime numbers are utilized. The first dimension is created by uniformly dividing the region into a certain number of sample points.

The Hammersley set and the Halton sequence is similar, except that the former incorporates a regular grid in one dimension. Generating a Hammersley sequence with more than ten dimensions is not recommended. The user can select samples from their own dataset or from provided bounds.

CHAPTER THREE

MATERIALS AND METHODS

This chapter describes creating a highly effective classification model that can accurately predict the class membership of proposed design problems based on their variable values. The use of supervised machine learning techniques has shown to be effective in engineering design exploration and optimization applications. These techniques help identify potential or viable areas within the design space [6].

The study employed four machine learning models to address three mechanical component design issues. The implemented models are support vector machines, random forests, Gaussian naïve byes, and neural networks (perceptron). Exploring the use of these algorithms to solve mechanical design problems will provide mechanical designers and engineers with valuable knowledge of machine learning models, allowing them to use these algorithms to resolve similar issues.

The research is a quantitative type and involves an experimental design to tackle the presented issues. The design problems chosen for this study are available in the literature and suggested for design space exploration and optimization through evolutionary algorithms, surrogate models, and other methods [5]. These problems have been selected to assess the effectiveness of the proposed method. Various problem characteristics have been considered while selecting the issues, such as problem dimensionality, number of constraints, variable type, and objective type.

2.8 Data Collection Techniques

Primary data have been collected from high-fidelity simulations for each design problem to prepare a reliable dataset. Using the design of experiments, data were generated for design variables within a given range. The generated design variables are input to determine the responses to the design problem. The feasibility of the response can be classified using conditional probability with a specified threshold value.

Data can be collected and analyzed using design of experiments (DOE) as a tool in various experimental situations. It is helpful for planning, conducting, analyzing, and interpreting controlled tests to evaluate the factors that influence the value of a parameter or group of parameters. DOE permits the manipulation of multiple input factors at a time to identify their effect on the response variable [36]. It is possible to find important interactions among the variables that

might be missed when experimenting with one factor at a time. All possible combinations can be investigated (full factorial) or only a portion of the possible combinations (fractional factorial).

Passive data collection leads to several problems in statistical modeling. Observed changes in a response variable may be correlated with, but not caused by, observed changes in individual factors (process variables). Simultaneous changes in multiple factors may produce complex interactions to separate into individual effects. Observations may be dependent, while a model of the data considers them to be independent. Designed experiments address these problems. In a designed experiment, the data-producing process is actively manipulated to improve information quality and eliminate redundant data. A common goal of all experimental designs is to collect data as parsimoniously as possible while providing sufficient information to estimate model parameters accurately.

2.9 Mathematical Formulations of Mechanical Component Design Problems

Mechanical components play a significant role in building simple to complex machines to perform many tasks. They are commonly found in many real-world applications designed with different sizes and shapes to satisfy some requirements. Engineering design and analysis of a machine as a whole is a complex problem to solve as it is because it leads to objective function with a large number of design variables and complex constraints. The usual approach is to divide a complex problem into small problems and find a solution for each simpler problem [37]. Since mechanical systems comprise mechanical components or subassemblies, performing design space exploration and optimization on the component or subassembly level will help the designer solve the problem efficiently and get an optimized final product.

While designing machine elements, design space exploration and optimization help in several ways to reduce material cost, ensure better service of components, increase production rate, and many other parameters. For this research, three different mechanical component design problems, namely, helical spring, belt-pulley drive, and pressure vessel are proposed to demonstrate the performance of supervised machine learning classification algorithms. These mechanical components have many applications in critical systems in industries and their design need more precision and safety.

3.1.1 Pressure Vessel Design Problem

Pressure vessels are containers designed to store fluids at a pressure considerably different from the ambient pressure. The critical subassembly of the vessel is the assembly of the shell and end enclosure which holds the fluid securely. The vessel should withstand the internal pressure produced by the fluid because of chemical reactions or heat as in boilers. Pressure vessels have many industrial applications and should be designed optimally to avoid failures; vessel failure will result in catastrophic damage to humans and properties in the industrial plant.

Let us consider a specific scenario described here to demonstrate the application of machine learning classification methods. Design a pressure vessel to store compressed air with a working pressure of 1000psi and a minimum volume of 750 ft^3. The schematic of a pressure vessel is shown in Figure 1. The cylindrical Pressure vessel is capped at both ends by hemispherical heads. Using a rolled steel plate (SAEJ 2340 TYPE 830R), the shell will be made in two halves joined by two longitudinal welds to form a cylinder. Each head is forged and then welded to the shell. The design variables are the shell radius (x_1), shell length (x_2), shell thickness (x_3), and head thickness (x_4).

The size and cost of the vessel can be determined with different combinations of these four design variables. The different combinations of these variables give us the design space, i.e., by taking different values of these variables, we can produce different vessels, but the size and cost may not be acceptable for all combinations of the design variables. Therefore, combining design variables should satisfy the objective function and obey the imposed constraints.

In this specific problem, the objective is to minimize the manufacturing cost of the pressure vessel by minimizing weight. The manufacturing cost of the pressure vessel is a combination of material cost, welding cost and forming cost. The objective function is described in terms of the design variables, as shown in Equation 1.

$$f(X) = 0.6224x_1x_2x_3 + 1.7781x_1^2x_3 + 3.1661x_2x_4^2 + 19.84x_4x_1^2 \qquad (3.1)$$

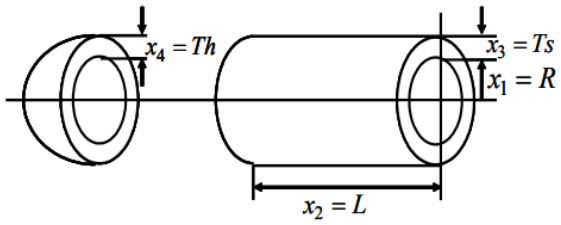


Figure 4 Schematic representation of pressure vessel

Circumferential or hoop stress, longitudinal stress, and volume are design parameters that impose constraints. The stress values in the vessel should be less than the strength of the vessel material to a certain extent to avoid any failure resulting from excessive stresses. Respective ASME code standards set the constraints. They are the two stresses induced in the vessel in response to the internal pressure, the volume, and the length of the vessel, described in equations 3.2-3.5.

1. Hoop stress <= Allowable stress

$$g_1(x) = 0.0193x_1 - x_4 <= 0 \quad (3.2)$$

2. Longitudinal stress <= Allowable stress

$$g_2(x) = 0.00954x_1 - x_3 \leq 0 \quad (3.3)$$

3. Volume = 750*1728 inch^3

$$g_3(x) = 750 * 1728 - \frac{4}{3}\pi x_1^3 - \pi x_1^2 x_2 \leq 0 \quad (3.4)$$

4. Length

$$g_4(x) = x_2 - 240 <= 0 \quad (3.5)$$

The upper and lower bounds of design variables are shown in Table 2. The range of variables is specified based on requirements. Hence, the goal is to generate the design variable values that give minimum cost by satisfying the constraints. Input data used to determine the objective function and design constraints are depicted in Table 3. The material considered in the design of the pressure vessel is SAE J2340 - 830R, a high-strength recovery annealed steel.

Table 2 upper and lower bounds of the design variables used in the design of pressure vessel

Design variables	**Symbols**	**Sample range (inch)**
Shell radius	x_1	25 to 150
Shell length	x_2	25 to 240
Shell thickness	x_3	0.0625 to 1.25
Head thickness	x_4	0.0625 to 1.25

Table 3 Constants used for determination of the design variables

Design Constants	**Symbol**	**Fixed values**
Modulus of elasticity	E	$200x10^9$ MPa
Yield strength	S_y	960 MPa
Factor of safety	n	1.78
Allowable yield strength	S_{all}	540 MPa
Applied pressure	p	6.80272 MPa (1000PSi)

3.1.2 Helical Spring Design Problem

A helical spring is a machine element that stores or absorbs mechanical energy. It is made up of a wire coiled as a helix primarily intended for applying compressive or tensile loads to the object (Figure 2). The wire which is used to make the spring may have circular, square, or rectangular cross-sections. There are two forms of helical springs, called compressive helical springs and tensile helical springs. Compressive spring is used to apply outward force on the object, whereas tensile spring creates an inward force to pull objects. Helical coiled springs have many applications, such as vehicle suspensions, mattresses, plumbing, engine starters, hinges, consumer goods, medical devices, aerospace, construction, tools, machinery, pens, keyboards, pogo sticks, gym equipment, and shock absorbers.

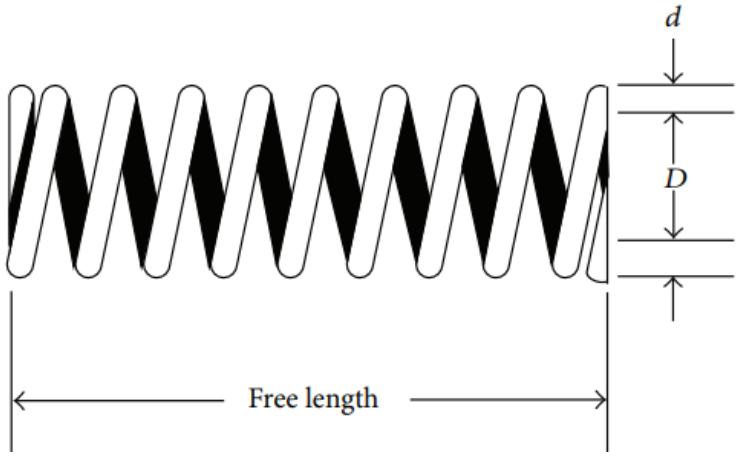


Figure 5 Schematic representation of closed helical spring

The closed coil helical spring design problem considered here has a single objective and eight constraints. As the constraints increase, the design space becomes narrower, which makes the problem more challenging to determine the feasible design variable values. To overcome such

challenges, more controlled data points are required during sampling design. Coil diameter, wire diameter, and number of coils shown in Table 4 are design variables used to specify the helical spring size. The number of coils is integer values and the others are continuous values. All the loads and stresses imposed on the spring depend on these variables. Therefore, optimizing these parameters is important to obtain the robust and compact size of a spring. In this case, the objective is to minimize the volume of the coiled helical spring by satisfying several constraints.

Table 4 Coiled helical spring problem design variables

Design variables	**Symbols**	**Sample range**
Wire diameter	d	0.508 to 1.016
Coil diameter	D	1.27 to 7.62
Number of coils	N_c	15 to 25

The stress constraint is due to the external load applied on the spring parallel to the axis, which creates shear stress on the spring because of the twisting effect. The shear stress induced in the wire should be less than the allowable shear stress of the material. The configuration constraints are imposed to prevent excessive deformation of the spring. All constraints are shown in Equations 3.7 to 3.14, and the objective function's performance threshold is shown in Equation 3.6. Equations for calculated quantities and constants used in the calculation are shown in Table 5 and Table 6.

Performance threshold: $V \leq 50\ cm^3, V \leq 70cm^3\ and\ V > 70cm^3$ (3.6)

Constraints: $S - S_s \geq 0$ (3.7)

$$l_{max} = l_f \geq 0 \quad (3.8)$$

$$d - d_{min} \geq 0 \quad (3.9)$$

$$D_{max} - (D + d) \geq 0 \quad (3.10)$$

$$C - 3 \geq 0 \quad (3.11)$$

$$\delta_{pm} - \delta_p \geq 0 \quad (3.12)$$

$$l_f - \delta_p \geq 0 \quad (3.13)$$

$$\frac{F_{max} - F_p}{K} \geq 0 \quad (3.14)$$

Table 5 Coiled helical spring calculated quantities using design variables

Calculated quantity	Equations
Volume of spring	$V = \frac{\pi^2}{4}(N_c + 2)Dd^2$
Spring index	$C = \frac{D}{d}$
Stress factor	$C_f = \frac{4C - 1}{4C - 4} + \frac{0.615}{C}$
Shear stress	$S_s = 8C_f F_{max} \frac{D}{\pi d^3}$
Spring constant	$K = \frac{Gd^4}{8N_c D^3}$
Deflection under maximum working load	$\delta_l = \frac{F_{max}}{K}$
Free length of the spring	$l_f = \delta_l + 1.05(N_c + 2)d$
Deflection under preload	$\delta_p = \frac{F_p}{K}$

Table 6 Coiled helical spring problem constants used for calculations

Design constants	Symbol	Fixed values
Maximum load	F_{max}	453.6 kg
Shear modulus	G	808543.6 kg/cm^2
Maximum spring length	l_{max}	35.56 cm
Minimum wire diameter	d_{min}	0.508 cm
Maximum coil diameter	D_{max}	7.62cm
Preload	F_p	136.08 kg
Maximum deflection by preload		15.24 cm
The allowable range of deflection		3.175

3.1.3 Belt Pulley Drive Problem

A method of transmitting power from one shaft to another is through pulleys that rotate at the same or different speeds (as seen in Figure 3). Factories and workshops commonly use stepped flat belt drives to transmit moderate power. It is important to note that the weight of the pulley can cause stress on the shaft and bearing, which can lead to failure. To avoid such failure, minimizing the weight of the flat belt drive is necessary. For a clearer idea of how this works, refer to the schematic representation of a belt-pulley drive in Figure 3.

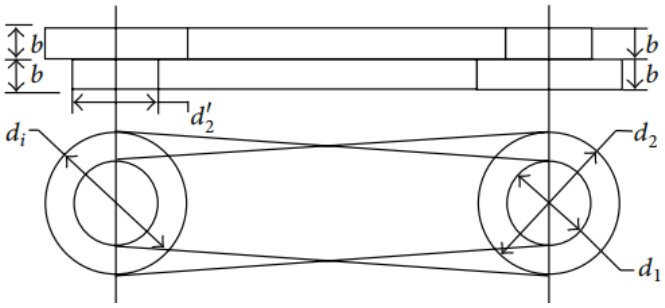


Figure 6 Schematic representation of belt pulley drive

This problem aims to minimize the weight of the pulleys so that the effect of stress induced in the support and on the shaft will be reduced. The design variables are shown in Table 7 and the performance threshold is shown in equation 3.15. Constraints of this problem are shown in equations 3.16 and 3.17. Equations used to determine the objective function and design constraints are shown in Table 8. The constants used in the calculation are shown in Table 9.

Performance threshold: $W_p \leq 40, W_p \leq 60 \; and \; W_p > 60$ (3.15)

Constraints: $bd_2 - 81.97 \geq 0$ *(3.16)*

$d_1 - 4b \geq 0$ *(3.17)*

Table 7 Design variables for belt pulley drive problem

Design variables	**Symbols**	**Sample range**
Diameter of the first pulley	d_1	15 to 25 cm
Diameter of the second pulley	d_2	70 to 80 cm
Width of the pulley	b	4 to 10 cm

Table 8 Constant used for determination of the design variables in the belt pulley drive design problem

Design constants	Symbol	Fixed values
Power	P	10hp
Density of belt material	ρ	7.2×10^{-3}
Allowable tensile stress of belt material	σ	$30\ kg/cm^2$
Belt thickness	t_b	1cm
Diameter of the third pulley	d'_1	$2d_1$
Diameter of the fourth pulley	d'_2	$0.5d_2$
Thickness of first pulley	t_1	$0.1d_1$
Thickness of the second pulley	t_2	$0.1d_2$
Thickness of third pulley	t_3	$0.2d_1$
Thickness of the fourth pulley	t_4	$0.05d_2$
The ratio of slack side tension and tight side tension	$\frac{T_2}{T_1}$	0.5
Angular speed of the first pulley	N_1	1000 rpm
Angular speed of the second pulley	N_2	250 rpm
Angular speed of the third pulley	N_3	500 rpm
Angular speed of the fourth pulley	N_4	500 rpm

Table 9 Equations of calculated quantities for belt pulley drive problem

Calculated quantity	Equations
Weight of pulleys	$W_p = \pi\rho b[d_1t_1 + d_2t_2 + d'_1t'_1 + d'_2t'_2]$ $= 0.113047bd_1^2 + 0.0028274bd_2^2$
Velocity of the belt	$V = \pi d_1 N_1$
Tight side tension	$T_1 = \frac{150P}{V}$

3.2 Synthetic Data (Sample) Generation

Data can be obtained in two ways: from records or measurements and synthetically generated. Synthetic data can be created for reasons such as to protect privacy, faster turnaround for product testing, new product design, and training machine learning algorithms [38]. Generating real data by experimenting with engineering design problems is costly and time-consuming. Therefore, the feasible data generation method to use machine learning to solve design problems is the simulation and design of experiments techniques. For performing the design of the experiment, it is possible to use sampling methods to select appropriate data points which are useful to get desired outputs.

For engineering design problems synthetically, generated data are preferable since there is a variation in problem formulation which depends on requirements. Every design problem has unique characteristics and requirements that need specific problem formulation. Every problem in mechanical component design needs to get multiple optimum values of the design variables that satisfy threshold values and constraints. Data generation aims to produce many combinations of design points to explore and exploit the design space so that the machine learning model can predict the combinations of the new design points and classify whether they meet the required class or not.

To generate synthetic data for mechanical component design problems, first, the problem must be well formulated by identifying the objective function, constraints, and design bounds. The objective of the design problem might be minimizing or maximizing some quantities such as weight, stress, cost, deflection, heat, etc. These quantities are dependent on the design variables and should be adequately expressed. The constraints are boundaries of the design space that limit the extent to which the objective function values are acceptable. Design bounds are the range of design variables considered convenient to get the best performance of the component or the product.

After formulating the problem, the next step is to generate sample design points within the given bounds. Since there are infinite points within the ranges, the Latin hypercube sampling method was implemented to explore the design spaces uniformly and without repetition of design points. The Latin hypercube sampling method is more efficient than other methods [39]. It has memory to avoid repetition of design points during sampling so that it reduces computational time. For each design problem, a set of 10,000 samples of design points was generated using a Latin hypercube

generator in Python 3.10 to serve as training data; further details are presented in Appendix A. The generator creates design points between 0 and 1 and can scale them to the desired lower and upper bounds.

Using the design points, simulation is performed to evaluate responses of objective function and constraints. From the responses, we can identify promising design variable points that satisfy the objective function threshold values and constraint equation limits. This means we can identify the feasible design space and then generate more design points and classify whether they satisfy the requirements. Based on our interest, economic value, and safety we can group the responses into two, three, or more classes (labels) such as feasible or not feasible, feasible with low performance, feasible with moderate performance, feasible with high performance, and maybe more. The simulation can be performed using analytical software to evaluate analytical expressions or finite element analysis methods for numerical expressions. In this thesis, the selected problems have analytical expression i.e., they do not require expensive simulations to obtain actual responses. However, they have different characteristics and a different number of constraints which narrows the design space and makes it difficult to obtain many preferable points from a small design space. As a result, it challenges the machine learning algorithms to classify the given points into the correct class.

3.3 Model Fitting

In the field of machine learning, the process of model fitting involves identifying the most suitable values for a model's parameters. This enables the model to generate accurate predictions when presented with new data. A properly fitted model produces more precise results by closely approximating the output. On the other hand, an overfitted model matches the data too closely, while an underfitted model doesn't match closely enough.

A machine learning model may have two types of parameters, model parameters and hyperparameters. The model parameters can be determined by the normal training process with training data, whereas hyperparameters cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model, such as its complexity or how fast it should learn. Therefore, hyperparameters should be adjusted to obtain an optimal model. In this work, the Bayesian optimization method is used to get the optimum values of the hyperparameters for SVM,

RF, and GNB models. Bayesian optimization picks a prior belief and then searches the parameter space by enforcing and updating those prior beliefs during training, i.e., prior beliefs influence future predictions. The grid search method is employed to tune the hyperparameters of the NN model.

The effect of tunning of hyperparameters on different data sizes was investigated. Three types of data size, small, medium, and large data were prepared and hyperparameters were tuned on these datasets to observe the effect of data size on the classifier performance. The results obtained from tunning will be compared with the results of default values. Performing hyperparameter tuning helps us to enhance and understand the impact of that tuning on the classifier performance. Hyperparameters of the different machine learning algorithms used in this research are shown in Table 10.

Table 10 Tunned hyperparameters of machine learning algorithms

ML algorithm	**Hyperparameters**	**Values**
SVM	C	0.01 to 100
	γ (gamma)	0.001 to 10
RF	Number of trees	100 to 600
	maximum tree depth	10 to 50
NN	Number of neurons in hidden layers	[100,100], [300,200], [500,400], [700, 600], [900,800]
	Activation function	Identity, ReLU, Logistic and tanh

3.4 Machine Learning Classification Models Evaluation

This section discusses various evaluation metrics that have been utilized to assess the model's performance. The study evaluated how well the machine learning models were able to identify viable design options. Assessing the effectiveness of a machine learning model involves model evaluation, where various evaluation metrics are utilized to determine its performance, strengths, and limitations. This process is crucial during the initial research phases and also plays a significant role in model monitoring. Evaluating the model is essential to ensure optimal and accurate results when it is deployed in production.

3.4.1 Cross Validation

In machine learning, cross-validation is a method used to assess the accuracy of a model on new data. This involves dividing the data into several subsets or folds, selecting one of these as the validation set, and training the model on the remaining folds. To ensure the model's performance is not hindered by overfitting, the cross-validation technique is used. This involves repeating the process multiple times, with each repetition using a different fold as the validation set. The results from each validation step are then averaged to produce a more reliable estimate of the model's performance. By evaluating the model on various validation sets, cross-validation provides a more realistic estimation of the model's ability to perform well on new and unseen data. It is also useful in optimizing the hyperparameters of a model, such as the regularization parameter, by selecting the values that produce the best performance on the validation set.

There are several types of cross-validation techniques, including k-fold cross-validation, leave-one-out cross-validation, and stratified cross-validation. The choice of technique depends on the size and nature of the data, as well as the specific requirements of the modeling problem. In this work, 10-fold cross-validation has been used to evaluate the performance of the machine learning models. This approach results in a less biased or less optimistic estimate of the model skill than other methods.

3.4.2 Binary and Multiclass Classification

For studying model performance at various dataset sizes, we have prepared both binary and multiclass label datasets. In binary classification, the designer can choose between feasible or not feasible options. To offer more solution options for the design problem, we can set requirements at different levels based on cost, quality, and safety. This may result in infeasible classes, feasible with moderate performance, and feasible with high-performance classes. After training on the dataset, the performance of machine learning models can be evaluated by how they classify new design variable values into different classes.

Moreover, multiclass classification helps the designer to partition the design space into different regions and observe the most promising region to further exploit the required design space according to requirements and to adjust the performance threshold values. For small dataset sizes, the design space will become smaller for each class; hence the probability of getting the most

promising design variable points increases. Based on the class labels same product can be designed for different but specific applications.

In addition, utilizing multiclass classification aids in dividing the design space into distinct regions, allowing for the identification of the most promising area to further explore and adjust performance threshold values based on specific requirements. With smaller dataset sizes, the design space for each class becomes more limited, increasing the likelihood of obtaining optimal design variable points. Furthermore, the same product can be designed for various specific applications based on class labeling.

A study was conducted to compare the performance of models on binary and multiclass labels. The study utilized a convergence plot on Python, specifically the sklearn package. Furthermore, the comparison study involves the application of accuracy, recall, and precision metrics. For more information on the convergence plot of models for both class classifications, refer to Figure 10-15.

3.4.3 Classifier Model Evaluation Metrics

Classification metrics rely on the confusion matrix, which shows the accurate and inaccurate predictions of a class. The goal of classification metrics is to determine how accurately the model classifies design variable data points into their proper categories. The confusion matrix presents four possible outcomes as shown in Table 11: true positive (TP) when the model correctly predicts a positive class, true negative (TN) when the model correctly predicts a negative class, false positive (FP) when the model incorrectly predicts a negative class as positive, and false negative (FN) when the model incorrectly predicts a positive class as negative. Most of the various metrics used in classification problems apply some combination from the entries of the confusion matrix. The confusion matrix used in multiclass classification is an expansion of the binary classification matrix. The columns show the expected distribution of classes, while the rows display the predicted distribution as determined by the classifier.

Table 11 Confusion matrix table for binary classification

		Actual Class	
		P	N
Predicted Class	P	TP	FP
	N	FN	TN

Accuracy:

Accuracy shows how the design variable points are close to their correct classes. When analyzing data, it's important to be cautious when using accuracy as a metric. If the dataset is imbalanced, relying solely on accuracy may result in misinterpretation. Accuracy can be expressed as follows,

$$ACC = \frac{TP+TN}{TP+FP+FN+TN} \quad (3.30)$$

Recall (True Positive Rate, Sensitivity):

Identifying design variables that are assigned as infeasible and feasible is important from the perspective of the designer's point of view since these values are important to get optimum design options. This can be achieved using the recall metric. Recall refers to the classifier's ability to identify all False (F) samples. This metric is expressed as follows:

$$TPR = Recall = \frac{TP}{TP+FN} \quad (3.32)$$

Precision (**Positive Prediction Value**):

Precision is used to detect the design variable points that are wrongly assigned in feasible design space. The precision metrics of a classifier are typically measured by its ability to correctly identify a sample as either true (T) or false (F), without mislabeling it. This is expressed in the following formula:

$$Precision = \frac{TP}{TP+FP} \quad (3.33)$$

F1-measure:

The F1 score is an often-used metric for binary classification, which takes into account both precision and recall by using a harmonic mean. It is particularly useful in evaluating the performance of classifiers in cases where there is an imbalance in the class or where better

performance is needed for the positive class. The formula for calculating the F-measure is as follows:

$$F_1 = \frac{2*precision*recall}{precision+recall} \quad (3.34)$$

3.5 Implementation

When conducting research using machine learning algorithms, it is important to consider the implementation environment, as it can greatly impact training times and evaluation criteria. Careful consideration should be given even before designing experiments and simulations. Selecting a suitable working environment can also aid in the preparation of reliable datasets.

Python programming language was chosen over other software packages due to its numerous advantages. Python is undoubtedly one of the most powerful and versatile programming languages available today. It is a user-friendly interface and simple syntax make it a top choice for developers of all levels of expertise. It is available for free and is distributed under the BSD license. It has an easy-to-use interface and comes with excellent documentation. The sklearn package in Python provides a standardized interface for various machine learning algorithms and offers multiple tuning parameters for each algorithm, along with reasonable default settings. Additionally, it offers a vast range of features for tasks related to machine learning. The SciPy package provides many types of sampling strategies to sample points from available design space.

3.6 Summary

The research methodology was expertly executed and can be easily understood through the workflow presented in Figure 7. This structured framework guided us through each step of the process, from defining the problem to implementing the solution, ensuring a logical and successful outcome. Despite many technical challenges, the task was completed successfully with precise and demanding attention to detail.

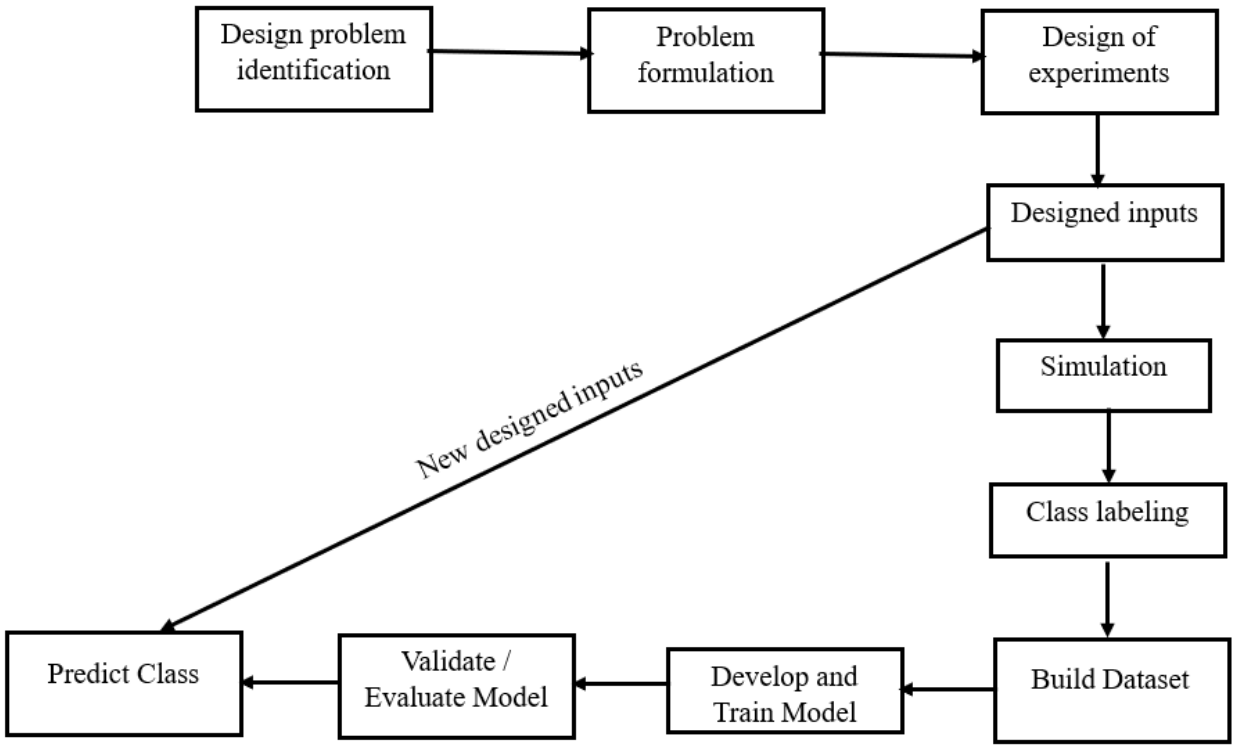


Figure 7 Workflow of the proposed methodology

The architecture describes each process of mechanical component design space exploration and optimization as shown in Figure 8.

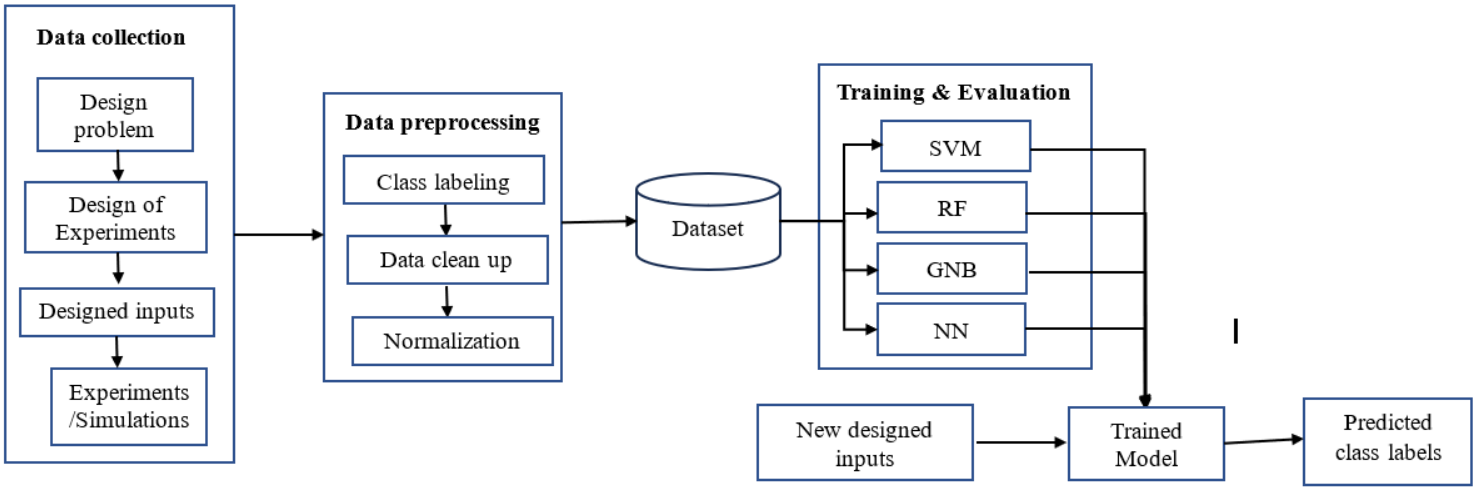


Figure 8 Mechanical component design space exploration and optimization system architecture

CHAPTER FOUR

RESULT AND DISCUSSION

This chapter presents the results of the research based on the methodology described in chapter three. Different machine learning methods are tested on different sizes of datasets to evaluate their performance for classifying given design points. Small, medium, and large datasets have been prepared to check the performance of machine learning methods on binary and multiclass classification of the three design problems. Hyperparameter tuning was done to improve the performance of the algorithms. The evaluation metrics used to evaluate the performance of the methods were accuracy, precision, recall, and f1 score which are common metrics for classification problems.

As mentioned in chapter three the selected sampling method was to select sample points uniformly from the available space so that all the regions of the design space are covered and no duplication of data points are allowed. The codes that are used to generate sample design points are depicted in Appendix A.

The efficiency of the machine learning techniques was evaluated across three design problems, utilizing datasets of varying sizes for both binary and multiclass categorization. Further discussion on the outcomes can be found in the following sections.

4.1 Dataset Visualization

Before the dataset is feed into a machine learning model it must be visualized to see the distribution of data points within the given design space. In order to the classifier model to recognize feasible design spaces and optimal values of design variables based on the performance requirement, we need to prepare the dataset accordingly. By visualizing the distribution of data points, we may resample data points from the available design space to get more representative data points that will aid to prepare a more reliable dataset.

We employed principal component analysis (PCA) and t-distributed stochastic neighbor embedding (t-SNE) data visualization techniques to observe the distribution of classes in the dataset. By changing the threshold value of the requirement, it is observed that the number of feasible classes decreases, which means it is possible to get the optimized results of the design

problem by tightening the performance threshold. Figure 9 shows the scatter plot of PCA analysis for binary classification of the belt pulley drive design problem. As shown in the figure, data points of the feasible class decrease as the performance threshold decreases, which indicates the possibility of shrinking the design space of the feasible class to the minimum for this specific case, i.e., optimizing the design space.

However, narrowing the performance threshold value may create an imbalanced dataset which will influence the performance of the algorithm. It is important to note that imbalanced datasets can greatly affect the sensitivity levels of various classifiers. To handle such problems, we can use more suitable evaluation metrics such as ROC and AUC in case of imbalanced datasets.

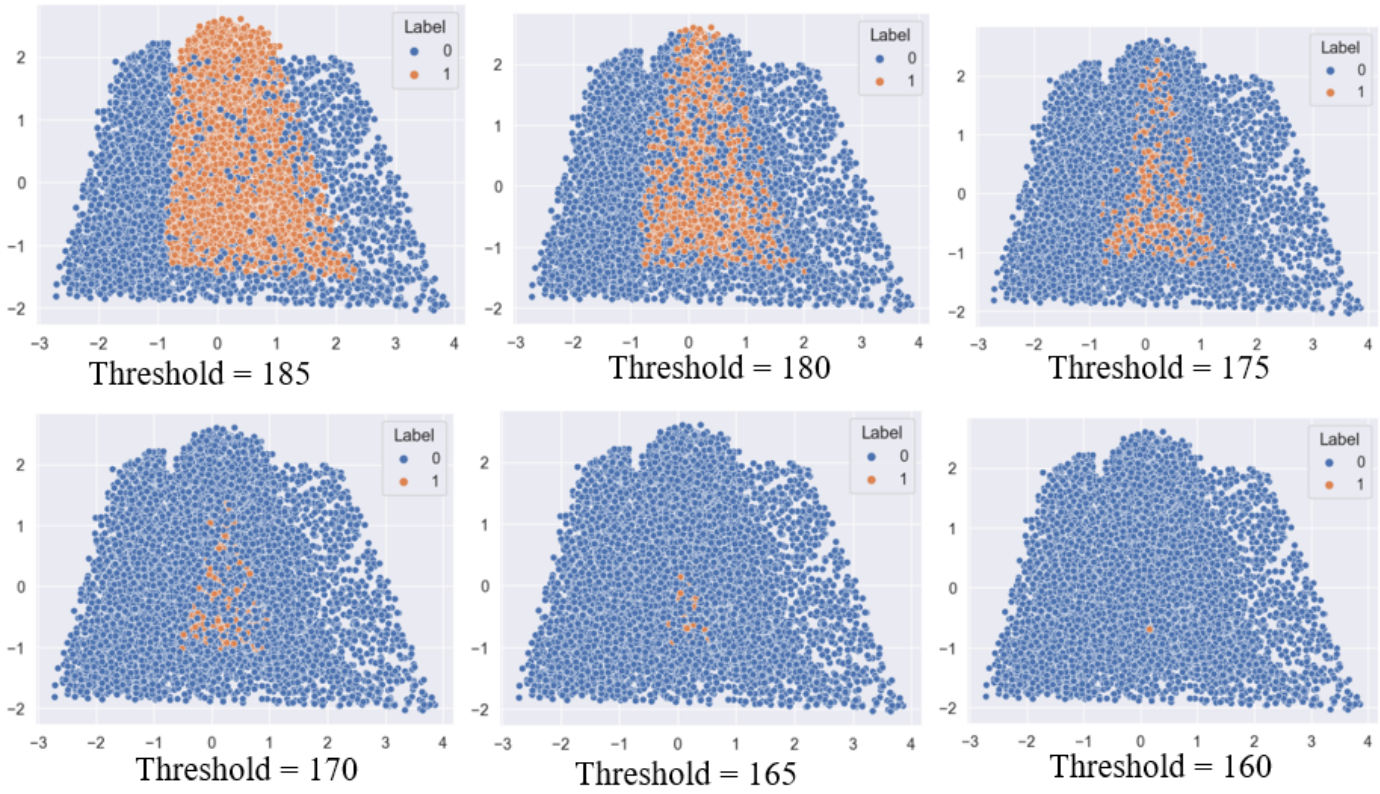


Figure 9 PCA scatter plot of BPD binary datasets with 10,000 training points as the threshold varies.

In Figure 10, the class distribution of the multiclass classification for the belt pulley drive problem is presented, clearly depicting how the class balance shifts as the threshold value decreases. It is important to note that the count of the most significant class reaches its lowest point. Significantly, this reduction is more rapid in multiclass classification compared to binary classification.

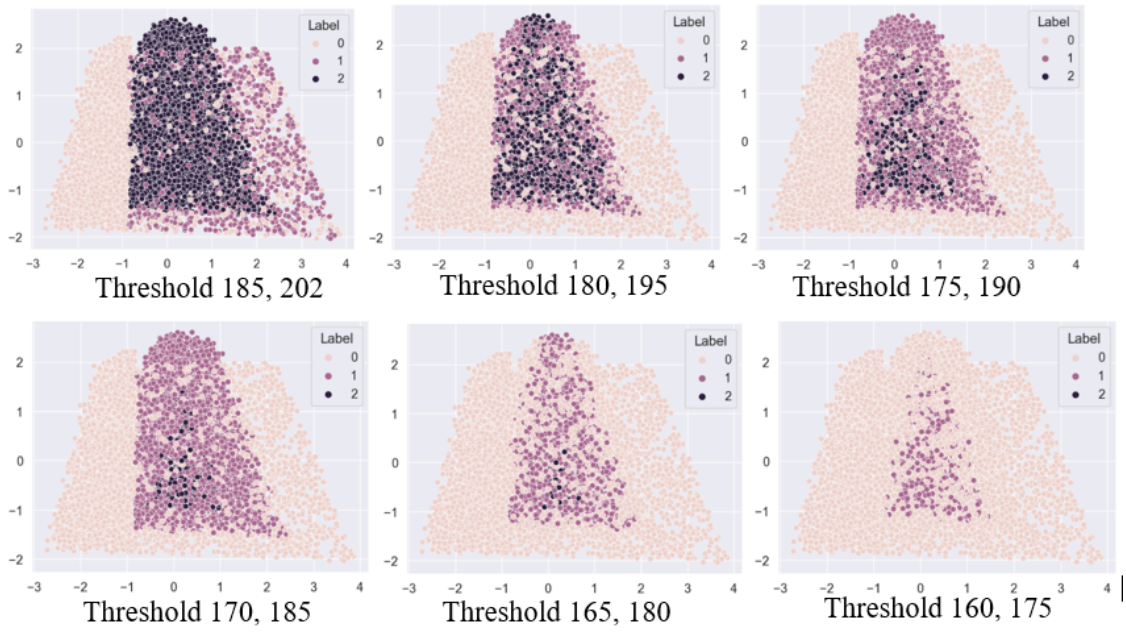


Figure 10 PCA scatter plot of BPD multiclass datasets with 10,000 training points as the threshold

4.2 Classifiers' Model Fitting

In order to enhance the performance of the machine learning models, it is necessary to adjust the hyperparameters of the model. These hyperparameters help us to control the behavior and find the right balance between the bias and variance. Even though there are no hard and fast rule of hyperparameter tuning for performance optimization that guarantee best performance on a given dataset, but there are methods to find the best hyperparameter values. In this research Bayesian optimization method has been utilized to obtain best results using Optuna module on Python. The obtained values are fitted on the model and a convergence plot is drawn to observe the effect of tuning.

4.2.1 Binary classification

The purpose of dataset preparation for binary classification is to identify feasible regions of the design space from the infeasible ones based on design requirements. Three sizes of datasets were prepared to see the effect of the available training data on the performance and sensitivity to tuning.

The first dataset was large size which contains 10,000 data points, the second was medium size which contains 1000 data points and the third one was small size which contains 200 data points. To compare the results of tuning with the default parameters and tuning with optimized hyperparameters a convergent plot was drawn as shown in the following figures.

Figure 10 shows the convergence plot of training sizes versus accuracy score on the three design problems with the tuned and untuned hyperparameters. As shown in Figure 11 some classification algorithms enable more self-tuning than others, leading to different levels of robustness to tuning among the various classifiers. RF dominates the illustrated design problems when the algorithms are untuned, indicating that RF provides reasonably good performance without tuning. However, NN and SVM become competitive and typically outperform the RF when tuned as shown in the HSD problem. Hyperparameter tuning is not performed for the GNB algorithm because in Scikit learn it uses maximum likelihood estimation to automatically learn kernel widths and it has only one hyperparameter if necessary to tune but tuning has no reasonable effect on GNB as described in previously done research[6].

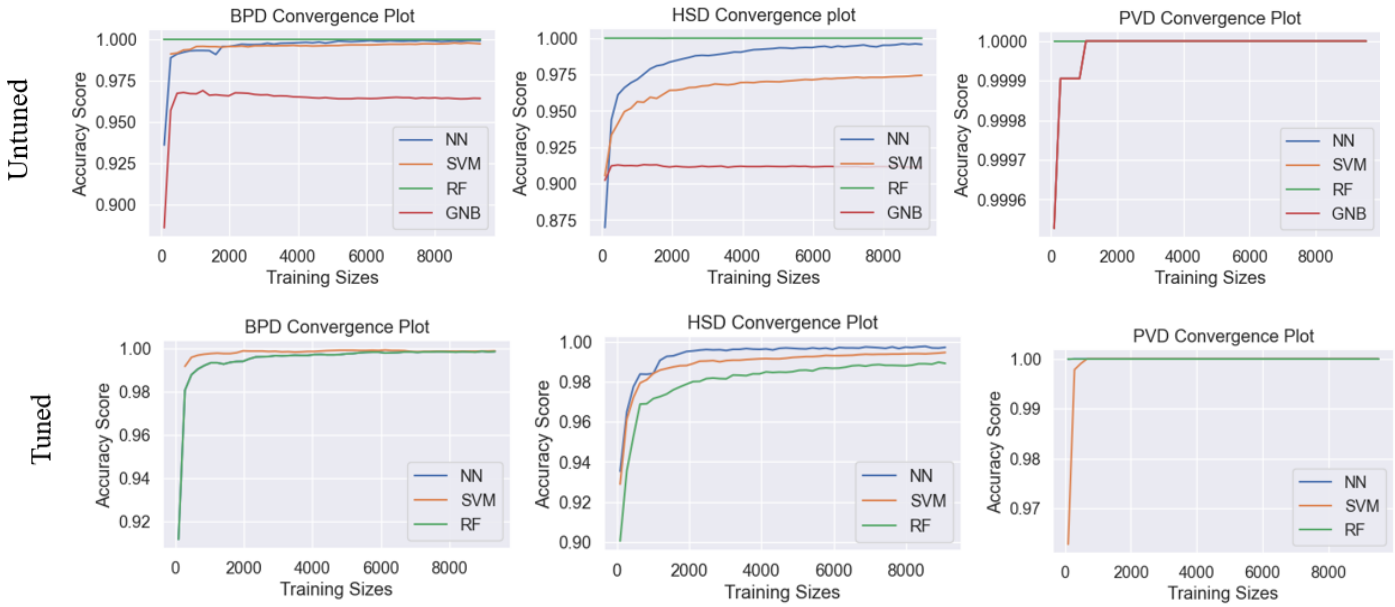


Figure 11 Convergence plots displaying accuracies for tuned and untuned algorithms on large-size datasets on binary classification

In Figure 12 the plot of training sizes versus accuracy score is depicted using the medium dataset size to observe the effect of the size of the dataset on the performance of the algorithms on the design problems. While RF performs better on the untuned parameters than the others, GNB performs poorly on the PBD and HSD problems. Both NN and SVM improve their performance when tuned in to all three problems. NN showed significant variation and dips in accuracy, while the RF algorithm showed smoother convergence curves. These local decreases in accuracy are likely the result of overfitting, such that the classifier performs well on the training data but more poorly on the test data. The degree of overfitting can vary with the locations of new data points in the training and testing sets and the randomness of the division of training data into cross-validation training sets. Other authors in the engineering design literature have observed similar nonmonotonic decreases in classification performance[6], [49].

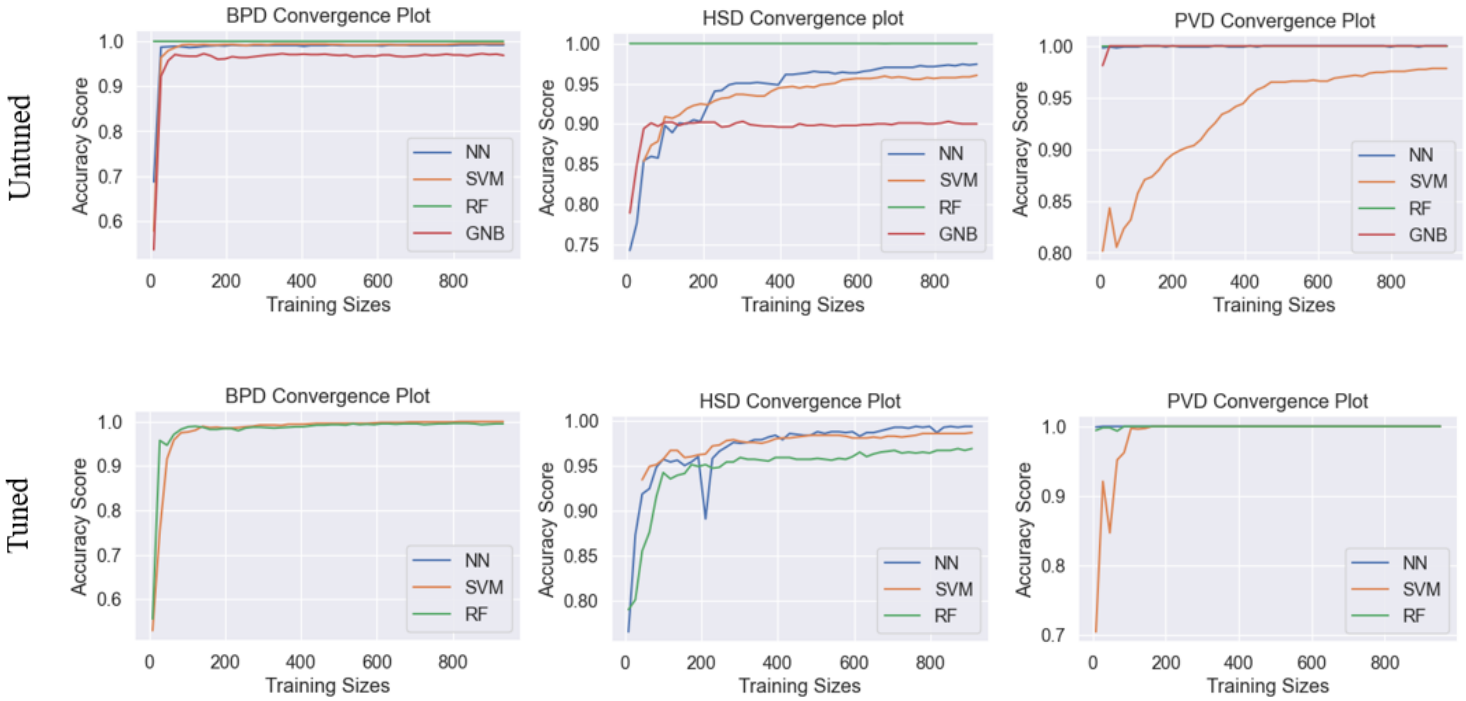


Figure 12 Convergence plots displaying accuracies for tuned and untuned algorithm on medium size dataset on binary classification

We randomly sampled small datasets without replacement from each large dataset to examine the impact of dataset size reduction on classification performance. It is evident from Figure 13 that changes in accuracy are more noticeable in the small dataset as compared to the medium and large datasets when the number of training size changes. Small datasets often lack sufficient details,

making it difficult for the classification model to identify patterns in the training data. Moreover, overfitting is more challenging to prevent, as it may extend beyond the training data and impact the validation set.

On default parameters, RF shows good performance, but NN and SVM can improve performance with the tuning of hyperparameters. GNB performs better on the PVD problem in all dataset sizes, but in other cases, its performance is weaker compared to the other methods.

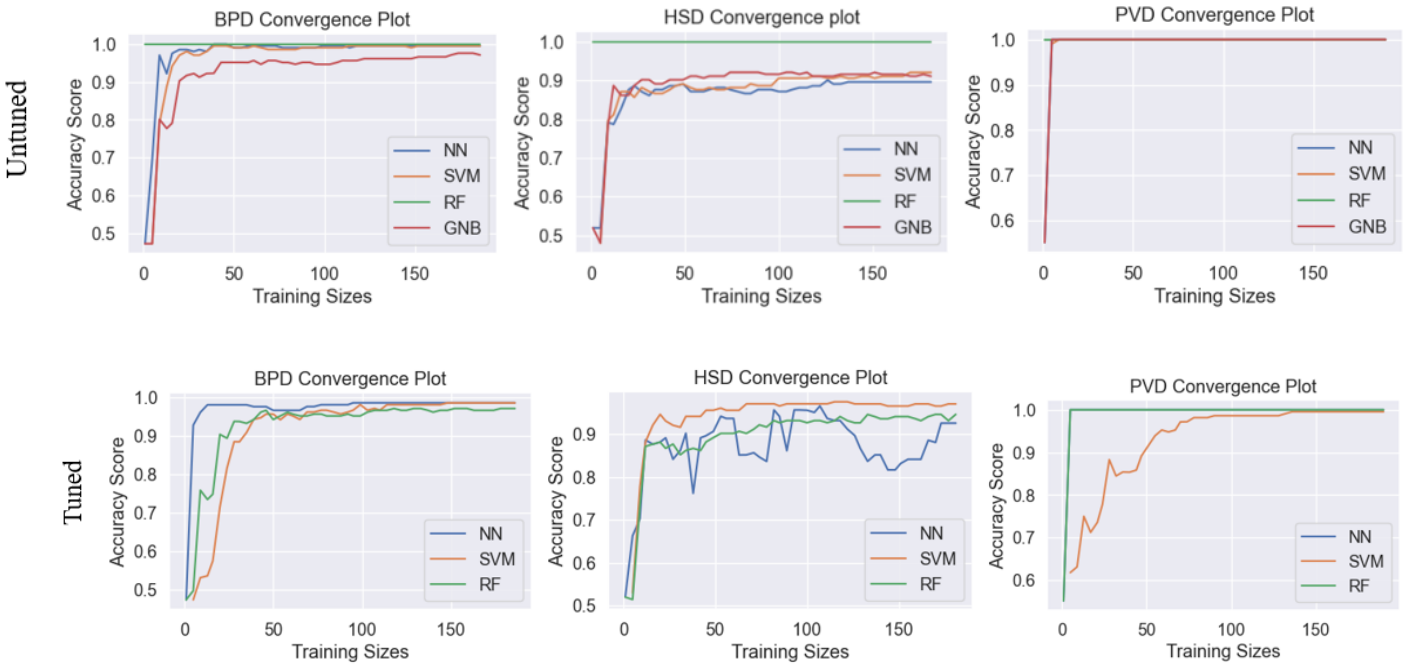


Figure 13 Convergence plots displaying accuracies for tuned and untuned algorithms on small-size datasets on binary classification

4.2.2 Multiclass Classification

Performing multiclass classification in design problems has the advantage of dividing the design space into distinct regions, which helps narrow down the design possibilities and leads to optimized values of design variables. Based on the large dataset displayed in Figure 14, it was observed that RF outperforms with default parameters in comparison to the other methods, and NN and SVM improve their performance when tuned while RF's performance declines after tuning. GNB, on the other hand, performs poorly across all problems.

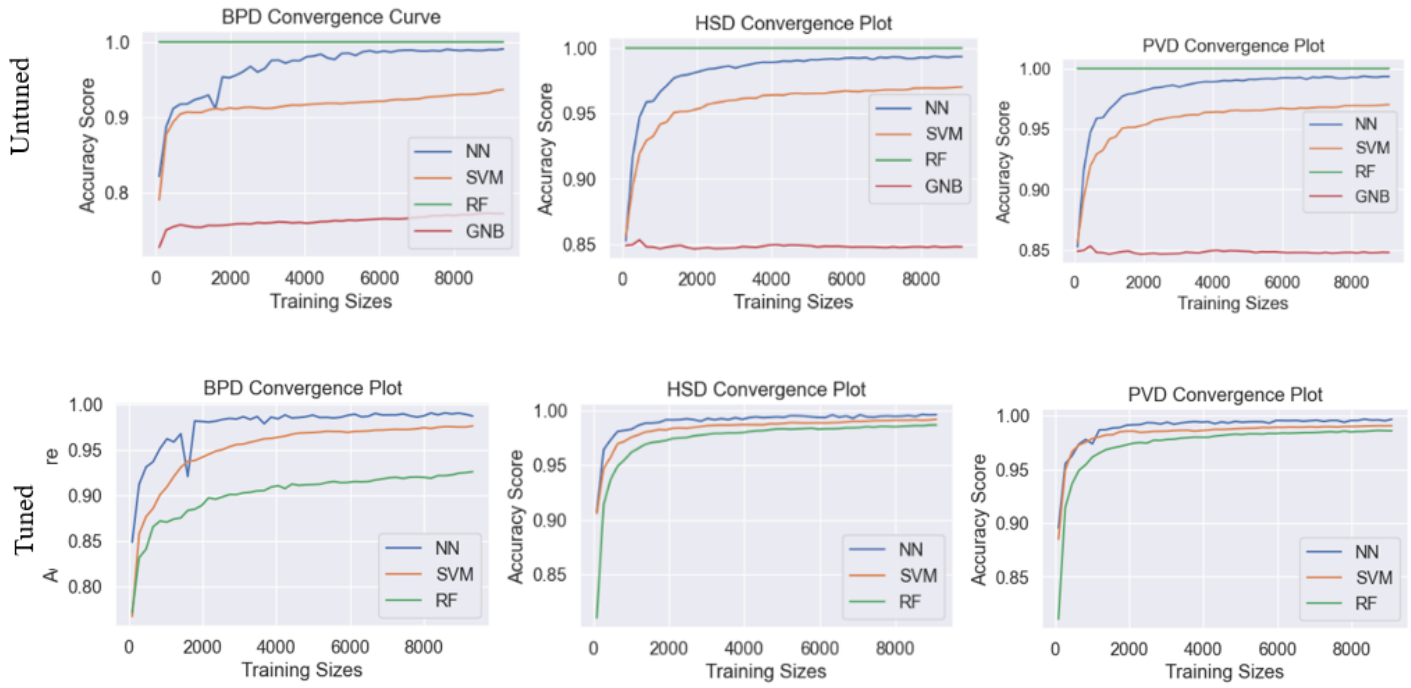


Figure 14 Convergence plots displaying accuracies for tuned and untuned algorithms on large-size datasets on multiclass classification.

In the case of medium-sized datasets that are not tuned, RF displays impressive performance, but its efficiency decreases significantly when it is tuned. This is because including additional features can have a negative impact on performance as it may introduce irrelevant data that can hinder the model's ability to learn the true relationships. While the random forest method performs implicit feature selection by splitting nodes based on the most important variables, other machine learning models do not have this capability. The NN and SVM score higher accuracies than RF when they are tuned. On the other hand, GNB consistently exhibits poor performance.

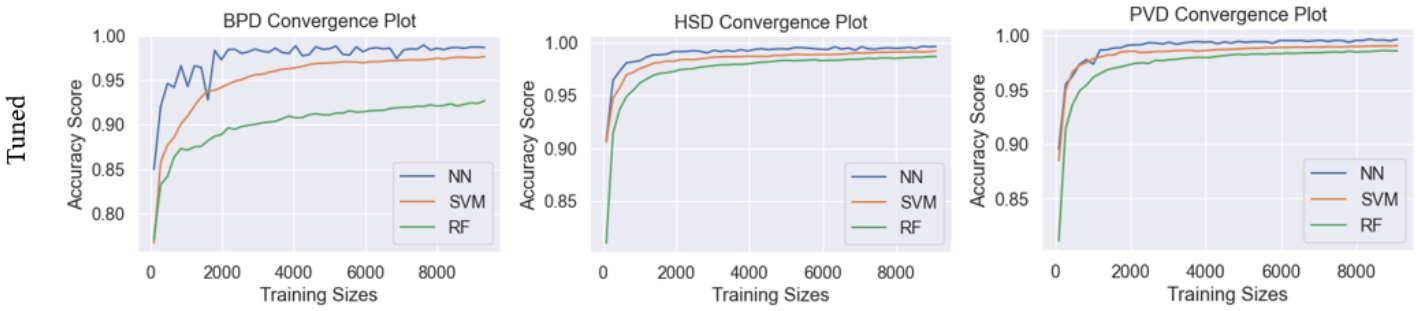


Figure 15 Convergence plots displaying accuracies for tuned and untuned algorithms on medium-size datasets on multiclass classification.

The irregularity of the plotting curve increases in the small dataset as the training size varies (see Figure 16). RF performs greater than the others in untuned plotting but a similar performance with the NN and SVM is seen in the tuned plot. From one problem to another, each algorithm has similar performance and the maximum achievement is equivalent to the other dataset plots.

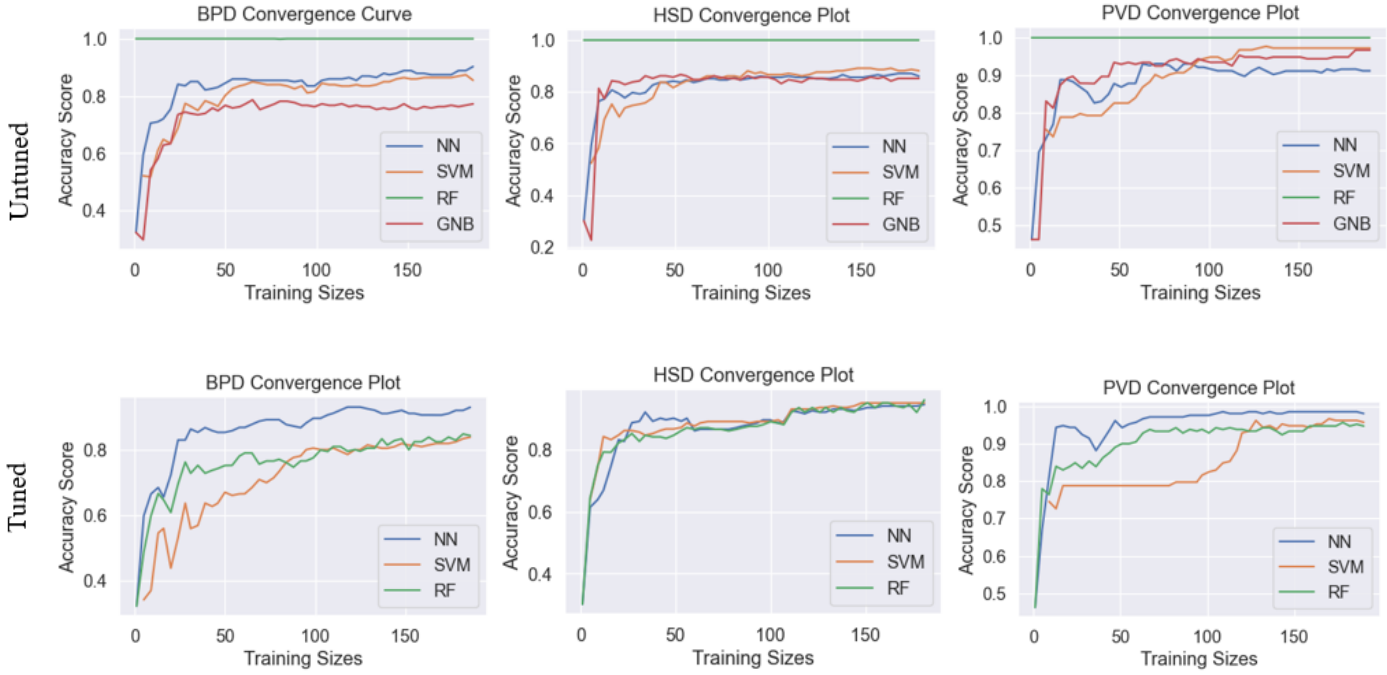


Figure 16 Convergence plots displaying accuracies for tuned and untuned algorithm on small size dataset on multiclass classification

4.3 Evaluation of the Classifier Models

In this section, we have compiled tables that present the accuracy of the classifiers for both binary and multiclass classification, across different dataset sizes for all the design problems. Various evaluation metrics, such as accuracy, precision, recall, and f1 score, have been employed in accordance with the methodology elaborated in Chapter 3. To delve into the specifics of precision, recall, and f1 score results, please refer to Appendix D.

4.3.1 Binary Classification

In Tables 1-3, it can be seen that there are no significant differences in the performance of classifiers for binary classification, regardless of the dataset size for all problems. This suggests that the effectiveness of classifiers is more dependent on how well the dataset represents the original distribution rather than its size. Utilizing a small number of data points that accurately represent the design space can help reduce the sampling budget.

The PVD problem shows high accuracy for all classifiers, but in other problems, RF, NN and SVM outperform the GNB classifier. NN's and SVM's performance improves with hyperparameter tuning across all dataset sizes, while RF's performance remains unchanged or decreased with tuning. The maximum accuracy is scored is 100% on the PVD problem with all classifiers and the lowest is accuracy scored is 86.8% on the HSD with GNB classifier.

Table 1 Binary classification accuracy of classifiers trained with 10,000 training points for the three design problems

Algorithm	**Pressure vessel**		**Helical coiled spring**		**Belt pulley drive**	
	Untuned	**Tuned**	**Untuned**	**Tuned**	**Untuned**	**Tuned**
SVM	1.0	1.0	0.973	0.973	0.994	0.994
NN	1.0	1.0	0.992	0.989	0.997	0.999
RF	1.0	1.0	0.973	0.973	0.996	0.996
GNB	1.0	-	0.914	-	0.993	-

Table 2 Binary classification accuracy of classifiers trained with 1,000 training points on the three design problems

Algorithm	**Pressure vessel**		**Helical coiled spring**		**Belt pulley drive**	
	Untuned	**Tuned**	**Untuned**	**Tuned**	**Untuned**	**Tuned**
SVM	1.0	1.0	0.963	0.990	0.990	1.0
NN	1.0	1.0	0.966	0.990	0.990	1.0
RF	1.0	1.0	0.963	0.963	0.990	0.990
GNB	1.0	-	0.887	-	0.964	-

Table 3 Binary classification accuracy of classifiers trained with 200 training points on the three design problems

Algorithm	Pressure vessel		Helical coiled spring		Belt pulley drive	
	Untuned	Tuned	Untuned	Tuned	Untuned	Tuned
SVM	1.0	1.0	0.918	0.983	1.0	1.0
NN	1.0	1.0	0.868	0.983	1.0	1.0
RF	1.0	1.0	0.918	0.918	1.0	1.0
GNB	1.0	-	0.868	-	1.0	-

4.3.2 Multiclass classification

Tables 4-6 indicate an important trend in multiclass classification: classifiers generally exhibit lower accuracy as dataset size decreases. Among the classifiers examined, neural network (NN) achieves the highest accuracy of 99.3%. Conversely, Gaussian Naive Bayes (GNB) achieves the lowest accuracy of 75.7% when working with a large dataset for the belt pulley drive design problem. Random Forest (RF) performs well even without fine-tuning, but it's worth noting that NN's and SVM's accuracies improve significantly with fine-tuning.

Table 4 Multiclass classification accuracy of classifiers trained with 10,000 training points on the three design problems

Algorithm	Pressure vessel		Helical coiled spring		Belt pulley drive	
	Untuned	Tuned	Untuned	Tuned	Untuned	Tuned
SVM	0.771	0.979	0.969	0.991	0.926	0.987
NN	0.915	0.960	0.993	0.993	0.989	0.990
RF	0.983	0.983	0.982	0.969	0.918	0.918
GNB	0.849	-	0849	-	0.757	-

Table 5 Multiclass classification performance of classifiers trained with 1,000 training points on the three design problems

Algorithm	Pressure vessel		Helical coiled spring		Belt pulley drive	
	Untuned	Tuned	Untuned	Tuned	Untuned	Tuned
SVM	0.752	0.924	0.933	0.980	0.926	0.987
NN	0.915	0.960	0.993	0.993	0.989	0.987
RF	0.950	0.950	0.957	0.933	0.918	0.918
GNB	0.864	-	0.867	-	0.757	-

Table 6 Multiclass classification accuracy of classifiers trained with 200 training points on the three design problems

Algorithm	Pressure vessel		Helical coiled spring		Belt pulley drive	
	Untuned	Tuned	Untuned	Tuned	Untuned	Tuned
SVM	0.890	0.812	0.819	0.901	0.857	0.888
NN	0.593	0.921	0.803	0.934	0.857	0.904
RF	0.875	0.875	0.885	0.819	0.825	0.825
GNB	0.921	-	0.786	-	0.825	-

4.4 Discussion

This research addresses application of supervised machine learning classifiers in mechanical component design problems for design space exploration and optimization. Although there are numerous techniques for design space exploration and optimization, they may not efficiently address the various characteristics of certain problems. Fortunately, machine learning classifiers can overcome these limitations by being applicable to all types of design problems and effectively addressing their unique features.

The use of supervised machine learning techniques has been found to be highly effective in engineering design exploration and optimization applications. These techniques are particularly useful for identifying promising or feasible regions within the design space, which can then be used to inform early-stage design exploration, provide reliability assessments, and aid convergence in multi-objective or multilevel problems that require collaborative design teams[6].

4.4.1 Binary Classification

Drawing from the convergent plots in section 4.2.1, it is evident that RF exhibits superior performance compared to GNB, NN and SVM across all three design problems and dataset sizes, provided the hyperparameters are not tuned. Nevertheless, when hyperparameters are fine-tuned, NN’s and SVM's performance shows a marked improvement, while RF's performance dips. It is worth noting that the fluctuations in performance, as the number of training points varies, are more pronounced in smaller dataset sizes, even with finely-tuned parameters.

Results of section 4.3.1 indicates that the highest accuracy scored with all the models is in the PVD problem in all the three dataset sizes. Characteristics of the PVD problem is single nonlinear objective function with four design variables and four design constraints. Contrast to this, the lowest accuracy score is observed in the HSD problem which is scored by the GNB model in the

small dataset size. HSD problems is single objective with nonlinear character, highly constrained which has eight design constraints and three design variables. From the design variables one takes an integer values the other two take continuous values. In this problem the accuracy of all the models decreases as the dataset size decreases. All the three models have similar accuracy score in the BPD problem across different dataset sizes. The nature of the BPD problem is single objective with nonlinear character, three design variables with continuous values and with two design constraints. This indicates that the selection of algorithms has no effect on low design constraint problems. Contrast to this in case of highly constrained problems, the performance of different algorithms varies so proper selection of algorithm is important to get higher accuracy.

4.4.2 Multiclass Classification

In the analysis outlined in section 4.2.2, it was noted that the convergence plots for multiclass classification exhibited a distinct performance trend with respect to training size for each model. This differs from the binary classification convergence plots presented in section 4.2.1. The reason for this discrepancy is that, in multiclass classification, there are more classes to differentiate the design space and meet the desired threshold values. RF models continued to outperform the other models in all three design scenarios with the default hyperparameters, much like binary classification. However, GNB exhibited the poorest performance. Hyperparameter tuning significantly boosted NN and SVM performance in all cases, surpassing the RF models.

According to the findings in section 4.3.2, the accuracy of the models tested varied based on the design problems and dataset sizes. Notably, in the BPD problem, which is relatively unconstrained, the GNB model recorded the lowest accuracy value (75.7%) in medium and large dataset sizes. On the other hand, in the moderately constrained PVD problem, the GNB model achieved a high score of 92.1% in a small dataset size.

It has been noted that multiclass classification poses a challenge for classifiers, as there are inconsistencies in accuracies across different problems and dataset sizes. As highlighted in section 4.3.2, the accuracy of RF and NN decreases as the dataset size decreases in all design problems. On the other hand, the accuracy of SVM increases in small dataset sizes, with an 89% accuracy rate, compared to 75.2% for medium dataset sizes and 77.1% for large dataset sizes. Hyperparameter tuning has been effective in improving the accuracy of NN and SVM across all

dataset sizes. However, in the HSD problem, RF's accuracy decreases in all dataset sizes, while no changes are observed in other problems.

This study highlights the potential benefits of incorporating machine learning into engineering design, particularly in machine design, to address optimization challenges and explore innovative design concepts. Machine learning can be leveraged as a valuable tool to expedite the design process, for instance, via the use of an RF model to navigate design space without the need for fine-tuning when time is a constraint. The research also sheds light on useful insights regarding dataset sizes and classification types. While previous research has primarily focused on machine learning for material property prediction, surrogate models, and failure analysis, this study demonstrates the efficiency of machine learning classifiers in addressing mechanical component design problems, specifically in the context of exploring and optimizing design space.

CHAPTER FIVE

CONCLUSIONS AND FUTURE WORKS

5.1 Conclusion

Simulation software is frequently employed in designing a product to determine the best design options. However, this can be costly since multiple iterations of different objective functions and constraint equations are required to achieve the desired outcomes. To develop a superior product that meets customer needs while minimizing costs and development time, exploring alternative solutions from existing design alternatives is crucial. Hence, a tool that aids mechanical designers and engineers in generating multiple problem solutions that satisfy client demands while conserving resources is of the utmost importance.

It can be challenging to explore and optimize the design space of a complete mechanical system, given the numerous objective function's design variables and complex constraints involved. As such, analyzing the individual components or subassemblies is often more practical and technically feasible. These mechanical components are critical for developing any machinery, and by conducting design space exploration and optimization studies, valuable insights can be gained for marketing various products with optimized components. Ultimately, this can result in improved quality, reduced costs, and faster time-to-market.

This research analyzed the feasibility of using machine learning classifiers for design space exploration and optimization in mechanical component design problems. To apply machine learning classifiers to classify design points to desired classes, data points are generated using Latin hypercube sampling. This efficient statistical sampling method takes samples from all the available design spaces without repetition. The objective function and design constraint equations are simulated using the sampled design points. Then, by setting a threshold value on the objective function a class label can be specified that satisfies the design requirements and the constraints. This creates an input-output relationship between the sampled design points and the class labels. The input-output relationship created can help to classify new sampled points using machine learning classifiers. To get the desired input-output relationship a design of experiment study was conducted.

For this research, three mechanical component design problems were proposed to test and evaluate the performance of four supervised machine learning classifiers. The datasets were prepared with three different sizes to study the effect of dataset sizes on the performance of the classifiers. To narrow the design spaces and provide more options, multiclass datasets were prepared with three different sizes, like binary classification.

According to the study, RF classifiers demonstrated superior performance to other classifiers for all three problems across all three dataset sizes using default model hyperparameters. However, NN and SVM classifiers showed improved performance with fine-tuning. The binary classification of the PVD problem with all four algorithms achieved a remarkable accuracy of 100%. In contrast, the multiclass classification of the BPD problem with the GNB classifier resulted in the lowest accuracy of 75.7%. Notably, the performance of all classifiers was relatively lower in multiclass classification than in binary classification.

The results of this study offer valuable insights to engineers and designers to select appropriate algorithms for a specific design problem. By analyzing classifier performance on related problems, engineers can make informed decisions about which algorithm to use. Moreover, the study sheds light on the amount of training data required to achieve satisfactory accuracy levels, which depends on the problem's complexity and dimensionality. This information can benefit the early stages of experiment planning and design.

5.2 Future Works

To enhance the findings of this research, further examination of related design problems can be conducted since the design problems in mechanical design are extensive. The study only evaluated the performance of four supervised machine-learning classifiers. At the same time, other forms of machine-learning techniques are also crucial to gain a better understanding of these methods for mechanical design applications. Through an in-depth examination of the performance of machine learning on imbalanced datasets, one can enhance the overall design and attain superior outcomes. Different sampling methods, such as heuristic strategies, can improve the quality of information obtained for small datasets through experimental design.

References

[1] D. K. Pratihar, ‘Traditional vs. Non-Traditional Optimization Tools’, in *Computational Optimization and Applications*, 2012.

[2] A. Stoll and P. Benner, ‘Machine learning for material characterization with an application for predicting mechanical properties’, *GAMM Mitteilungen*, vol. 44, no. 1, Mar. 2021, doi: 10.1002/gamm.202100003.

[3] K. E. Parsopoulos and M. N. Vrahatis, ‘Unified Particle Swarm Optimization for solving constrained engineering optimization problems’, in *Lecture Notes in Computer Science*, Springer Verlag, 2005, pp. 582–591. doi: 10.1007/11539902_71.

[4] X. Ke, Y. Zhang, Y. Li, and T. Du, ‘Solving design of pressure vessel engineering problem using a fruit fly optimization algorithm’, *International Journal of Simulation: Systems, Science and Technology*, vol. 17, no. 43, pp. 5.1-5.7, 2016, doi: 10.5013/IJSSST.a.17.43.05.

[5] B. Thamaraikannan and V. Thirunavukkarasu, ‘Design optimization of mechanical components using an enhanced teaching-learning based optimization algorithm with differential operator’, *Math Probl Eng*, vol. 2014, 2014, doi: 10.1155/2014/309327.

[6] C. Sharpe, T. Wiest, P. Wang, and C. C. Seepersad, ‘A comparative evaluation of supervised machine learning classification techniques for engineering design applications’, *Journal of Mechanical Design, Transactions of the ASME*, vol. 141, no. 12, Dec. 2019, doi: 10.1115/1.4044524.

[7] N. Amalina, D. Suhaimi, H. Abas, and P. Security, ‘A Systematic Literature Review on Supervised Machine Learning’, no. January, 2020.

[8] U. Urbas, D. Zorko, and N. Vukašinović, ‘Machine learning based nominal root stress calculation model for gears with a progressive curved path of contact’, *Mech Mach Theory*, vol. 165, Nov. 2021, doi: 10.1016/j.mechmachtheory.2021.104430.

[9] C. C. S. Clinton Morris, Logan Bekker, Michael R. Haberman, ‘Design Exploration of Reliably Manufacturable Materials and Structures with Applications to Negative Stiffness Metamaterials and Microstereolithography’, *Mechanical Design*, vol. 140, no. 11, 2018.

[10] X. Sajad Saraygord Afsharia, Fatemeh, Enayatollahib XiangyangXu, ‘Machine learning-based methods in structural reliability analysis: A review’, *Reliab Eng Syst Saf*, vol. 219, no. 108223, 2022, doi: https://doi.org/10.1016/j.ress.2021.108223.

[11] B. Durakovic, ‘Design of experiments application, concepts, examples: State of the art’, *Periodicals of Engineering and Natural Sciences*, vol. 5, no. 3, pp. 421–439, 2017, doi: 10.21533/pen.v5i3.145.

[12] G. H. Lewes, ‘Support Vector Machines for Classification’, no. July 2018, 2015, doi: 10.1007/978-1-4302-5990-9.

[13] D. Vlah, A. Kastrin, J. Povh, and N. Vukašinović, ‘Data-driven engineering design: A systematic review using scientometric approach’, *Advanced Engineering Informatics*, vol. 54. 2022. doi: 10.1016/j.aei.2022.101774.

[14] Y. Xiong *et al.*, ‘Erratum: “Data-Driven Design Space Exploration and Exploitation for Design for Additive Manufacturing” [ASME J. Mech. Des. 2019, 141(10), p. 101101; DOI: 10.1115/1.4044872]’, *Journal of Mechanical Design, Transactions of the ASME*, vol. 141, no. 11. American Society of Mechanical Engineers (ASME), Nov. 01, 2019. doi: 10.1115/1.4043587.

[15] T. G. Kolda, R. M. Lewis, and V. Torczon, ‘Optimization by Direct Search: New Perspectives on Some Classical and Modern Methods *’, *Society for Industrial and Applied Mathematics*, vol. 45, no. 3, pp. 385–482, 2003, doi: 10.1137/S0036144502428893.

[16] R. M. Lewis and V. Torczon, ‘Pattern Search Algorithms for Bound Constrained Minimization *’.

[17] A. Ahmad, ‘Review of Modern Optimization Techniques’, no. February 2017, 2015, doi: 10.17577/IJERTV4IS041129.

[18] A. Ahmad, ‘Review of Modern Optimization Techniques’, no. February 2017, 2015, doi: 10.17577/IJERTV4IS041129.

[19] G. Venter, ‘Review of Optimization Techniques Review of Optimization Techniques’, no. December 2010, 2018, doi: 10.1002/9780470686652.eae495.

[20] Y. del Valle, G. K. Venayagamoorthy, S. Mohagheghi, J. C. Hernandez, and R. G. Harley, 'Particle swarm optimization: Basic concepts, variants and applications in power systems', *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 2. pp. 171–195, Apr. 2008. doi: 10.1109/TEVC.2007.896686.

[21] M. A. El-Shorbagy and A. E. Hassanien, 'Particle Swarm Optimization from Theory to Applications', *International Journal of Rough Sets and Data Analysis*, vol. 5, no. 2, pp. 1–24, Jan. 2018, doi: 10.4018/ijrsda.2018040101.

[22] D. Henderson, S. H. Jacobson, and A. W. Johnson, 'The Theory and Practice of Simulated Annealing'.

[23] 'Surrogate model - Wikipedia'. Accessed: Oct. 15, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Surrogate_model

[24] J. W. B. et Al, 'Space mapping: the state of the art', *in IEEE Transactions on Microwave Theory and Techniques*, vol. 52, 2004, doi: doi: 10.1109/TMTT.2003.820904.

[25] Y. Xiong *et al.*, 'Erratum: "Data-Driven Design Space Exploration and Exploitation for Design for Additive Manufacturing" [ASME J. Mech. Des. 2019, 141(10), p. 101101; DOI: 10.1115/1.4044872]', *Journal of Mechanical Design, Transactions of the ASME*, vol. 141, no. 11. American Society of Mechanical Engineers (ASME), Nov. 01, 2019. doi: 10.1115/1.4043587.

[26] A. Batabyal, J. Zhang, S. Yang, X. Du, and J. Chen, 'Applying Machine Learning to Optimize Sintered Powder Microstructures from Phase Field Modeling the Purdue University Graduate School Statement Of Committee Approval', 2020.

[27] 'Random Search and Grid Search for Function Optimization - MachineLearningMastery.com'. Accessed: Oct. 15, 2023. [Online]. Available: https://machinelearningmastery.com/random-search-and-grid-search-for-function-optimization/

[28] 'Full Factorial Experiment - Sixsigma DSI'. Accessed: Oct. 15, 2023. [Online]. Available: https://sixsigmadsi.com/glossary/full-factorial-experiment/

[29] R. L. Iman, J. M. Davenport, and D. K. Zeigler, *Latin hypercube sampling (program user's guide)*. 1980.

[30] K. Q. Ye, 'Orthogonal Column Latin Hypercubes and Their Application in Computer Experiments', 1998.

[31] P. Del Moral, A. Doucet, and A. Jasra, 'Sequential Monte Carlo samplers', *Journal of the Royal Statistical Society, Series B*, vol. 68, no. 3, pp. 411–436, Jun. 2006, doi: 10.1111/j.1467-9868.2006.00553.x.

[32] H. Gould and J. Tobochnik, 'An introduction to computer simulation methods: applications to physical systems', 1988.

[33] D. P. Kroese, T. Brereton, T. Taimre, and Z. I. Botev, 'Why the Monte Carlo method is so important today', *WIREs Comput Stat*, vol. 6, no. 6, pp. 386–392, Nov. 2014, doi: 10.1002/wics.1314.

[34] T. Pengo, A. Muñoz Barrutia, and C. Ortiz Desolórzano, 'Halton sampling for autofocus', *J Microsc*, vol. 235, no. 1, pp. 50–58, Jul. 2009, doi: 10.1111/j.1365-2818.2009.03180.x.

[35] T.-T. Wong, W.-S. Luk, and P.-A. Heng, 'Sampling with Hammersley and Halton Points', *Graphics Tools---The jgt Editors' Choice*, pp. 255–270, 2005, doi: 10.1201/b10628-32.

[36] 'What Is Design of Experiments (DOE)? | ASQ'. Accessed: Oct. 15, 2023. [Online]. Available: https://asq.org/quality-resources/design-of-experiments

[37] R. V. Rao, V. J. Savsani, and D. P. Vakharia, 'Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems', *Computer-Aided Design*, vol. 43, no. 3, pp. 303–315, Mar. 2011, doi: 10.1016/J.CAD.2010.12.015.

[38] 'Synthetic Data Generation: Definition, Types, Techniques, & Tools'. Accessed: Oct. 15, 2023. [Online]. Available: https://www.turing.com/kb/synthetic-data-generation-techniques

[39] A. S. Ruichen Jin, Wei Chen, 'An efficient algorithm for constructing optimal design of computer experiments', *J Stat Plan Inference*, vol. 134, no. 1, pp. 268–287, 2005.

[40] M. Awad and R. Khanna, ‘Efficient learning machines: Theories, concepts, and applications for engineers and system designers’, *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*, no. July 2018, pp. 1–248, 2015, doi: 10.1007/978-1-4302-5990-9.

[41] G. James, D. Witten, T. Hastie, and R. Tibshirani, ‘An Introduction to Statistical Learning’, *Springer Texts in Statistics*, 2017, doi: DOI 10.1007/978-1-4614-7138-7.

[42] ‘What is Random Forest? | IBM’. Accessed: Oct. 15, 2023. [Online]. Available: https://www.ibm.com/topics/random-forest

[43] H. Belyadi and A. Haghighat, ‘Supervised learning’, *Machine Learning Guide for Oil and Gas Using Python*, pp. 169–295, 2021, doi: 10.1016/B978-0-12-821929-4.00004-4.

[44] ‘Classification (Machine Learning) - an overview | ScienceDirect Topics’. Accessed: Oct. 15, 2023. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/classification-machine-learning

[45] Fabian Pedregosa and Ga Varoquaux and Alexandre Gramfort and Vincent Michel and Bertrand Thirion and Olivier Grisel and Mathieu, ‘Scikit-learn: Machine Learning in Python’, *ArXiv*, vol. abs/1201.0, 2011, doi: 10659969.

[46] R. Rumelhart, D., Hinton, G. & Williams, ‘Learning representations by back-propagating errors.’, *Nature*, vol. 323, pp. 533–536, 1986, doi: https://doi.org/10.1038/323533a0.

[47] B. Harish, K. Eswara Sai Kumar, and B. Srinivasan, ‘Topology optimization using convolutional neural network’, *Lecture Notes in Mechanical Engineering*, pp. 301–307, 2020, doi: 10.1007/978-981-15-5432-2_26.

[48] M. Abadi *et al.*, ‘TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems’, 2016, [Online]. Available: http://arxiv.org/abs/1603.04467

[49] M. F. Wei Chen, ‘Beyond the Known: Detecting Novel Feasible Domains Over an Unbounded Design Space’, *Journal of Mechanical Design*, vol. 139, no. 11, p. 10, 2017.

Appendix A: Codes to Generate Samples for Datasets

The following code demonstrate sample design points generations for belt pulley design problem using Latin hypercube sampling method. The generated design points can be saved to excel file for performing design of experiments.

In [1]:
```
import numpy as np
import pandas as pd
import openpyxl as xls
from scipy.stats import qmc
```

In [2]:
```
#Generating sample design variable points using latin hypercube sampling
```

In [3]:
```

bp=pd.DataFrame(qmc.LatinHypercube(d=3).random(n=100000))
```

In [5]:
```
# To scale the variable values to the required interval
```

In [6]:
```
l_bounds = [15, 70, 4]
u_bounds = [25, 80, 10]
qmc.scale(bp, l_bounds, u_bounds)

dfbp = pd.DataFrame(qmc.scale(bp, l_bounds, u_bounds))
```

In [8]:
```
# Writing the file to excel
```

In [67]:
```
Writer=pd.ExcelWriter(r'C:\Users\TSAT\Documents\Thesis Project\BELT PULLEY
                      DRIVE DESIGN PROBLEM\BP Sampling May 12, 2023V1.xlsx')
dfbp.to_excel(Writer,sheet_name='BP Sampling May 12, 2023V1', index=False)
Writer.save()
```

Appendix B: Datasets

Large, medium and small size dataset for binary classification of belt pulley design problem is shown in the following tables. Datasets for pressure vessel and helical spring problems both for binary and multiclass classification were prepared in similar manner as the belt pulley design problem.

Table 12 Large size dataset for belt pulley drive design problem

	x1	x2	x3	Label
0	15.557230	73.514507	4.144532	1
1	15.783434	70.592839	4.084521	1
2	15.592407	73.182853	4.291350	1
3	15.601598	75.430023	4.078617	1
4	15.872140	72.318496	4.035466	1
...	...	...	...	...
10361	16.652369	75.507598	4.230870	0
10362	16.473157	70.818085	4.086317	0
10363	17.128516	74.134537	4.030305	0
10364	17.168410	76.377176	4.000444	0
10365	17.098803	74.758031	4.105047	0

10366 rows × 4 columns

Table 13 Medium size dataset for belt pulley drive design problem

	x1	x2	x3	Label
0	15.815624	73.724115	4.098931	1
1	15.348572	72.161657	4.025556	1
2	15.075459	73.094843	4.371701	1
3	15.342714	76.279884	4.022654	1
4	15.170918	74.926337	4.224515	1
...	...	...	...	...
1032	15.838415	70.670424	4.266198	1
1033	16.280269	78.120632	4.021188	0
1034	16.034494	76.652975	4.403707	0
1035	16.561232	71.752582	4.072623	0
1036	15.890867	78.452832	4.018136	1

1037 rows × 4 columns

Table 14 Small size dataset for belt pulley drive design problem

	x1	x2	x3	Label
0	16.898352	73.353994	4.236584	0
1	15.347093	79.271101	4.525893	0
2	16.472045	70.021470	4.503005	0
3	15.727388	71.881256	4.213143	1
4	15.197884	71.823255	4.074252	1
...	...	...	...	...
202	15.889855	70.241168	4.230992	1
203	15.846875	79.443027	4.364554	0
204	16.170232	75.085385	4.035283	0
205	15.508060	70.434770	4.235914	1
206	15.332882	76.070635	4.191817	1

207 rows × 4 columns

Appendix C: Implementation Codes to Train and Evaluate Machine Learning Algorithms

The following code shows the binary class implementation of machine learning models on Python module scikit learn starting from preprocessing to model evaluation for the belt pulley drive design problem.

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import learning_curve
from sklearn import svm
from sklearn import metrics
import seaborn as sns;sns.set(font_scale=1.2)
%matplotlib inline
```

In [ ]:
```
bpdata = pd.read_csv('BPDataset BC May13,2023.csv')
```

In [ ]:
```
bpdata
```

In [ ]:
```
#Shuffling
```

In [ ]:
```
np.random.seed(42)  # uncomment this line to get the same shuffle each time
```

In [ ]:
```
bpdata = bpdata.reindex(np.random.permutation(bpdata.index))
```

In [ ]:
```
bpdata.reset_index(inplace = True, drop = True)
```

In [ ]:
```
bpdata
```

In [ ]:
```
data = bpdata.drop(columns='Label')
```

In [ ]:
```
data
```

In [ ]:
```
target=bpdata['Label']
```

In [ ]:
```
target
```

Feature scaling

In [ ]:
```
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
standard = StandardScaler()
```

In [ ]:
```
scaled_data = standard.fit_transform(data)
```

In [ ]:
```
scaled_data
```

In [ ]:
```
# Principal component analysis
from sklearn.decomposition import PCA
```

In [ ]:
```
pca_2c= PCA(n_components = 2)
X_pca = pca_2c.fit_transform(scaled_data)
```

In [ ]:
```
X_pca.shape
```

In [ ]:
```
X_pca
```

In [ ]:
```
pca_2c.explained_variance_ratio_
```

In [ ]:
```
pca_2c.explained_variance_ratio_.sum()
```

In [ ]:
```

```

In [ ]:
```
# Scatter plot
import seaborn as sns
sns.scatterplot(X_pca[:,0], X_pca[:,1], hue = bpdata['Label'])
```

In [ ]:
```
# TSNE plot
from sklearn.manifold import TSNE
tsne = TSNE(n_components = 2, init = "pca",learning_rate = "auto", random_state=123)
```

In [ ]:
```
bp_tsne =tsne.fit_transform(scaled_data)
```

In [ ]:
```
import numpy as np
```

```
import matplotlib.pyplot as plt
#f=plt.figure(figuresize=(8,8))
ax=plt.subplot(aspect = "equal")
for i in range(10):
    plt.scatter(bp_tsne[target ==i,0], bp_tsne[target ==i,1])
```

In [ ]:
```
# Parallel coordinates
import matplotlib.pyplot as plt
from pandas.plotting import parallel_coordinates
```

In [ ]:
```
# Make a plot
parallel_coordinates(bpdata, 'Label')
```

In [ ]:
```
# Show the plot
plt.show()
```

In [ ]:
```

```

In [ ]:
```
# Separating the data into training and testing
```

In [ ]:
```
X_train, X_test, Y_train, Y_test=train_test_split(scaled_data, target, test_size=0.3, random_state=0)
```

support vector machine with default parameters

```
In [ ]: #Generating the model
```

```
In [ ]: cls=svm.SVC(kernel='rbf', C=1.0, gamma='auto')
```

```
In [ ]: #Train the model
```

```
In [ ]: cls.fit(X_train,Y_train)
```

```
In [ ]: #Predict the response
```

```
In [ ]: pred_svm=cls.predict(X_test)
```

```
In [ ]:  print("Accuracy:", metrics.accuracy_score(Y_test, pred_svm))
```

```
In [ ]: #Precision score
```

```
In [ ]: print("Precision:",metrics.precision_score(Y_test, pred_svm))
```

```
In [ ]: #Recall score
```

```
In [ ]: print("Recall:",metrics.recall_score(Y_test,pred_svm))
```

```
In [ ]: print("F1 score:",metrics.f1_score(Y_test,pred_svm))
```

```
In [ ]: print(metrics.classification_report(Y_test,pred_svm))
```

Support vector machine hyperparameter tuning

```
In [ ]: # Hyperparameter tuning using optuna
import optuna
```

```
In [ ]:
def objective(trial):
    #define hyperparameter to search
    C = trial.suggest_uniform('C', 0.01, 100)
    gamma = trial.suggest_uniform('gamma', 0.001, 10)
    # Generating SVM model
    clst= svm.SVC(kernel='rbf', C=C, gamma=gamma)
    # Train the model
    clst.fit(X_train,Y_train)
    # Make prediction on the test set and calculate accuracy
    pred_svmt=clst.predict(X_test)
    accuracy = accuracy_score(Y_test, pred_svmt)
    return accuracy
```

```
In [ ]: # Run hyperparameter optimization with optuna
study=optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
```

In [ ]:
```
#train svm model with the best hyperparameters obtained by optuna
best_params=study.best_params
clst=svm.SVC(kernel='rbf',C=best_params['C'],gamma=best_params['gamma'])
clst.fit(X_train,Y_train)
```

In [ ]:
```
#make prediction on the test set and calculate accuracy
pred_svmt=clst.predict(X_test)
accuracy=accuracy_score(Y_test,pred_svmt)
```

In [ ]:
```
#print the best hyperparameters and accuracy found
print('Best hyperparameters:',best_params)
print('Accuracy:',accuracy)
```

In [ ]:
```
print("Precision:",metrics.precision_score(Y_test, pred_svmt))
```

In [ ]:
```
print("Recall:",metrics.recall_score(Y_test,pred_svmt))
```

In [ ]:
```
print("F1 score:",metrics.f1_score(Y_test,pred_svmt))
```

In [ ]:
```
print(metrics.classification_report(Y_test,pred_svmt))
```

In [ ]:
```
# ROC and AUC
```

In [ ]:
```
import matplotlib.pyplot as plt
from sklearn.svm import SVC
model_svc=SVC(kernel='rbf',random_state=4)
model_svc.fit(X_train, Y_train)
y_pred_svm=model_svc.predict(X_test)
```

In [ ]:
```
# Plot ROC and compare AUC
from sklearn.metrics import roc_curve, auc
svm_fpr, svm_tpr,threshold=roc_curve(Y_test, y_pred_svm)
auc_svm=auc(svm_fpr, svm_tpr)
#plt.figure(figuresize=(5,5),dpi=100)
plt.plot(svm_fpr, svm_tpr, linestyle='-', label='svm(auc=%0.3f)'% auc_svm)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```

Random forest with default parameters

In [ ]:
```
rf = RandomForestClassifier()
```

In [ ]:
```
rf.fit(X_train, Y_train)
rf.score(X_test, Y_test)
```

In [ ]:
```
pred_rf=rf.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import confusion_matrix
```

In [ ]:
```
cf_matrix = confusion_matrix(Y_test, pred_rf)
```

In [ ]:
```
print(cf_matrix)
```

In [ ]:
```
import seaborn as sns
#labels = ['Shell radius','Shell length', 'Shell thickness','Head thickness']
#fig=plt.figure(figuresize=(10,10))
sns.heatmap(cf_matrix, annot=True)
ax.set_title('Seaborn Confusion Matrix with Labels\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values')
```

In [ ]:
```

```

In [ ]:
```
print("Precision:",metrics.precision_score(Y_test, pred_rf))
```

In [ ]:
```
print("Recall:",metrics.recall_score(Y_test,pred_rf))
```

In [ ]:
```
print("F1 score:",metrics.recall_score(Y_test,pred_rf))
```

In [ ]:
```
print(metrics.classification_report(Y_test,pred_rf))
```

Random forest with hyperparameter tuning

In [ ]:
```
import optuna
from optuna.samplers import TPESampler
from sklearn.model_selection import cross_val_score

def objective(trial):
    # search space
    n_estimators =  trial.suggest_int('n_estimators', low=100, high=600, step=50)
    criterion = trial.suggest_categorical('criterion', ['gini', 'entropy'])
    min_samples_split = trial.suggest_int('min_samples_split', low=2, high=4, step=1)
    min_samples_leaf = trial.suggest_int('min_samples_leaf', low=1, high=5, step=1)
    max_depth = trial.suggest_int('max_depth', low=10, high=50, step=5)
    max_features = trial.suggest_categorical('max_features', ['auto', 'sqrt','log2'])

    # random forest classifier object
    rfc = RandomForestClassifier(n_estimators=n_estimators,
                                 criterion=criterion,
                                 min_samples_split=min_samples_split,
                                 min_samples_leaf=min_samples_leaf,
                                 max_depth=max_depth,
                                 max_features=max_features,
                                 random_state=42)
    score =  cross_val_score(estimator=rfc,
                             X=X_train,
                             y=Y_train,
                             scoring='accuracy',
```

```
                                cv=5,
                                n_jobs=-1).mean()

    return score
```

In [ ]:
```
# create a study (aim to maximize score)
study = optuna.create_study(sampler=TPESampler(), direction='maximize')
```

In [ ]:
```
# perform hyperparamter tuning
study.optimize(objective, n_trials=100)
```

In [ ]:
```
# store result in a data frame
values_bayesian = study.best_trial.value
```

In [ ]:
```
values_bayesian
```

In [ ]:
```
trial=study.best_trial
```

In [ ]:
```
print('Best hyperparameters {}'.format(trial.params))
```

In [ ]:
```
rft = RandomForestClassifier(n_estimators=450, max_depth=35)
rft.fit(X_train, Y_train)
rft.score(X_test, Y_test)
```

In [ ]:
```
pred_rft=rf.predict(X_test)
```

In [ ]:
```
print("Accuracy:",metrics.accuracy_score(Y_test, pred_rft))
```

In [ ]:
```
print("Precision:",metrics.precision_score(Y_test, pred_rft))
```

In [ ]:
```
print("Recall:",metrics.recall_score(Y_test,pred_rft))
```

In [ ]:
```
print("F1 score:",metrics.recall_score(Y_test,pred_rf))
```

Using Naive Bayes GaussianNB

In [ ]:
```
clf = GaussianNB()
clf.fit(X_train,Y_train)
clf.score(X_test, Y_test)
```

In [ ]:
```
pred_nb = clf.predict(X_test)
```

In [ ]:
```
print("Accuracy:",metrics.accuracy_score(Y_test, pred_nb))
```

In [ ]:
```
print("Precision:",metrics.precision_score(Y_test, pred_nb))
```

In [ ]:
```
print("Recall:",metrics.recall_score(Y_test,pred_nb))
```

In [ ]:
```
print("F1 score:",metrics.f1_score(Y_test,pred_nb))
```

In [ ]:
```
print(classification_report(Y_test, pred_nb))
```

Convergence plot with default hyperparameters

In [ ]:
```
train_sizes, train_scores, test_scores = learning_curve(svm.SVC(), scaled_data,
target, cv=10, scoring ='accuracy', n_jobs=-1, train_sizes=np.linspace(0.01, 1, 50), verbose=1)
```

In [ ]:
```
train_mean = np.mean(train_scores, axis=1)
```

In [ ]:
```
#train_mean
```

In [ ]:
```
train_std = np.std(train_scores, axis=1)
```

In [ ]:
```
#train_std
```

In [ ]:
```
test_mean = np.mean(test_scores, axis=1)
```

In [ ]:
```
#test_mean
```

In [ ]:
```
test_std = np.std(test_scores, axis=1)
```

In [ ]:
```
#test_std
```

In [ ]:
```
train_sizes, train_scores, test_scores = learning_curve(RandomForestClassifier(),
scaled_data, target, cv=10, scoring ='accuracy', n_jobs=-1, train_sizes=np.linspace(0.01, 1, 50), verbose=1)
```

In [ ]:
```
test_mean1 = np.mean(train_scores, axis=1)
```

In [ ]:
```
test_mean1
```

In [ ]:
```
train_sizes, train_scores, test_scores = learning_curve(GaussianNB(),
scaled_data, target, cv=10, scoring ='accuracy', n_jobs=-1, train_sizes=np.linspace(0.01, 1, 50), verbose=1)
```

In [ ]:
```
test_mean2 = np.mean(test_scores, axis=1)
```

In [ ]:
```
#test_mean2
```

In [ ]:
```
import matplotlib.pyplot as plt
```

In [ ]:
```
plt.plot(train_sizes, test_mean, label ='SVM')
plt.plot(train_sizes, test_mean1,label = 'RF')
plt.plot(train_sizes, test_mean2,label = 'GNB')
plt.title('BPD Convergence Plot')
plt.xlabel('Training Sizes')
plt.ylabel('Accuracy Score')
plt.legend(loc = 'best')
```

Convergence plot with tuned hyperparameters

In [ ]:
```
train_sizes, train_scores, test_scores = learning_curve(svm.SVC(kernel='rbf',C=12,
    gamma =5.93), scaled_data, target, cv=10, scoring ='accuracy', n_jobs=-1,
    train_sizes=np.linspace(0.01, 1, 50), verbose=1)
```

In [ ]:
```
ttrain_mean = np.mean(train_scores, axis=1)
```

In [ ]:
```
#ttrain_mean
```

In [ ]:
```
ttest_mean = np.mean(test_scores, axis=1)
```

In [ ]:
```
#ttest_mean
```

In [ ]:
```
ttrain_std = np.std(train_scores, axis=1)
```

In [ ]:
```
#ttrain_std
```

In [ ]:
```
train_sizes, train_scores, test_scores = learning_curve(RandomForestClassifier
(n_estimators=350, max_depth=50), scaled_data, target, cv=10, scoring ='accuracy', n_jobs=-1,
train_sizes=np.linspace(0.01, 1, 50), verbose=1)
```

In [ ]:
```
ttrain_mean1 = np.mean(train_scores, axis=1)
```

In [ ]:
```
#ttrain_mean1
```

In [ ]:
```
ttest_mean1 = np.mean(test_scores, axis=1)
```

In [ ]:
```
#ttest_mean1
```

In [ ]:
```
ttrain_std1 = np.std(train_scores, axis=1)
```

In [ ]:
```
#ttrain_std1
```

In [ ]:
```

```

In [ ]:
```
plt.plot(train_sizes, ttest_mean, label ='SVM')
plt.plot(train sizes, ttest mean1,label = 'RF')
```

```
plt.plot(train_sizes, ttest_mean1,label = 'RF')
plt.title('BPD Convergence Plot')
plt.xlabel('Training Sizes')
plt.ylabel('Accuracy Score')
plt.legend(loc = 'best')
```

Appendix D: Algorithm Performance Resusts

The following tables show the performance of machine learning algorithms on the design problem at the three dataset sizes for both binary and multiclass classification.

A) Binary Classification

Table 15 Binary classification performance of different algorithms with untuned parameters on large dataset size of the design problems

Algorithm	Design problem											
	Pressure vessel				Helical coiled spring				Belt pulley drive			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
SVM	1.0	1.0	1.0	1.0	0.973	0.949	1.0	0.993	0.994	0.997	0.992	0.995
NN	1.0	1.0	1.0	1.0	0.992	0.986	1.0	0.992	0.997	0.998	0.996	0.997
RF	1.0	1.0	1.0	1.0	0.973	0.949	1.0	0.973	0.996	0.998	0.993	0.993
GNB	1.0	1.0	1.0	1.0	0.914	0.877	0.965	0.918	0.993	0.993	0.923	0.957

Table 16 Binary classification performance of different algorithms with tuned parameters on large dataset size of the design problems

Algorithm	Design problem											
	Pressure vessel				Helical coiled spring				Belt pulley drive			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
SVM	0.9987	0.997	1	0.9987	0.993	0.986	0.999	0.993	0.999	1.0	0.999	0.999
NN	1.0	1.0	1.0	1.0	0.989	0.993	0.986	0.989	0.999	0.999	1.0	0.999
RF	1.0	1.0	1.0	1.0	0.973	0.949	1.0	0.973	0.996	0.998	0.993	0.993

Table 17 Binary performance of different algorithm with untuned parameters on medium dataset on the design problems

Algorithm	Design problem											
	Pressure vessel				Helical coiled spring				Belt pulley drive			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
SVM	1.0	1.0	1.0	1.0	0.963	0.932	1.0	0.965	0.990	0.987	0.993	0.990
NN	1.0	1.0	1.0	1.0	0.966	0.938	1.0	0.968	0.990	0987	0.993	0.990
RF	1.0	1.0	1.0	1.0	0.963	0.932	1.0	0.965	0.990	0.987	0.987	0.987
GNB	1.0	1.0	1.0	1.0	0.887	0.847	0.947	0.895	0.964	0.993	0.939	0.965

Table 18 Binary performance of different algorithm with tuned parameters on medium dataset size of the design problems

Algorithm	Design problem											
	Pressure vessel				Helical coiled spring				Belt pulley drive			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
SVM	1.0	1.0	1.0	1.0	0.990	0.980	1.0	0.990	1.0	1.0	1.0	1.0
NN	1.0	1.0	1.0	1.0	0.990	0.980	1.0	0.990	1.0	1.0	1.0	1.0
RF	1.0	1.0	1.0	1.0	0.963	0.932	1.0	0.965	0.990	0.993	0.987	0.987

Table 19 Binary performance of different algorithm with untuned parameters on small dataset size of the design problems

Algorithm	Design problem											
	Pressure vessel				Helical coiled spring				Belt pulley drive			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
SVM	1.0	1.0	1.0	1.0	0.918	0.864	1.0	0.973	1.0	1.0	1.0	1.0
NN	1.0	1.0	1.0	1.0	0.868	0.80	1.0	0.888	1.0	1.0	1.0	1.0
RF	1.0	1.0	1.0	1.0	0.918	0.864	0.927	0.927	1.0	1.0	1.0	1.0
GNB	1.0	1.0	1.0	1.0	0.868	0.875	0.875	0.875	1.0	1.0	1.0	1.0

Table 20 Binary performance of different algorithm with tuned parameters on small dataset size of the design problems

Algorithm	Design problem											
	Pressure vessel				Helical coiled spring				Belt pulley drive			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
SVM	1.0	1.0	1.0	1.0	0.983	0.969	1.0	0.984	1.0	1.0	1.0	1.0
NN	1.0	1.0	1.0	1.0	0.983	0.969	1.0	0.984	1.0	1.0	1.0	1.0
RF	1.0	1.0	1.0	1.0	0.918	0.864	1.0	0.927	1.0	1.0	1.0	1.0

B) Multiclass Classification

Table 21 Multiclass classification performance of different algorithm with untuned parameters on large dataset size of the design problems

Algorithm	Design problem											
	Pressure vessel				Helical coiled spring				Belt pulley drive			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
SVM	0.771	0.771	0.771	0.771	0.969	0.969	0.969	0.969	0.926	0.926	0.926	0.926
NN	0.915	0.915	0.915	0.915	0.993	0.993	0.993	0.993	0.989	0.989	0.989	0.989
RF	0.983	0.983	0.983	0.983	0.982	0.982	0.982	0.982	0.918	0.918	0.921	0.918
GNB	0.849	0.849	0.849	0.849	0849	0849	0849	0849	0.757	0.757	0.757	0.757

Table 22 Multiclass classification performance of different algorithm with tuned parameters on large dataset size of the design problems

Algorithm	Design problem											
	Pressure vessel				Helical coiled spring				Belt pulley drive			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
SVM	0.979	0.979	0.979	0.979	0.991	0.991	0.991	0.991	0.987	0.987	0.987	0.987
NN	0.960	0.960	0.960	0.960	0.993	0.993	0.993	0.993	0.990	0.990	0.990	0.990
RF	0.983	0.983	0.983	0.983	0.969	0.969	0.969	0.969	0.918	0.918	0.918	0.918

Table 23 Multiclass classification performance of different algorithm with untuned parameters on medium dataset size of the design problems

Algorithm	Design problem											
	Pressure vessel				Helical coiled spring				Belt pulley drive			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
SVM	0.752	0.752	0.752	0.752	0.933	0.933	0.933	0.933	0.926	0.926	0.926	0.926
NN	0.915	0.915	0.915	0.915	0.993	0.993	0.993	0.993	0.989	0.989	0.989	0.989
RF	0.950	0.950	0.950	0.950	0.957	0.957	0.957	0.957	0.918	0.918	0.918	0.918
GNB	0.864	0.864	0.864	0.864	0.867	0.867	0.867	0.867	0.757	0.757	0.757	0.757

Table 24 Multiclass classification performance of different algorithm with tuned parameters on medium dataset size of the design problems

Algorithm	Design problem											
	Pressure vessel				Helical coiled spring				Belt pulley drive			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
SVM	0.924	0.924	0.924	0.924	0.980	0.980	0.980	0.980	0.987	0.987	0.987	0.987
NN	0.960	0.960	0.960	0.960	0.993	0.993	0.993	0.993	0.987	0.987	0.987	0.987
RF	0.950	0.950	0.950	0.950	0.933	0.933	0.933	0.933	0.918	0.918	0.918	0.918

Table 25 Multiclass classification performance of different algorithm with untuned parameters on small dataset size of the design problems

Algorithm	Design problem											
	Pressure vessel				Helical coiled spring				Belt pulley drive			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
SVM	0.890	0.890	0.890	0.890	0.819	0.819	0.819	0.819	0.857	0.857	0.857	0.857
NN	0.593	0.593	0.593	0.593	0.803	0.803	0.803	0.803	0.857	0.857	0.857	0.857
RF	0.875	0.875	0.875	0.875	0.885	0.885	0.885	0.885	0.825	0.825	0.825	0.825
GNB	0.921	0.921	0.921	0.921	0.786	0.786	0.786	0.786	0.825	0.825	0.825	0.825

Table 26 Multiclass classification performance of different algorithm with tuned parameters on small dataset size of the design problems

Algorithm	Design problem											
	Pressure vessel				Helical coiled spring				Belt pulley drive			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
SVM	0.812	0.809	0.812	0.801	0.901	0.901	0.901	0.901	0.888	0.888	0.888	0.888
	0.921	0.921	0.921	0.921	0.934	0.934	0.934	0.934	0.904	0.904	0.904	0.904
RF	0.875	0.875	0.875	0.875	0.819	0.819	0.819	0.819	0.825	0.825	0.825	0.825