Master Thesis

Using ML to Model and Optimize Chip Geometry for Improved Lithography

Junaid Ahmad

2023

Master Thesis

Deggendorf Institute of Technology

Faculty of Computer Science

The University of South Bohemia in České Budějovice

Faculty of Science

MSc. Artificial Intelligence and Data Science

Using ML to Model and Optimize Chip Geometry for Improved Lithography

Master thesis to obtain the academic degree:

Master of Science (MSc.)

Submitted By:	Junaid Ahmad
Matriculation Number:	22107474
First Examiner:	Prof. Dr. Cezar Ionescu
Supervisors:	Prof. Dr. Georg Pelz
Submission Date:	August 10, 2023

Erklärung

Name des Studierenden: Junaid Ahmad

Name des Betreuenden: Prof. Dr. Cezar Ionescu

Thema der Abschlussarbeit:

Verwendung von ML zur Modellierung und Optimierung der Chip-Geometrie für eine verbesserte Lithografie

1. Ich erkläre hiermit, dass ich die Abschlussarbeit gemäß § 35 Abs. 7 RaPO (Rahmenprüfungsordnung für die Fachhochschulen in Bayern, BayRS 2210-4-1-4-1-WFK) selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

   Deggendorf, 10.08.2023 ....

   Datum Unterschrift des Studierenden

2. Ich bin damit einverstanden, dass die von mir angefertigte Abschlussarbeit über die Bibliothek der Hochschule einer breiteren Öffentlichkeit zugänglich gemacht wird:

   ○ Nein

   ✓ Ja, nach Abschluss des Prüfungsverfahrens

   ○ Ja, nach Ablauf einer Sperrfrist von ... Jahren.

   Deggendorf, 10.08.2023 ....

   Datum Unterschrift des Studierenden

Bei Einverständnis des Verfassenden vom Betreuenden auszufüllen:

Eine Aufnahme eines Exemplars der Abschlussarbeit in den Bestand der Bibliothek und die Ausleihe des Exemplars wird:

Acknowledgements

I would like to thank Prof. Dr. Georg Pelz at Infineon AG, and Prof. Dr. Cezar Ionescu for their guidance, support, and feedback throughout my thesis. Their expertise and dedication were key to the successful completion of my project. I am grateful for their mentorship and honored to have worked with them.

I would like to express my deepest appreciation and love to my parents and siblings for their unwavering support and encouragement throughout my academic journey. Their love and support have been a constant source of motivation.

Thank you all for your invaluable support, encouragement and assistance.

Annotation

Ahmad, J., 2023: “Using ML to Model and Optimize Chip Geometry for Improved Lithography” Mgr. Thesis, in English. -51 p., Faculty of Applied Computer Science, Deggendorf Institute of Technology, Deggendorf, Germany, Faculty of Science Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Annotation

This thesis delves into the creation and application of a predictive model aimed at optimizing chip production on a wafer, while maintaining the on-resistance (Ron) of a MOSFET within acceptable limits. Through a systematic approach, various regression models were developed, including linear regression, Random Forest, XGBoost, and a Deep Neural Network (DNN), to predict chip quantities considering both wafer and chip geometry. Model performance was rigorously evaluated using mean absolute error, with a focus on comparing machine learning models to a geometry-based predictor. The DNN demonstrated superior accuracy and was integrated into an optimization algorithm that managed the balance between chip quantity and Ron value. This algorithm employed Differential Evolution to identify the optimal chip layout, expanding its scope by considering reticle-based scenarios. This work contributes valuable insights into semiconductor manufacturing and chip layout optimization, offering a method to enhance wafer productivity, efficiency,

and cost-effectiveness.

I declare that I am the author of this qualification thesis and that in writing it I have used the sources and literature displayed in the list of used sources only.

In **Deggendorf** on, 31.08.2023 ....

Student's Signature

Abstract

This thesis explores the development and application of a predictive model for optimizing the number of chips produced on a wafer while keeping the on-resistance (R_{on}) of a MOSFET within an acceptable range of an existing MOSFET type. A systematic approach has been employed, starting with the creation and assessment of multiple regression models, including linear regression, Random Forest, XGBoost, and a Deep Neural Network (DNN). These models were built to predict the number of chips on a wafer considering both wafer and chip geometry.

The models' performance was meticulously evaluated using mean absolute error as the performance metric. Furthermore, a comparative analysis was conducted between the machine learning models and a geometry-based predictor crafted using geometry rules. The DNN emerged as the superior model based on its accuracy.

The selected DNN was integrated into an optimization algorithm to manage the critical trade-off between the number of chips and the R_{on} value - a key performance indicator for semiconductor devices. The optimization algorithm harnessed the power of the Differential Evolution technique to explore the solution space and identify the optimal chip layout.

Additionally, the optimization algorithm was also tested by incorporating the formula for the number of chips on a reticle, a rectangular mask featuring chip designs, thus expanding the scope of potential scenarios under consideration. This thesis offers valuable insights into the semiconductor manufacturing process, particularly concerning chip layout optimization. Moreover, it introduces a methodology for maximizing wafer productivity, which has implications for efficiency and cost-effectiveness in semiconductor manufacturing.

Contents

Glossary

AI Artificial Intelligence. 1, 2

DE Differential Evolution. 36, 37, 45, 46

DNNs Deep neural networks. x, 3, 15, 29, 31, 34, 42, 44

EA Evolutionary Algorithms. 24, 25

MAE Mean Absolute Error. 23

ML Machine learning. 12

MSE Mean Squared Error. 23

ReLU Rectified linear unit. 17, 18

RUF Reticle utilization factor. 29, 30

XGBoost Extreme Gradient Boosting. viii–x, 3, 15, 29, 33, 42, 44, 47

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

This chapter outlines the considerations surrounding thesis selection, alongside the associated challenges and limitations. The first section delves into the driving factors behind the problem and its potential ramifications. Subsequent sections delineate specific research questions this thesis seeks to address. After presenting a broad overview of the adopted approach and methodology, the chapter concludes by highlighting the study's constraints and limitations.

1.1 Motivation

The semiconductor industry is projected to become a trillion-dollar industry by 2030[1]. However, this growth comes with its challenges, particularly the high cost of semiconductor equipment. The machinery required for semiconductor manufacturing is highly specialized and expensive, posing a significant barrier for many companies entering the market. To address this issue, artificial intelligence (AI) can be leveraged to optimize the utilization of available equipment. By implementing AI algorithms and advanced analytics, manufacturers can enhance production efficiency, reduce downtime, and improve overall equipment effectiveness. Additionally, the industry is currently grappling with an equipment shortage, further exacerbating the cost and supply challenges. AI can assist in managing this shortage by analyzing data in order to identify bottlenecks, optimize production schedules, and make informed decisions about equipment allocation, thereby maximizing output within existing resources.

Lithography is a critical step in semiconductor fabrication, responsible for transferring complex patterns onto a silicon wafer, which are then etched to form the device structures. Due to its complexity and precision requirements, it often becomes a significant bottleneck in the overall manufacturing process of semiconductors. Currently, lithography equipment is in short supply[2]. Also, these machines are quite expensive. The latest EUV lithography machine from ASML, for instance, carries a price tag of USD 100 million[3]. The scarcity of lithography machines poses a major challenge to the semiconductor industry, further emphasizing the need for efficient utilization of available equipment. By employing AI-driven optimization techniques, manufacturers can mitigate the impact of this shortage by maximizing the productivity and lifespan of existing machines. This approach ensures that the limited lithography machines are effectively utilized, enhancing production capacity and meeting the growing demand for semiconductors.

1.2 Research Topic and Questions

This thesis presents a novel contribution to the field by integrating machine learning techniques with optimization methodologies. By integrating machine learning algorithms with optimization techniques, the prediction and optimization of chip production in the lithography process can be enhanced. This optimization can be achieved by increasing the number of chips produced from a single wafer, thereby optimizing the manufacturing process.

Traditionally, the design of a MOSFET chip for a specific on-resistance (R_{on}) is executed without considering its impact on the total number of chips produced. This lack of consideration can lead to suboptimal production outcomes, thus highlighting the need for an approach that integrates R_{on} design with the maximization of chip production. This approach not only streamlines the optimization process but also opens doors for extending the application of machine learning to other complex manufacturing steps in the semiconductor industry. This thesis will answer these main questions:

1. How do chip dimensions influence the number of chips produced?

2. In this context, how do the different regression models stack up against each other in terms of their performance?

3. How can we optimize the number of chips on a wafer with minimal R_{on} impact?

1.3 Approach and Methodology

In the process of lithography, semiconductors chip designs are fitted in a reticle mask which is then exposed to a UV light to transfer the designs on a wafer. Predicting the number of chips in a single wafer can help with optimizing the designs. The production data from previously manufactured MOSFET types can be leveraged to model and comprehend the behavior associated with the number of chips.

Machine learning algorithms, such as linear regression, decision trees, XGBoost, and Deep Neural Networks (DNNs), can be employed to create a regression model. The developed machine learning model can be integrated into an optimization algorithm. This optimizer can then refine the chip geometry based on a specified R_{on} value and an associated tolerance, determining the acceptable variation in the case of a new R_{on}. By leveraging machine learning and optimization techniques, manufacturers can fine-tune the lithography process, ensuring precise and efficient chip dimensions, which can result in improved overall semiconductor manufacturing quality and quantity.

1.4 Scope and Limitations

Because of the broad nature of the thesis and the time constraints, it has been limited to an extent. Following are the limitations of this thesis:

- Only the chip sizes meeting the design requirements were selected.
- A select few types of MOSFET were used for the optimization section.

1.5 Outline

The next chapter will deal with the theoretical background of the research. In Chapter 2, basic theoretical background of lithography, machine learning and optimization is introduced. In Chapter 3, the relevant published literature is introduced and discussed. The methodology for this thesis will be discussed in Chapter 4. In Chapter 5, the results of the experiments will be presented and discussed. Finally, the thesis will be concluded in Chapter 6.

Chapter 2

Theoretical Background

This chapter examines the theoretical foundations of the research presented in the thesis, including a review of related theories, models, and conceptual frameworks. In particular, these research questions will be addressed:

- What are the factors that impact chip quantities in the process of lithography?
- What sort of regression algorithms are available for the modelling?
- Which optimization algorithms can be used to improve the production process?

The answers to these questions will help us gain more detailed understanding of the various works that are discussed in the literature review. Consequently, this will assist in comprehending the contemporary techniques and methodologies employed in the field. Section 2.2 will provide us with a basic understanding of the lithography process and its sub-processes, namely reticle design and wafer layout design. This background is necessary to better formulate and understand the problem. Section 4.3 will describe different regression techniques available in the machine learning toolbox. It will also discuss some of the metrics that are used to compare these models. Finally, optimization will be discussed along with the available techniques in Section 2.3.

2.1 Lithography

To fully comprehend the rationale behind the research conducted in this thesis, it is imperative to grasp the significance of the critical manufacturing process of lithography.

This understanding will serve as the foundation for subsequent steps in modeling and optimization.

Lithography, as a fundamental process in semiconductor manufacturing, plays a pivotal role in achieving the desired patterns and structures on a wafer. It involves the precise transfer of patterns from a mask or a reticle onto a photosensitive material, typically through the use of light or other forms of radiation. The accuracy and efficiency of the lithography process significantly impact the overall quantity, performance, and cost-effectiveness of semiconductor devices.

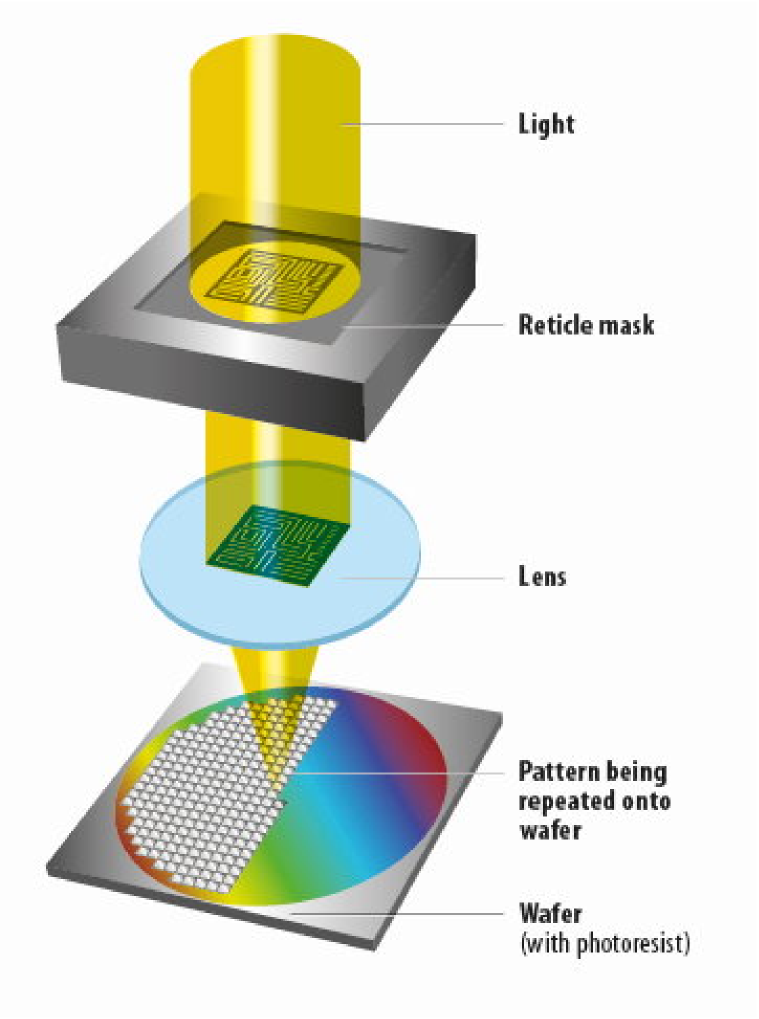

Figure 2.1: Lithography Overview [4]

An examination of the intricacies of lithography, encompassing its underlying principles, equipment, and techniques, sheds light on the challenges and complexities inherent in maximizing chip production. This understanding will highlight the need for modeling and optimization approaches to enhance the lithography process.

Furthermore, comprehending the critical manufacturing steps of lithography, such as reticle design and wafer layout design, will illuminate the key considerations and trade-offs faced by semiconductor manufacturers. The design and arrangement of the reticle and wafer layouts directly impact the number of chips that can be produced, as well as the desired electrical characteristics, such as R_{on}. Therefore, knowledge of these sub-processes will provide valuable insights into the opportunities for improvement and optimization.

By establishing an understanding of lithography's critical manufacturing processes, this research aims to contribute to the field by developing a number of chips prediction model and an optimizer. These components will enable more accurate number of chip predictions and effective optimization strategies, ultimately leading to enhanced chip quantities and improved efficiency in semiconductor manufacturing. The two important steps in lithography are:

- Reticle Design
- Wafer Floorplanning

2.1.1 Reticle Design

Reticles, also known as photomasks, are an integral part of the lithography process in semiconductor manufacturing. They serve as templates or stencils that contain the desired patterns to be transferred onto the wafer during the fabrication process. Reticles are typically made of glass or quartz substrates coated with a thin layer of chrome or other materials that can selectively block or transmit light.

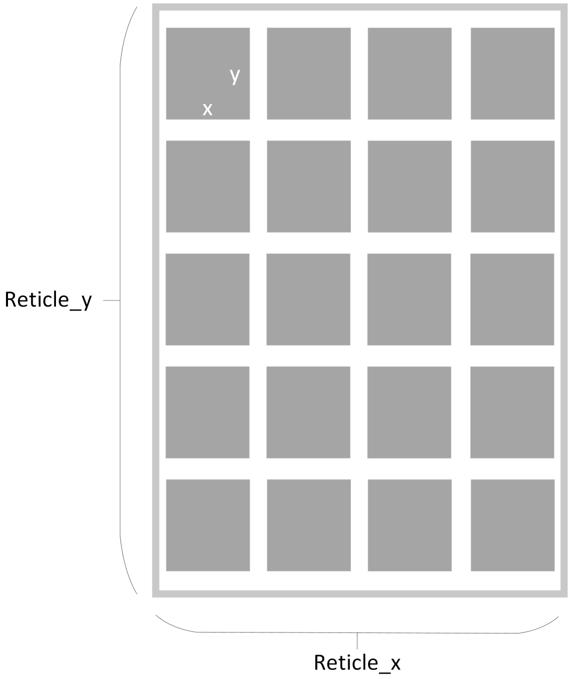

Figure 2.2: A single reticle

The planning and design of reticles involve several important considerations. First, reticle design encompasses the layout and arrangement of the patterns that need to be projected onto the wafer. It involves determining the location, size, and shape of the patterns, as well as any alignment marks or registration features. The reticle design process is typically carried out using computer-aided design (CAD) software, where designers create and manipulate the patterns with high precision.

Once the reticle design is finalized, reticle manufacturing takes place. This involves transferring the design onto the reticle substrate using processes such as photolithography and etching. The precise and accurate fabrication of reticles is crucial to ensure the fidelity of pattern transfer during lithography.

Reticle planning involves optimizing the use of reticles for efficient chip production. As multiple chips are typically patterned on a single wafer, reticle planning aims to maximize the number of chips produced while minimizing material waste and manufacturing costs. This involves strategically arranging the chip patterns on the reticle to minimize the space between them and maximize the use of available reticle area. The number of chips (n) of size (x, y) on a reticle with a width R_x and height R_y can be roughly calculated using the following formula:

$$\mathrm{n} = \frac{R_x}{x} \cdot \frac{R_y}{y} \tag{2.1}$$

In reticle planning, considerations such as chip size, design complexity, and alignment requirements come into play. The goal is to achieve the highest possible yield by minimizing defects, reducing overlay errors, and optimizing the use of reticles. Reticle planning also involves managing the life cycle of reticles, including their maintenance, cleaning, and replacement.

Overall, reticles and their planning are crucial elements in the lithography process. They enable the accurate transfer of patterns onto the wafer, contributing to the precise fabrication of semiconductor devices. Effective reticle design and planning are essential for optimizing chip production and ensuring efficient utilization of resources in semiconductor manufacturing.

2.1.2 Wafer Layout Design

Wafers are the substrate materials used in semiconductor manufacturing. They are typically circular, thin slices made of silicon or other semiconductor materials. Wafers provide the foundation on which integrated circuits and other semiconductor devices are fabricated.

Wafer layout and design involve the arrangement and organization of multiple chip patterns on a single wafer. The layout determines the position, size, and orientation of each chip on the wafer surface. Wafer design is crucial for maximizing chip production efficiency, optimizing yield, and minimizing costs.

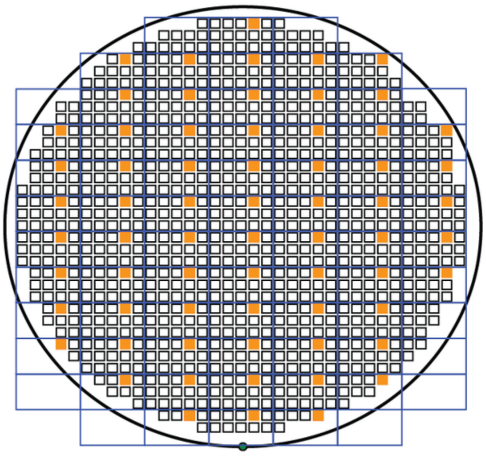

Figure 2.3: A wafer map [5]

The process of wafer layout begins with defining the size and shape of individual chips. This includes considering the dimensions and requirements of the intended devices.

The layout design also considers factors such as inter-chip spacing, alignment marks, and electrical connections.

One important aspect of wafer layout design is the consideration of process variations. Variations in the lithography process, such as exposure and etching, can lead to deviations in the final chip dimensions. Designers must account for these variations and incorporate sufficient margins (M) to ensure that the chips are not compromised during the fabrication process. This can be expressed using the following equation:

$$Dimenions_{(final)} = Dimenions_{(desired)} + M \tag{2.2}$$

Efficient utilization of the wafer area is a critical objective in wafer layout design. Maximizing the number of chips produced on a wafer is essential for achieving high yield and cost-effectiveness. Designers employ various optimization techniques, such as tiling and nesting, to efficiently pack the chip patterns on the wafer surface while minimizing unused areas.

In wafer design, considerations are also given to wafer-level tests and probing. Test structures, such as test chips or test pads, are strategically placed on the wafer to enable the testing and validation of the fabricated devices. These test structures provide crucial feedback on the performance and quality of the manufacturing process.

Furthermore, wafer design involves implementing design rules and guidelines that ensure manufacturability and compatibility with the lithography process. Designers need to adhere to specific constraints related to minimum feature sizes, spacing, and layer alignments to ensure successful fabrication.

In summary, wafer layout and design play a significant role in semiconductor manufacturing. Effective wafer layout design optimizes chip production, maximizes yield, and minimizes costs. It involves careful placement and arrangement of chip patterns on the wafer, considering process variations, design rules, and test requirements. By efficiently utilizing the wafer area and incorporating optimization techniques, designers can enhance the efficiency and effectiveness of the overall lithography process. The consideration of process variations and the inclusion of design margins (M) ensure that the final chip dimensions meet the desired specifications.

2.2 Machine Learning

For a comprehensive grasp of the regression models and other concepts presented in this thesis, a foundational knowledge of machine learning concepts is essential. The techniques described here will help form the basis for further analysis, discussion and subsequent conclusion. To generate predictions about unseen data, there is a need for algorithms which can inherently identify and mimic the patterns in the existing examples. Such algorithms are commonly associated with machine learning techniques. Machine learning is a sub-field of artificial intelligence, which is broadly defined as the capability of a machine to imitate intelligent human behavior[6]. The algorithms are designed and trained by a human expert who can tune and test these algorithms to make better predictions. The algorithm then learns the pattern through trial and error. It can be thought of as fitting an equation to a set of data points. One very clear example of a machine learning algorithm is stock price prediction. Using least squares linear regression we can predict stock prices[7]. Similar models can be used to predict inflation, GDP growth, production of vehicle parts and many other variables. Machine learning is broadly classified into four different approaches:

- Supervised learning
- Unsupervised learning
- Semi-supervised learning methods
- Reinforcement learning

The coming discussion will be limited to the supervised learning as it is relevant to the thesis.

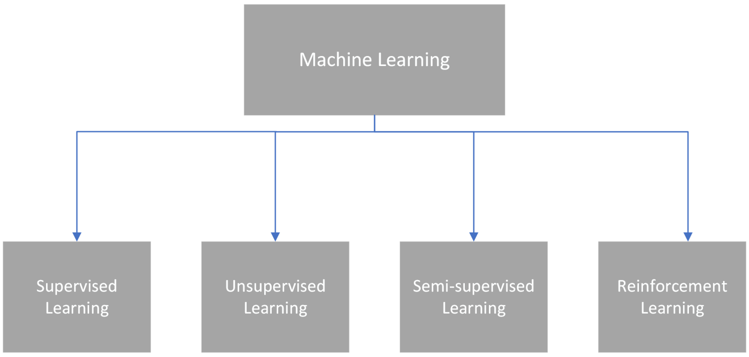

Figure 2.4: Machine learning classification

2.2.1 Supervised learning

In supervised learning, the algorithms are trained on labeled datasets. That means that the algorithm is fed with the information about how the output should look like or what the algorithm should expect for a certain input[8]. By understanding the known inputs and output(s) the algorithm is able to learn and predict unseen and unlabeled examples. Supervised machine learning is one of the most common techniques used in the larger domain of ML. Some of the commonly known types are:

- Regression: A continuous input variable is fed into an algorithm, which then calculates and returns a continuous output. One example of regression is the prediction of power consumption depending on the number of residents and rooms.
- Classification: The algorithm utilizes the provided input to classify it into a specific category. This category is selected from the set of possible classes on which the algorithm has been trained. An example is the classification of different dog breeds.
- Forecasting: In this approach, the input data is associated with a timestamp, enabling the algorithm to recognize temporal patterns and predict future outcomes based on historical trends. Stock price prediction is an example of this category.
- Sequence: A sequential data, commonly in the form of audio or words, is fed into the algorithm. The algorithm is then able to generate a sequence of outputs that are connected with each other. ChatGPT is a very good example of this.

Dataset

In supervised learning [8], the training data consists of “tagged” output data along with the input data. The model is fed input data X and the corresponding output data Y. During training the model learns patterns and behavior from labeled data, thus creating a mapping. The model acts as a function f which maps input X into the labeled data Y.

$$f : X \to Y \tag{2.3}$$

Once the model has “learned” during the training phase, it can now generalize to new unseen inputs. A dataset $\mathcal{D}$ can be thought of as an ordered set (x_i, y_i), while $x_i \in X$ and $y_i \in Y$. In case of regression both x_i and y_i are real-valued and can be multi-dimensional.

On the other hand, in case of classification problems, the output variable y_i is a discrete variable which can assume any class represented by it.

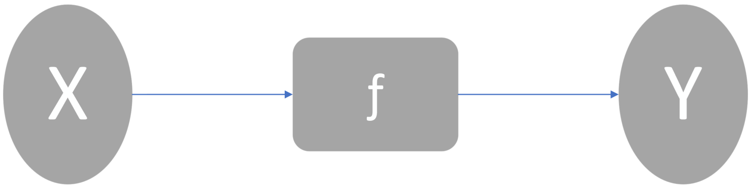

Figure 2.5: X, Y mapping

2.2.2 Linear Regression

Linear Regression is a widely used method to model a relationship between a dependant variable and one or more independant variables. This method tries to find the best-fit linear version of an equation which can describe the output behavior. If $y_1, y_2, ..., y_n \in Y$ is the output vector or the target vector and $x_1, x_2, ..., x_n \in X$ is the input vector, then linear regression can be defined as:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n \tag{2.4}$$

where, $\beta_0, \beta_1, ..., \beta_n$ are the tunable co-efficients that can be trained to model a particular dataset.

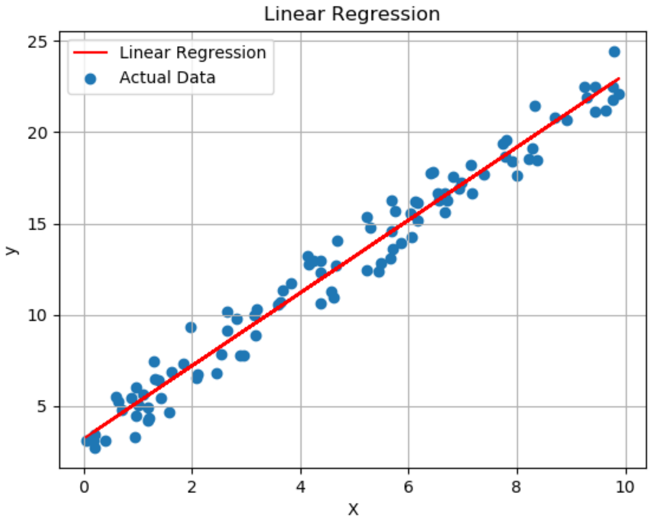

Figure 2.6: Linear regression

2.2.3 Random Forest

Random forest is a popular machine learning algorithm that is used for regression and classification problems. It is a type of ensemble machine learning method, which means that it combines a number of decision trees to generate an output. Random forest is quite useful as it offers robustness and flexibility. Additionally, it performs well with high dimensional datasets[9].

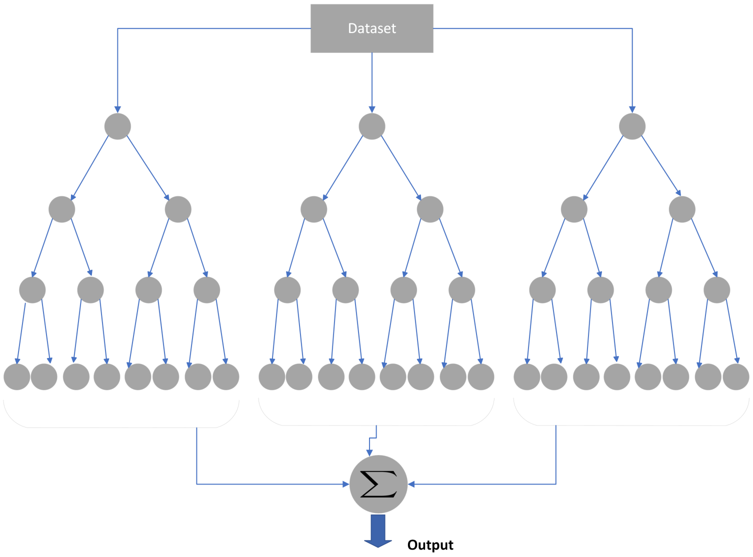

Figure 2.7: Random forests

The Random Forest algorithm operates by generating and training a multitude of decision trees, each operating on a random subset of the data. Decision trees in a random forest split data recursively based on different features to create branches and leaf nodes. Each decision tree generates an output and then all the outputs are aggregated to generate a prediction. The aggregation of results from multiple trees allows the algorithm to capture non-linearity and handle relationships between different features. This also ensures reduced over-fitting and better generalization on unseen data.

Random forest algorithm is robust to outliers and is good with missing data. Because of these advantages, random forest works well for many applications like regression analysis, classification problems and feature selection[10].

2.2.4 XGBoost

XGBoost is another powerful machine learning algorithm used for regression and classification tasks. It essentially works by combining predictions from weak learners to create a robust and accurate prediction.

The key concept lies in the optimization objective, which is really a combination of a loss function and a regularization term. To minimize the objective function weak learners (or decision trees) are iteratively added[11]. Mathematically, it can be represented as:

$$Obj = \sum loss(y_i, \hat{y_i}) + \sum \Omega(f_k) \tag{2.5}$$

where the loss function calculates the discrepancy between the real y_i and predicted values $\hat{y_i}$ for each instance. Each f_k represents an independent tree structure with associated leaf weights w and parameters q. The $\Omega(f_k)$ is the regularization term that manages the complexity of the problem.

Now to optimize the objective function, XGBoost uses gradient boosting, which is the minimization of objective function by adding weak learners. Each weak learner is trained to minimize the gradients of the loss function with respect to the predictions. The process can be modeled as:

$$\hat{y_i} = \hat{y}_{i-1} + \alpha * f_m(x_i) \tag{2.6}$$

where $\hat{y_i}$ is the predicted value at iteration i, $\hat{y_i} - 1$ is the prediction at the previous iteration, α is the learning rate, and $f_m(x_i)$ represents the prediction of the m-th weak learner for the input instance x_i.

Furthermore, this algorithm incorporates techniques such as pruning, regularization and feature sub-sampling, which enhances generalization and prevents over-fitting. All of these are already a part of objective function and the training process, which makes it even more efficient.

Due to the strong optimization framework, XGBoost can achieve remarkable performance on a variety of datasets. It is scalable[11], efficient, and has the ability to model complex relationships.

2.2.5 Deep Neural Networks

Deep neural networks(DNNs) [12] are machine learning algorithms that are inspired by the functioning of human brains. A DNNs is composed of layers of interconnected “nodes”, collectively these nodes can learn hierarchical structures in the data. Each of these "nodes" takes an n-dimensional vector $x_1, x_2, ..., x_n \in X$ as an input from the previous layer, the vector is then multiplied by a weight vector $w_1, w_2 \in W$ to produce an output a.

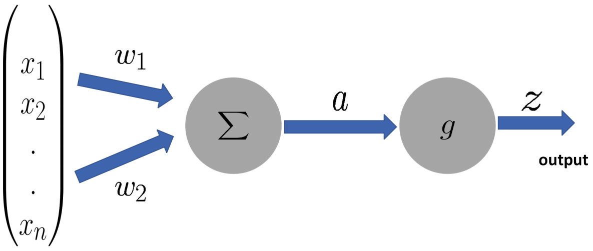

Figure 2.8: A single neuron

Mathematically a single neuron takes an input vector, applies a weighted sum and then passes the result through an activation function. This can be represented as:

$$z = g(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n) \tag{2.7}$$

A single layer of neuron will be:

$$y_i^l = g(\sum_j w_{ij}^l x_j^l + b_i^l) \tag{2.8}$$

Where,

y_i^l: output of the i-th neuron in the ℓ-th layer,

x_j^l: output of j-th neuron in the (ℓ-1) layer,

w_{ij}^l: weight associated with the i-th neuron in the ℓ-th layer,

b_i^l: bias associated with the i-th neuron in the ℓ-th layer,

g: activation function applied to a particular neuron

To tune these parameters, a loss function calculates the loss associated with a chosen set of parameters [12]. In case of regression, an example of a loss function $\mathcal{J}$ can be mean absolute error (MAE):

$$\mathcal{J} = \frac{1}{N} \sum (\mid y_i - \hat{y}_i \mid) \tag{2.9}$$

where N is the number of data-points, y_i is the real output for the i-th sample, and $\hat{y}_i$ is the predicted value for the i-th sample. The weights of deep neural networks are assigned randomly at the start but with each training cycle the weights are re-adjusted. The readjustment is done based on the calculated loss gradient w.r.t the weights. To update the weights, the calculated gradients are added to them [12]. The update can be illustrated as:

$$w_{new} = w_{old} - \alpha \frac{\partial \mathcal{J}}{\partial w_{old}} \tag{2.10}$$

Where α is the adjustable hyper-parameter known as the learning rate that can be tuned. Similarly, for bias the update equation will be:

$$b_{new} = b_{old} - \alpha \frac{\partial \mathcal{J}}{\partial b_{old}} \tag{2.11}$$

Activation Functions

Now that the basics working of a deep neural network is established, some background about the activation functions is needed. An activation function [13] plays an integral part in a neural network by introducing a non-linearity. It basically decides if a neuron should be fired or not. A neuron can also be partially "ON", which allows a neural network to learn complex and intricate patterns. Depending upon the application there are many types of activation functions. The discussion will be limited to regression tasks, so the common activation functions for regression tasks are:

- Identity function
- Sigmoid function
- Rectified Linear Unit (ReLU)

The identity function or simply the linear function is used for simple regression cases. This activation functions works well, when the behavior of the system can be modeled through linear equations. It preserves the original output produced by the neuron. Mathematically it can be represented as:

$$f(x) = x \tag{2.12}$$

Although identity functions work well with linear relationships, they are of no use when non-linear patterns are to be captured. Usually they are used at the output layer of the neural network in that case. Sigmoid function is an alternative that is used to model the non-linear relationships. The sigmoid function maps the input function between the values of 0 and 1. This is represented mathematically as:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.13}$$

Although, the sigmoid function can capture non-linearity, it has a disadvantage. Towards the extremes of the input range it tends to saturate which causes a problem of vanishing gradients. A problem in which the gradients become very small, which makes the training very slow.
Apart from these two options, there is also the rectified linear unit, commonly known as ReLU [14]. A ReLU function is also able to capture non-linearity. When the input is negative the function sets the output to zero, thus deactivating the neuron. In case of ReLU there is no upper bound, so it works well there is no predefined range for the input. It is represented as:

$$ReLU(x) = \begin{cases} x, & x > 0 \\ 0, & otherwise \end{cases} \tag{2.14}$$

The ReLU function is able to address the problem of vanishing gradients, an issue which causes the gradients of the loss function to become increasingly small as they are propagated back through the earlier layers of the network. This leads to slower convergence or even a complete halt in learning. It does not suffer from this problem, because the gradient is always 1 for positive input value. Thus it allows efficient training of the neural network.

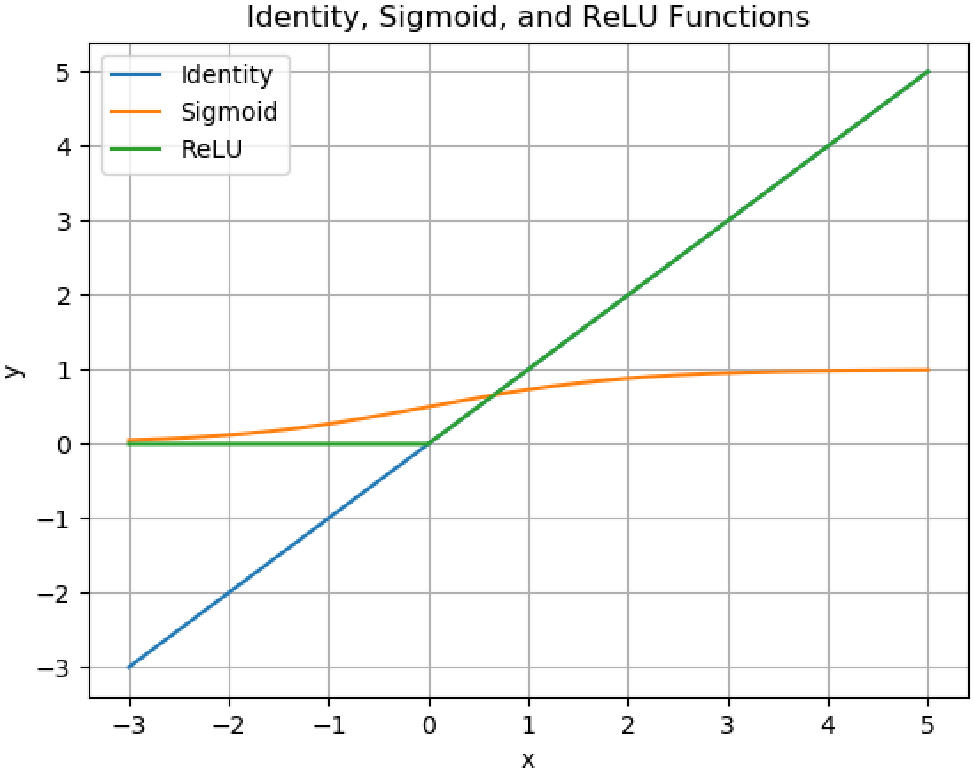

Figure 2.9: Comparison of Identity, Sigmoid and ReLU functions

Optimizers

Optimizers are crucial components in the training of deep neural networks (DNNs). They are the algorithms that are used to update a model's parameters after each iteration. It is done by minimizing the loss function, which in turn improves the model's performance. Each type of optimizer uses a unique technique to find the optimal parameter values, like weights and biases. Some of the common optimizers used in a DNN training are:

- Stochastic Gradient Descent (SGD): SGD is one of the simplest optimizers. It basically works by computing the gradient of the loss function with respect to each parameter using a single training example at a time. Since the gradients are updated on every single training example, this approach introduces a lot of noise in the parameters update. It's also slow to converge [15] because of this reason.

- Mini-batch Gradient Descent: This algorithm is actually an extension of SGD, where instead of updating on every single training example, the algorithm updates on a small batch of training examples. This approach reduces the noise in the parameters update and the algorithm is able to converge faster.

- AdaGrad (Adaptive Gradient Algorithm): AdaGrad is able to adapt the learning rate for each parameter based on historical trends. It dynamically assigns more weight to less frequently updated parameters and less weight to frequently updated parameters. This is especially useful for handling sparse data and it also helps to improve learning on parameters with different scales.

- RMSprop (Root Mean Square Propagation): AdaGrad has a problem of overly aggressive learning rate decay [16] and this problem is rectified by RMSprop. It solves this problem by calculating the moving average of squared gradients to control the learning rate adaptation.

- Momentum: This technique helps accelerate SGD in the right direction and dampens oscillations. It does that by adding a fraction of the previous update to the current update, which in-turn helps the optimizer maintain the velocity along the dominant directions of the loss function.

- Adam (Adaptive Moment Estimation): This method combines the benefits of momentum and RMSprop. It is based on adaptive estimation of learning rates for each parameter based on the first-order (mean) and second-order (uncentered variance) moments. Adam is widely used in practice because of its robustness and efficiency.

2.2.6 Hyperparameter Tuning

One crucial aspect of training machine learning models is hyperparameter tuning. It involves finding optimal values of hyperparameters that affect the learning process. Hyperparameters can be thought of as configuration settings that are configured before the training process begins. These values cannot be learned from the training data unlike the other model parameters. Each model can have its unique set of hyperparameters such as learning rate, number of nodes, depth of a decision tree, number of neurons in a neural network.

The central purpose of hyperparameter tuning is to find the specific set of hyperparameters which yield not only the best fit for the input data but the model is also able to generalize well on unseen data. Unfit hyperparameters can lead to a variety of problems which can lead to models that either underfit (insufficiently complex) or overfit (too complex) the data, resulting in poor performance. There are several approaches to hyperparameter tuning, including:

- Grid Search: Grid search is a very blunt or brute-force approach that tries to test all the possible combinations of hyperparameters (that are practical). Since this approach exhausts all the possible combinations, it always leads to a solution but at the same time it is computationally very expensive.
- Random Search: It was proposed by Bergstra and Bengio in their paper ”Random Search for Hyper-Parameter Optimization” (2012) [17]. It is an alternative to the grid search in which hyperparameters are randomly selected from a defined search space. Despite not utilizing all the possible solutions, this approach leads to good results. Random search is particularly useful with high dimensional search spaces, which are otherwise computationally expensive in case of grid search.
- Bayesian Optimization: This approach is based on Gaussian process and it models the performance metric such as validation error as a probabilistic surrogate model[18]. It balances the trade-off between the two: active learning (exploitation) and best objective function (exploration). It iteratively selects the next hyperparameters to evaluate based on an acquisition function that balances exploration and exploitation.
- Evolutionary Algorithms: These algorithms create a population of potential hyperparameter configurations and evolve them over several pre-specified generations. After a number of iterations it produces the best “offspring” or the hyperparameter setting [19].

2.2.7 Evaluation Metrics

Evaluation metrics are used to measure the performance and effectiveness of a machine learning model. These metrics provide a quantitative measure, which helps the researchers and practitioners to understand how well the model performs on a given dataset. Today, there are various evaluation metrics, each tailored and utilized for specific machine learning goals. These metrics are selected based on tasks like regression, classification, clustering etc.

In classification tasks, the goal is to assign data points to predefined categories or classes, so accuracy is the widely used evaluation metric. As the name describes, accuracy is used to measure the proportion of correctly predicted labels over a number of datapoints in a dataset. Although accuracy works well with straightforward datasets with balanced classes, it does not work well with imbalanced datasets. In imbalanced datasets, some classes are more prevalent than the others, so metrics like precision, recall, and the F1-score work well. Precision and recall for example are defined as:

$$Precision = \frac{TP}{TP + FP} \tag{2.15}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.16}$$

Where, TP (True Positives) are the values that are predicted as true, and are actually true, FP (False Positives) are the values that are predicted as true but are not actually true and FN (False Negative) are the values that are predicted not to be true but are actually true.

In case of binary classification tasks, two common evaluation metrics are the receiver operating characteristics (ROC) and the area under the curve (AUC). The ROC curve plots the true positive rate against the false positive rate at various classification thresholds. So it is a graphical method used to measure the performance of a binary classifier. The AUC is the measure of the area under the ROC curve, so it summarizes the performance of a model in the form of a singular scalar value. The True Positive Rate (TPR) is the same as Recall while the False Positve Rate (FPR) is defined as:

$$FPR = \frac{FP}{FP + TN} \tag{2.17}$$

where, TN (True Negaives) are the values that are predicted as not true and are actually not true. An example ROC curve could look like:

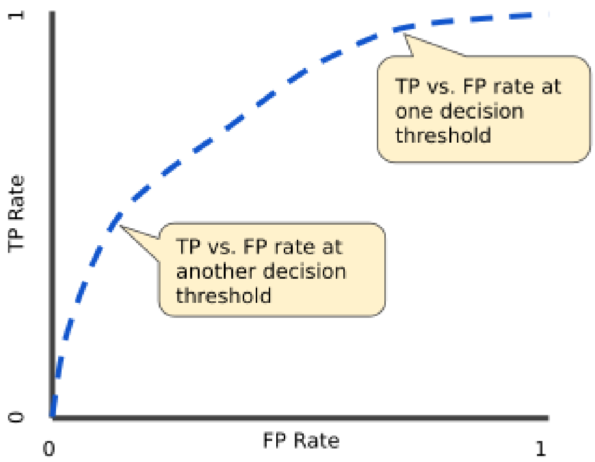

Figure 2.10: ROC Curve [20]

In regression tasks, for example, the goal is to predict continuous values based on a given input. The metrics like mean squared error (MSE) and mean absolute error (MAE) are the standards. MSE calculates the average squared difference between the predicted and actual values while the MAE calculates the absolute difference between the two. Thus, MSE gives more weight to larger errors while the MAE is less sensitive to outliers compared to MSE.

Depending on the task at hand, a solid understanding of each of these evaluation metrics is essential for creating a practical model. By the correct use of these metrics the model performance can be evaluated and enhanced. Interpreting these values allows the practitioner to make informed decisions and draw right conclusions about a model.

2.3 Optimization

To grasp the optimization techniques employed in this thesis, a foundational understanding of the concept of optimization is essential. Optimization plays an important role in various fields such as machine learning, operational research, and engineering. The usual goal is to find the best possible solution or to achieve an optimal value for a given problem.

In the context of this thesis, optimization will refer to minimizing or maximizing a certain objective function, while maintaining certain constraints and limitations. The objective function will typically be minimizing a certain value to be introduced later on.

There are several optimization techniques, but the central goal they share is to explore the solution space and identify the point that provides the best solution to the problem. Optimization techniques iteratively adjust and refine the selected parameters to reach a certain outcome. By leveraging mathematical algorithms and search strategies, optimization algorithms traverse the solution space, evaluate different possibilities and iteratively improve the objective function value. Mathematically, this can be represented as an iterative process:

$$x^{(k+1)} = x^k + \Delta x \tag{2.18}$$

In the given context, x represents a vector of decision variables that influence possible solutions, and Δx denotes a minor shift in the search space following the k-th iteration. One commonly used approach is gradient-based optimization. It utilizes the gradient information of the objective function to guide the search towards the optimal solution. Other optimization methods are evolutionary algorithms, swarm intelligence, and mathematical programming. All of them are very much used for optimization problems.

In practical applications, optimization is used to solve a wide range of complex problems. The problems can vary from resource allocation, scheduling, logistics to parameter tuning in machine learning models. Researchers and practitioners are able to improve and enhance effectiveness in their respective domains.

Broadly optimization is divided into several categories, which include deterministic and stochastic methods, continuous and discrete optimization, as well as constrained and unconstrained optimization. The selection of an appropriate technique depends on the problem structure, constraints and the characteristics of a particular problem.

Overall, optimization is indeed a very useful technique used to find optimal solutions in any domain. By employing these techniques, complex problems, process optimization and new advancements can be made. Since the optimization technique used in this thesis is based on Evolutionary Algorithms, the next section will focus solely on evolutionary algorithms and the techniques used.

2.3.1 Evolutionary Algorithms

Evolutionary algorithms (EAs) are optimization algorithms based on the principles of evolution and natural selection. They work by maintaining a population of candidate solutions, which evolve iteratively through processes of selection, recombination, and mutation. The suitability of each solution is assessed using a fitness function that determines the quality of the solution for the problem at hand.

The workflow of evolutionary algorithms typically involves:

Initialization: A population of candidate solutions is generated, typically randomly. The size of this population is usually held constant through the evolution process.

Fitness Evaluation: Each solution in the population is assigned a fitness score, calculated using a pre-defined fitness function. This score determines the solution's suitability to the problem being addressed.

Selection: Candidate solutions are selected for reproduction based on their fitness scores. The selection process is typically stochastic, with solutions having higher fitness scores having higher probability of being selected.

Recombination: Offspring are generated by combining elements of two or more parent solutions. The recombination process varies widely depending on the specific type of EA being used.

Mutation: Random changes are made to the offspring solutions to maintain population diversity and avoid premature convergence to a sub-optimal solution.

Replacement: The current population is replaced by the newly generated offspring. The replacement strategy can be deterministic (e.g., replace the worst solutions) or stochastic.

Algorithm 1 General: Evolutionary Algorithm

```
Initialize population with random candidate solutions
while termination criteria not met do
    Evaluate: each candidate's fitness
    Select: individuals for reproduction based on fitness
    Recombine: Generate offspring by combining elements of selected individuals
    Mutate: For each offspring, apply random changes with a certain probability
    Evaluate: each offspring's fitness
    Replace: population with new offspring population
end
Result: Best solution found
```

Chapter 3

Literature Review

Optimization techniques, lithography, regression models, and the on-resistance (R_{on} or R_{ds}) are fundamental concepts presented in this thesis. The integration of these concepts has the potential to create powerful models for predicting and optimizing the number of chips on a wafer.

3.1 Lithography

Lithography is a critical step in semiconductor manufacturing, and the yield from lithography processes is a key factor in determining the cost and performance of the final product[21]. Several factors affect the number of chips on a wafer, including the precision of the lithography equipment, the quality of the photomask, and the conditions of the process[22]. Research and optimization of lithography is a vital area of study in semiconductor manufacturing.

3.2 Regression

Regression models serve as essential tools in both statistics and machine learning, facilitating the prediction and forecasting of outcomes based on independent variables. There are many types of regression models like linear, logistic, polynomial, etc., each serving a specific purpose depending on the type and distribution of data[23]. In semiconductor manufacturing, regression models have been employed to predict outcomes like yield and performance based on process parameters [24].

3.3 Resistance Drain-Source (R_{on})

The resistance drain-source (R_{on} or R_{ds}) is a critical parameter in MOSFET devices, affecting the device's power usage and speed. Lowering R_{on} allows for faster switching speeds and lower power usage, which are key goals in semiconductor device design[25]. The R_{on} models presented in the paper [26] are borrowed and used during the optimization part of this thesis.

3.4 Optimization

Optimization methods are essential in semiconductor manufacturing to minimize the cost and maximize the output and performance. This can involve optimizing process parameters, designs, or schedules. Traditional optimization techniques include linear programming and statistical methods, but more recent approaches use evolutionary algorithms and machine learning techniques[27].

3.5 Conclusion

In summary, the integration of lithography, regression models, optimization techniques, and the R_{on} parameter presents a promising approach to improving outcomes in semiconductor manufacturing. These elements, when considered collectively, have the potential to refine our predictions and understanding of the lithography process and the overall semiconductor production.

Chapter 4

Methodology

This study uses several regression models, which include linear regression, random forest, XGBoost and Deep Neural Networks (DNNs) to make predictions. All of these models are essentially able to predict the number of chips on a wafer with varying accuracy. Following these predictions, differential evolution was applied to optimize the number of chips on a wafer. This chapter discusses the details of the steps followed during prediction and the optimization. The exact hyperpareters and results will be described in Chapter 5. This chapter will attempt to answer the following questions:

- How are different features selected?
- What sort of normalization is applied to the input features?
- Which methods are used for predicting the number of chips on a wafer?
- Which method is used for the optimization of chip geometry?

4.1 Feature Selection

The first step was to select the input features. This was done using data visualization, hit-and-trial and calculating correlations. During feature selection relevant inputs are identified and retained while the redundant ones are removed. This increases the accuracy and saves a lot of computation time during the training process.

Reticle utilization factor (RUF) is the measure of how well the area of a single reticle is utilized. So if the chip has a width X and height Y, and no chip rotation is allowed then using Equation 2.1 the RUF has the following spread:

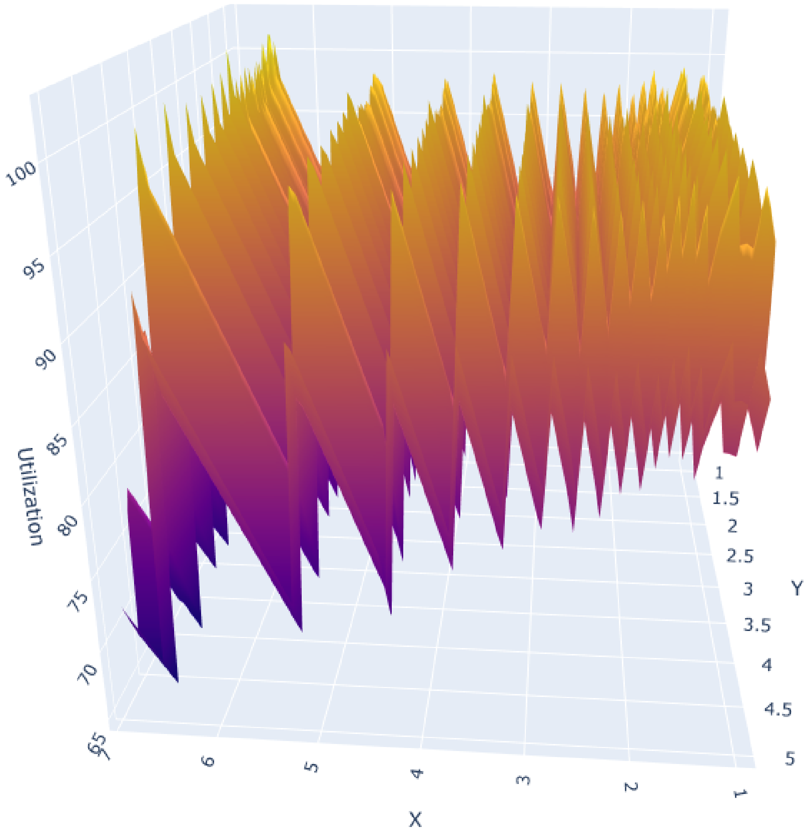

Figure 4.1: Chip Dimensions Vs Reticle Utilization

As expected for very large chip sizes, the RUF is bound to be very poor. For other sizes, there is a recurring pattern in which RUF keeps on oscillating. A number of features such as chip width, chip height, chip size, reticle width, reticle height, and number of chips were correlated. The resulting correlation matrix is showed in Figure 4.2. All of these features are highly correlated with the number of chips but due to data unavailability, reticle width and height are not used as input features.

4.2 Data Normalization

Prior to the implementation of the models, the input features were normalized to ensure smooth and better training. Normalization is an essential step in data preprocessing since different features exist on different scales [28]. This essentially speeds up the training process while increasing accuracy. Normalization was done depending on the model being trained.

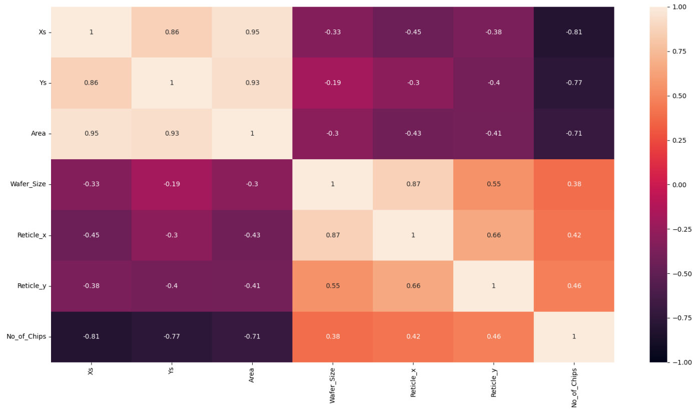

Figure 4.2: Feature Correlation Matrix

For linear regression, the sklearn's RobustScalar was utilized. RobustScalar rescales the dataset by using the following equation:

$$X_{\text{scaled}} = \frac{X - Q1(X)}{Q3(X) - Q1(X)} \tag{4.1}$$

where, X represents the data and $Q3(X) - Q1(X)$ represent the interquartile range (IQR). By using this range instead of the standard deviation, the RobustScaler reduces the influence of outliers on the scaling process without completely eliminating them.

In the case of Deep Neural Networks (DNNs), a normalization layer was used at the input stage. Normalization plays a pivotal role in deep learning. By ensuring consistent scales for input features, it prevents minor variances from inducing significant output changes, thus stabilizing the network. Additionally, normalization facilitates faster training by improving optimization conditions and averts potential issues like activation function saturation, thereby maintaining consistent gradient norms [29].

The normalization layer used in this study standardizes the inputs to have zero mean and unit variance, a method often referred to as Z-score or Standard Score normalization. The equation for this method is as follows:

$$X_{norm} = \frac{X - \mu}{\sigma} \tag{4.2}$$

where μ is the mean and σ is the standard deviation of the input features. By maintaining zero mean and unit variance, this type of normalization prevents the DNN from giving more weight to features with a higher average or variability, leading to more accurate predictions [29].

The use of normalization methods has been validated in various studies as effective in enhancing model performance, by ensuring that the optimization algorithm converges faster to the minimum, and by preventing the algorithm from getting stuck in local optima [30] [23].

4.3 Regression Models

Various regression models are employed in this study to predict the number of chips on a wafer. Each model brings a unique approach to prediction and has its advantages and potential limitations.

4.3.1 Linear Regression

The model used in this study is implemented using the sklearn ElasticNet linear regression model. This form of regression is easy to understand and interpret but it does not work well when the relationships are complex and non-linear [31].

ElasticNet is a linear regression model that combines both $L1$ and $L2$ regularization, from Lasso and Ridge regressions respectively, to both avoid overfitting and allow for feature selection. It can be useful when there are multiple correlated features. The hyperparameters of sklearn's ElasticNet used here are:

- alpha: This is the parameter that scales the penalty term. It's a constant that multiplies the penalty terms, thereby determining the amount of regularization. A larger value of alpha results in a higher level of regularization, which can help to avoid over-fitting. The default value is 1.0.

- *l1_ratio*: This is the ElasticNet mixing parameter, and it is used to balance the contributions of $L1$ and $L2$ regularization in the model. For $l1\_ratio = 1$, the penalty is an $L1$ penalty (Lasso), and for $l1\_ratio = 0$, it's an $L2$ penalty (Ridge). For values between 0 and 1, you get a combination of $L1$ and $L2$.

4.3.2 Random Forest

Random Forest is renowned for its resilience to outliers and its ability to manage non-linear datasets effectively. Nevertheless, its limitations include potentially suboptimal performance on datasets with very high dimensions or those of a smaller size [32]. Since, the dataset being used here is not very high dimensional, the results should be satisfactory. For this study the random forest model from sklearn was used. Here are the main hyperparameters for sklearn's RandomForestRegressor:

- n_estimators: The number of trees in the forest. More trees reduce the variance of predictions, improving the model's performance. However, after a certain number, performance gain is negligible, and the extra computation becomes inefficient. Default is 100.

- max_features: The number of features to consider when looking for the best split. It can take a number of types: "auto", "sqrt", "log2", or None (consider all features), or an integer (exact number of features), or a float (fraction of total features). Choosing max_features ¡ n_features leads to a reduction of variance and an increase in bias. Default is "auto".

- max_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. The higher the value, the more complex the model, leading to higher chances of overfitting. Default is None.

- max_leaf_nodes: The maximum number of leaf nodes. If None then unlimited number of leaf nodes. If not None then max_depth will be ignored. Default is None.

4.3.3 XGBoost

XGBoost or Extreme Gradient Boosting is an implementation of gradient boosted decision trees specifically designed for speed and performance [11]. In XGBoost an ensemble of weak prediction models, typically decision trees, are used simultaneously to create a strong prediction. They are able to handle a variety of data types, relationships and distributions [11]. In this thesis, XGBoost was implemented using the Python XGBoost library. Here are some of the main hyperparameters for XGBoost:

- min_child_weight: Minimum sum of instance weight (hessian) needed in a child. Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.

- gamma: Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm will be.

- subsample: Subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees and this will prevent overfitting.

- colsample_bytree: Subsample ratio of columns when constructing each tree.

- max_depth: Maximum tree depth for base learners. Increasing this value makes the model more complex and likely to overfit.

4.3.4 Deep Neural Networks(DNNs)

Deep Neural Networks (DNNs) are a type of artificial neural networks with multiple hidden layers [33]. These hidden layers allow the model to learn complex patterns and hierarchies in the data. The use of multiple hidden layers is indeed very powerful, but it may also require lots of data for training [33]. Also, DNNs are prone to over-fitting. In this thesis, the DNN model was built using Keras with a Tensorflow backend. The network consists of several dense layers, the specific number and configuration was determined by trial and error. Deep Neural Networks (DNN) in TensorFlow have numerous hyperparameters. Some of them related to regularization, optimizer, and learning rate are:

- $L1/L2$ regularization: Regularization methods that add a penalty proportional to the $L1$ or $L2$ norm of the weights to the loss function in order to reduce overfitting by discouraging complex models.

- Optimizer: The optimization algorithm used to minimize the loss function. Some popular choices include SGD (Stochastic Gradient Descent), Adam (Adaptive Moment Estimation), RMSProp (Root Mean Square Propagation), etc.

- Learning rate: The size of the steps the optimizer takes to reach the minimum of the loss function. A smaller learning rate might converge slowly, while a larger learning rate might skip the minimum.

4.3.5 Hyperparameter Tuning

To obtain the best results, a critical step was the tuning of hyperparameters for each regression model to ensure optimal performance. For the Linear Regression model, regularization terms were tuned to control the complexity of the model. For the Random Forest model, parameters like the number of trees, the maximum depth of the trees, and the minimum number of samples required to split a node were optimized. In the case of the XGBoost model, learning rate, maximum tree depth, and the number of estimators were among the parameters tuned. The Deep Neural Network (DNN) required a more complex set of hyperparameters to be tuned, including the number of hidden layers, the number of neurons in each layer, and the learning rate.

To determine the best values for these parameters, a Grid Search methodology was employed. Grid Search involves specifying a subset of the hyperparameter space and exhaustively trying all combinations within this subset. For each combination of parameters, the model was trained and evaluated using cross-validation, and the performance of the model was recorded. This method allowed for the identification of the hyperparameters that resulted in the best performance for each model. After the optimal hyperparameters were found, these were used to train the final versions of each model on the entire training dataset.

Through meticulous hyperparameter tuning using Grid Search, it was ensured that each model was given the best chance to reach its maximum predictive performance, leading to more robust and reliable results.

4.3.6 Geometrical Approach

The problem at hand involves predicting the number of chips that can be fitted into a wafer, which is essentially a circular disk with a notch in it. One approach to solve this problem is through geometric methods.

In this approach, we can consider the circular wafer as a two-dimensional shape and use geometric calculations to determine the maximum number of chips that can be placed within it. By carefully analyzing the dimensions of the chips and the wafer, taking into account the notch in the wafer, we can derive an optimized arrangement of chips to maximize the number that can fit inside the circular area. The problem is solved algorithmically using Algorithm 2. Figure 4.3 shows the placement of first two rows in the top half circle.

Algorithm 2 Calculation: Maximum Number of Chips in a Wafer

```
Initialize $total\_chips$ and $used\_height$ to 0.
Set $p$ (the perpendicular distance from the wafer's center to the chip) as $\frac{chip\_height}{2}$.
for $each\ half\ in\ \{Top, Bottom\}$ do
    while $used\_height < r$ (half = Top) or $used\_height < r - notch$ (half = Bottom) do
        If First Iteration Update $p = p + chip\_height + kerfy$.
        Calculate $upper\_bound = 2 \times \sqrt{r^2 - p^2}$ (the maximum horizontal distance that the chips can occupy in a row).
        Calculate $chip\_count = \left\lfloor \frac{upper\_bound}{chip\_width + kerfx} \right\rfloor$ (the maximum number of chips that can be placed within a row).
        Add $chip\_count$ to $total\_chips$. $used\_height = p + chip\_height + kerfy$
    end
end
Result: The maximum number of chips that can fit in the wafer is given by total_chips.
```

This geometric approach provides valuable insights into chip placement strategies and can be used as a complementary method to machine learning models for chip area optimization.

4.4 Optimization With Differential Evolution

After creating and selecting a regression model, the Differential Evolution (DE) technique was employed for optimization of number of chips on a wafer. DE, which is a population-based metaheuristic optimization algorithm, was initially proposed by Storn and Price in 1997. It excels in dealing with continuous variables [34].

Differential Evolution continuously generates trial candidate solutions in the solution space and selects the best “offspring” for the next generation based on their fitness value, which is the quality of the solution. The whole algorithm has three main steps, which are mutation, cross-over and selection. These steps are performed iteratively until a termination criteria is met. The termination criteria could be a maximum number of

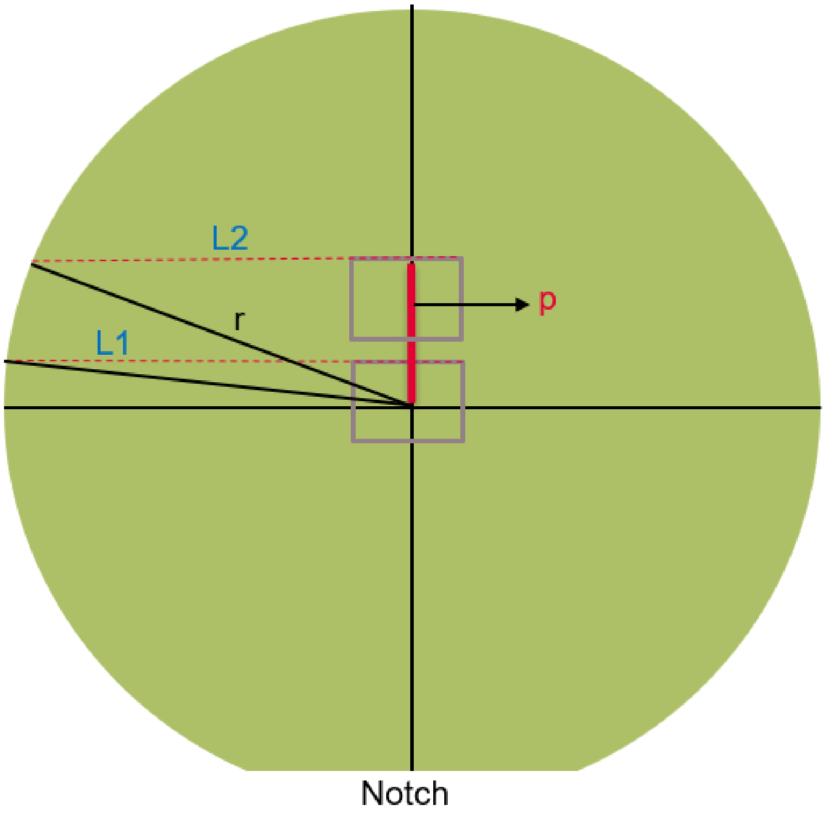

Figure 4.3: Feature Correlation Matrix

iterations or a minimum error threshold [34].

In the context of this thesis, the central goal is the optimization of the maximum number of chips on a wafer, while keeping the on-resistance R_{ds} or R_{on} in the defined error range. To achieve this an objective function is defined which takes in the chip width x, chip width y, wafer size, maximum R_{on}, minimum R_{on} and the R_{on} tolerance.

As one of the key advantages of differential evolution is that it requires fewer adjustable parameters compared to other evolutionary algorithms such as particle swarm optimization or PSO, it is easy to use. DE has been found to be very effective in handling non-linear and non-differentiable optimization problems. Because of this, it is an excellent choice for a variety of complex optimization problems [35]. The DE algorithm used here is borrowed from the Python's Scipy library. The following are some of the important hyperparameters in this algorithm:

- maxiter: The maximum number of generations over which the entire population is evolved. The maximum number of function evaluations (with no polishing applied) is: (maxiter + 1) * popsize * len(x), where len(x) is the number of parameters in the function to be optimized.

- strategy: This is the differential evolution strategy to use. It should be one of 'best1bin', 'best1exp', 'rand1exp', 'randtobest1exp', 'currenttobest1exp', 'best2exp', 'rand2exp', 'randtobest1bin', 'currenttobest1bin', 'best2bin', 'rand2bin', 'rand1bin'. The default is 'best1bin'.

- popsize: A multiplier for setting the total population size. The population has popsize * len(x) individuals (where len(x) is the number of parameters in the function to be optimized).

- mutation: The mutation constant. In the literature, this is also known as differential weight. This should be a floating-point value in the range [0, 2].

- recombination: The recombination constant, should be in the range [0, 1]. In the literature, this is also known as the crossover probability.

Chapter 5

Results & Empirical Data

In this chapter, an overview of the obtained results, the data utilized, and the experimental process implemented to address the research questions is presented. Section 5.1 is dedicated to a detailed examination of the dataset and the experimental setup. Subsequently, the results achieved through this study are comprehensively articulated in Section 5.2. Overall, the following questions will be answered in this chapter:

- How many data points were utilized in the training, testing, and analysis stages?
- What software libraries or tools were employed for metrics calculation, dataset generation, and conducting experiments?
- What strategies were used to optimize the algorithms?
- How did the performance of different regression algorithms compare?
- Which tools were used for the optimization process?
- How does the effectiveness of the optimization compare to that of existing solutions or products?

5.1 Empirical Data

The subsequent section provides an empirical quantification to the methodology delineated in the preceding chapter. Figure 5.1 illustrates the data dispersion through a box-and-whisker plot, whereas Table 5.1 provides a comprehensive statistical overview.

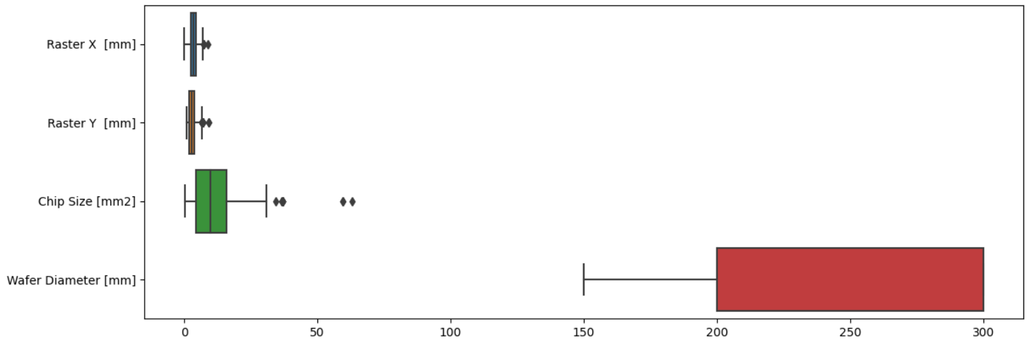

Figure 5.1: Outlier Detection

Table 5.1: Dataset Overview

	Raster X [mm]	Raster Y [mm]	Chip Size [mm2]	Wafer Diameter [mm]
Count	701	701	701	701
Mean	3.664775	2.904724	11.970264	238.516405
Std.	1.562096	1.511133	9.893986	56.252709
Min	0.441776	0.158417	-6.198533	150.000000
25%	2.465412	1.774196	4.716342	200.000000
50%	3.531037	2.715778	9.673779	200.000000
75%	4.536036	3.731191	16.589483	300.000000
Max	9.437646	9.316388	64.638373	300.000000

As illustrated in Figure 5.1, only a handful of outliers are present among the input features. Distinctly different distributions can be observed for different features, as depicted in Figure 5.2. To rectify this issue, we employed the RobustScaler from the scikit-learn library for linear regression. For the Deep Neural Network (DNN) model, the normalization layer from Keras was utilized to standardize the input features. However, for the Random Forests and XGBoost models, no normalization techniques were applied, as these methods are not reliant on normalization[36]. The adjusted distributions post-application of the scikit-learn's RobustScaler are depicted in Figure 5.3.

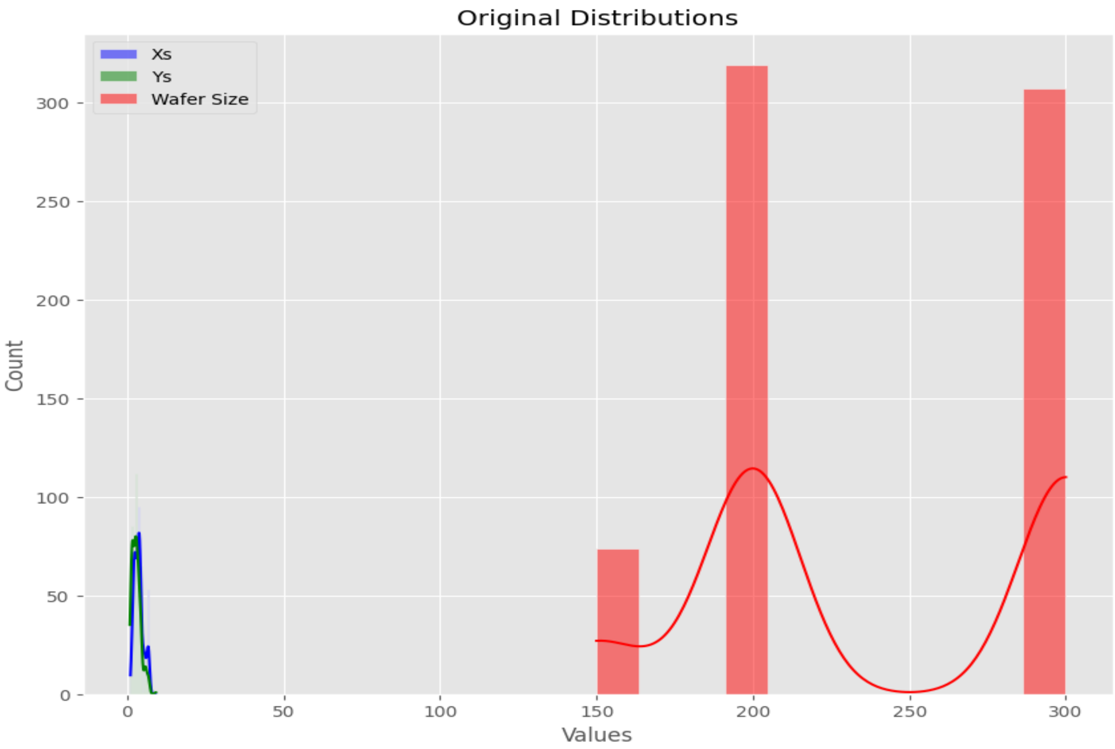

Figure 5.2: Feature Distribution

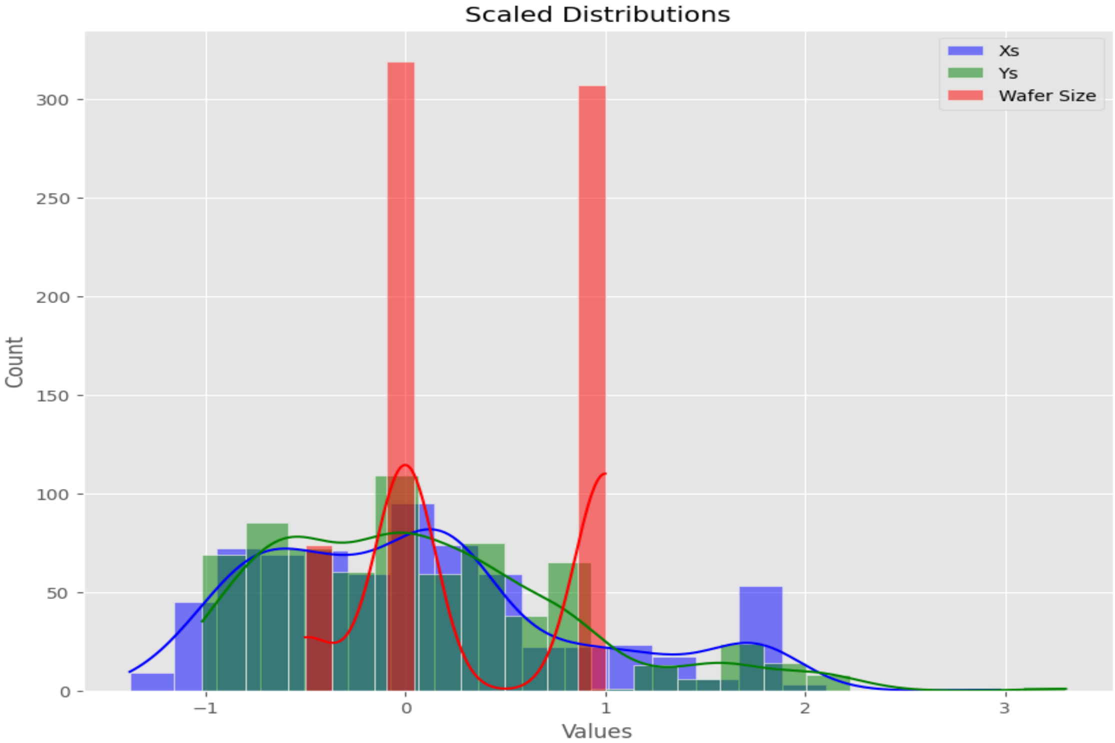

Figure 5.3: Feature Distribution After Scaling

Manufacturing data is extracted from an Excel file and imported into a Pandas

DataFrame employing the 'read_excel()' function. Upon loading, duplicates are eliminated and outliers are identified. The data is then visualized using the Seaborn library, followed by the creation of distinct training and testing datasets from the preprocessed data. Algorithms, such as linear regression, random forest, XGBoost, and Deep Neural Networks (DNNs), are trained using the training data. Once training is finalized, these models are evaluated using the test data, facilitating the tuning of hyperparameters based on the obtained results. An overview of the entire process is presented in Figure 5.4.

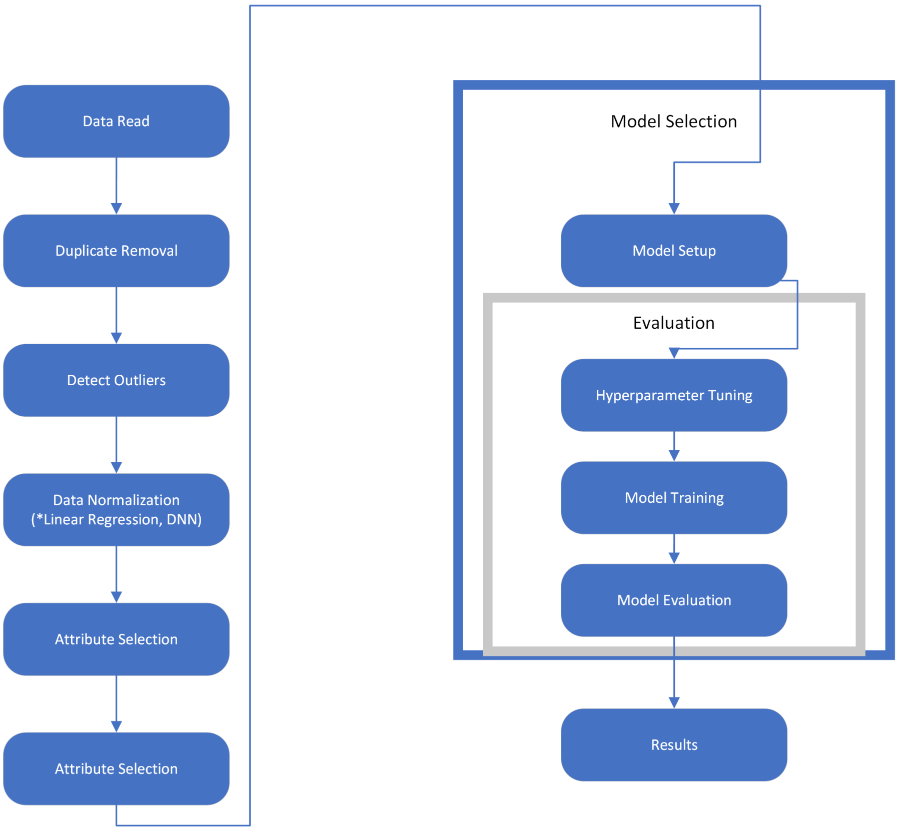

Figure 5.4: Modelling Process

Upon completion of the modeling phase, the developed model was utilized in conjunction with the R_{on} model, as delineated in Section 3.3, for the subsequent optimization process. The optimization algorithm implemented in this context was the Differential Evolution algorithm. A minimum of four pre-existing products were enhanced using this algorithm. The concept is graphically represented in Figure 5.5.

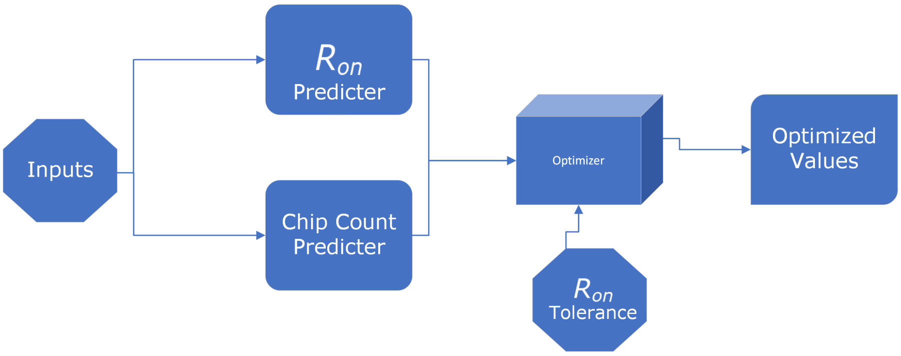

Figure 5.5: Optimization Process

The following python libraries were used for the different steps:

- dash
- seaborn
- matplotlib
- pandas
- numpy
- scipy
- scikit-learn
- XGBoost
- tensorflow

5.2 Results

This section commences with the examination of the training error from the Deep Neural Network (DNN) model, followed by the presentation of results from all regression models for subsequent comparison. Upon providing a summary of the regression results, selected examples of optimization are introduced. These instances have been selected from products already existent within the industry.

5.2.1 Regression

For methodologies including linear regression, random forest, XGBoost, and Deep Neural Networks (DNNs), hyperparameter tuning was performed as delineated in Subsection 4.3.5. Essentially, this entailed extensive and exhaustive testing of different parameters. The Table 5.2 enumerates the various hyperparameters that were subjected to testing.

Table 5.2: Hyperparameters for each model

Model Name	**Hyperparameter**	**Values**
Linear Regression	alpha	[0.1, 0.3, 0.5, 0.7, 0.9]
	L1 ratio	[0.1, 0.3, 0.5, 0.7, 0.9]
Random Forests	n_estimators	[25, 50, 100, 150, 300]
	max_features	['sqrt', 'log2', None]
	max_depth	[3, 6, 9, 12, 15, 18]
	max_leaf_nodes	[3, 6, 9, 12, 15]
XGBoost	min_child_weight	[1, 5, 10]
	gamma	[0.5, 1, 1.5, 2, 5, 8, 10]
	subsample	[0.6, 0.8, 1.0]
	colsample_bytree	[0.6, 0.8, 1.0]
	max_depth	[3, 4, 5, 6, 7, 8]
DNN	L1	[0, 0.001, 0.1, 1]
	L2	[0, 0.001, 0.1, 1]
	optimizer	['Adam', 'Adadelta']
	learning rate	[0, 0.001, 0.1, 1]

Upon examining the various hyperparameter combinations, the models that demonstrated the best performance in minimizing Mean Absolute Error (MAE) were selected. Figure 5.6 elucidates the progression of training and validation losses, while Table 5.3 provides a detailed account of the achieved metrics. These include training Mean Square Error (Train MSE), training Mean Absolute Error (Train MAE), testing Mean Square Error (Test MSE), and testing Mean Absolute Error (Test MAE). In the case of the geometric model, since it does not require training, its performance is assessed solely based on the test data.

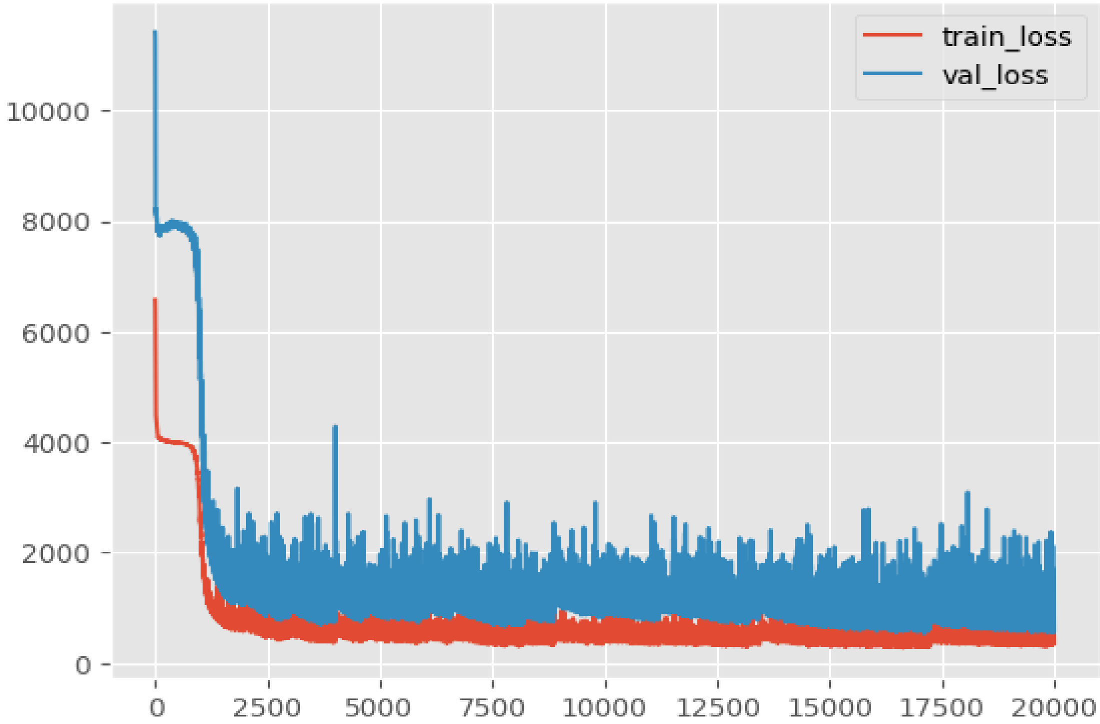

Figure 5.6: DNN Losses

Table 5.3: Models Performance

	Train RMSE	**Train MAE**	**Test RMSE**	**Test MAE**
Linear Regression	3986.8	5604.8	3986.8	5239.3
Random Forests	99.19	266.2	2462.4	922.9
XGBoost	567.1	385.8	2426.8	1132.1
DNN	684.7	307.4	864.3	475.7
Geometric Model	–	–	1108	587

5.2.2 Optimization

The optimization process utilized the Differential Evolution (DE) algorithm, a robust and efficient optimization method renowned for its efficacy. The primary goal was to retain the R_{on} value within a specified range of tolerance. This objective is paramount as it strikes a balance between the power performance of the device and the prospect of proposing an optimized chip geometry.

In the heart of the DE algorithm lies the objective function that it strives to minimize.

This objective function takes into account factors such as chip dimensions, wafer size, the R_{on} model, and the model for the number of chips. If the chip dimensions, generated during a certain generation, fail to result in an R_{on} value within the designated range, that generation is penalized relative to the error magnitude. Conversely, if the R_{on} lies within the specified range, the function returns the negative of the number of chips generated by the objective function, which DE then seeks to minimize through iterative evolution.

The model underwent rigorous testing using an array of parameter combinations during the experimentation process. Various facets of the algorithm's configuration, such as population size, mutation factor, and crossover strategy, were evaluated. The comprehensive testing process assisted in identifying the most effective settings, thereby contributing to the robust performance of the optimization model.

Lastly, the algorithm was deployed on four distinct product models available in the market. These models represent typical devices in the semiconductor industry, serving as real-world examples. The algorithm was tested on both the reticle Equation 2.1 and the DNN regression model for the number of chips on a wafer. The practical application of the algorithm to these product models tested its viability and effectiveness. The results derived from the DNN regression model are summarized in Table 5.4, while those obtained from the reticle floorplanning optimization are presented in Table 5.5.

Table 5.4: Wafer Floorplanning Optimization Performance

	No. of Chips		**Difference**	
Type	**Original**	**Optimized**	**No. of Chips**	R_{on}
A	2048	2213	+8%	0.84%
B	3129	3428	+9.5%	0.89%
C	3735	3935	+5.3%	0.83%
D	4253	4284	+0.7%	0.89%

Table 5.5: Reticle Floorplanning Optimization Performance

	No. of Chips		**Difference**	
Type	**Original**	**Optimized**	**No. of Chips**	R_{on}
A	25	28	+12%	0.03%
B	36	42	+16.7%	0.01%
C	40	50	+25%	0.03%
D	54	55	+1.85%	0.11%

5.2.3 Discussion

The review of results from the regression models developed for chip quantity prediction reveals that the Deep Neural Network (DNN) model significantly outperformed other models based on the test mean absolute error (MAE). The Linear Regression model, showing subpar performance across all metrics, notably underfit the data. This is primarily attributed to the non-linear relationship between the input features and the output.

Interestingly, both Random Forest and XGBoost models demonstrated disappointing results. In fact, the performance of XGBoost was even weaker than that of Random Forest, an observation possibly attributable to noise and the small dataset size. Noise appears to impact XGBoost more negatively, hence the better performance of Random Forest. However, their overall lackluster performance could be traced back to the dataset's small size.

In response to these issues, a carefully designed DNN model, not overly deep, was deployed, yielding superior results compared to other regression models. To mitigate overfitting in DNN, both $L1$ and $L2$ regularization were utilized. Notably, the DNN model performed well even when compared to the geometric model, which exploited the problem's geometric aspect.

Turning the attention to the optimization phase, the Differential Evolution algorithm proved successful in increasing the number of chips in both cases: whether the R_{on} predictor was linked with the number of chips in a reticle or the number of chips on a wafer. Interestingly, the optimizer delivered better results when used with the number of chips on a reticle. When applied to the number of chips on a reticle, the optimizer induced a more significant percentage increase in chip quantity compared to when used with the number of chips on a wafer. Moreover, the model coupling the R_{on} predictor with the number of chips on a reticle yielded a lower R_{on} deviation compared to the model linked with the number of chips on a wafer. This might be attributed to working more closely with the reticle— the source of wafer chips through multiple shots— which forms the core of the problem.

Chapter 6

Conclusion

Lithography, as a key component of the chip fabrication process, acts as a bottleneck in manufacturing, thereby necessitating optimization. The optimization scope of this process encompasses two essential stages: reticle floorplanning and wafer layout design. Effective chip size optimization and prudent floorplanning are potential solutions to optimize the number of chips generated from a single wafer.

To optimize the number of chips produced, a predictor was developed to predict the number of chips on a wafer based on specific chip geometry and the wafer in use. Treating this prediction as a regression problem, an assortment of machine learning models were prepared, tuned, and tested. The models—comprising linear regression, random forest, XGBoost, and Deep Neural Networks—were validated against a geometry-based model. Deep Neural Networks emerged as the most precise model according to mean absolute error (MAE), the chosen metric for comparison.

Following the establishment and selection of the most accurate regression model, the requirement for an optimizer became evident to further optimize chip geometry. Differential evolution, an evolutionary algorithm, was chosen to fulfill this requirement. An objective function was formulated, incorporating two models: an R_{on} model derived from previous research and the initially-created model for predicting the number of chips on a wafer. The objective of optimization was to increase the number of chips. The search was rewarded for producing chip geometry that matched the original R_{on} and maximized the number of chips on a wafer. Upon testing on existing market products, the optimizer successfully generated a model that increased chip production while minimally impacting the R_{on} value.

This concludes our efforts to streamline lithography in chip fabrication by optimizing chip quantity while maintaining a balance with R_{on}. By leveraging machine learning models and an evolutionary algorithm, this research provides a method to enhance the efficiency of semiconductor manufacturing processes.

Possible directions for future work are:

- This work lays the groundwork for the exploration of other manufacturing processes, potentially leading to new optimization strategies in semiconductor fabrication.
- The regression algorithm could be improved by collecting more features and data for training, thereby enhancing its prediction accuracy.
- The prediction model developed for the number of chips could be expanded to other manufacturing processes beyond lithography, offering a broader applicability.
- Including other electrical and thermal parameters, apart from R_{on}, in the optimization problem could offer a more comprehensive view of the trade-offs in chip fabrication.

Appendix **A**

Appendix

A.1 Code

The code for the complete thesis can be found in the repo at:

https://github.com/junaidahmad17/Chip-Geometry-Optmizer.git

To run the code:

- Install python.
- Install and activate python virtual environment.
- Install requirements by executing: **pip install -r requirements.txt**.
- Set the test and train files path in the **.env** folder.
- Run **paper.ipynb** notebook cells to train different regression models.
- Set the chip bounds and the path for the selected model inside the environment file.
- Execute the command: **python app_wafer_pred.py**.
- Go to **http://localhost:8080** inside the browser and input all the required values and click optimize.
- For reticle layout and design optimization run: **python app_wafer_pred.py** and go to **http://localhost:8081** and repeat the same.
- For geometry based predictor run: **python geom_wafer.py** and input the values.

Bibliography

[1] Julia Dragon Ondrej Burkacky and Nikolaus Lehmann. “The semiconductor decade: A trillion-dollar industry”. In: *mckinsey* (2022). URL: https://www.mckinsey.com/industries/semiconductors/our-insights/the-semiconductor-decade-a-trillion-dollar-industry.

[2] Peter Clarke. “Lithography equipment in short supply through 2023 says ASML”. In: *eeNews Europe* (2022). URL: https://www.eenewseurope.com/en/lithography-equipment-in-short-supply-through-2023-says-asml/.

[3] Toby Sterling. “Intel orders ASML system for well over USD 340 mln in quest for chipmaking edge”. In: *Reuters* (2022). URL: https://www.reuters.com/technology/intel-orders-asml-machine-still-drawing-board-chipmakers-look-an-edge-2022-01-19/.

[4] Paul van Gerven. “EUV for dummies”. In: *BitsChips* (2017). URL: https://bits-chips.nl/artikel/euv-for-dummies/.

[5] Warren Flack et al. “Lithography technique to reduce the alignment errors from die placement in fan-out wafer level packaging applications”. In: *Proceedings - Electronic Components and Technology Conference* (May 2011), pp. 65–70. DOI: 10.1109/ECTC.2011.5898493.

[6] Sara Brown. “Machine learning, explained”. In: *MIT Sloan* abs/1709.09480 (2021). URL: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained.

[7] C. C. Emioma and S. O. Edeki. “Stock price prediction using machine learning on least-squares linear regression basis”. In: *Journal of Physics: Conference Series*

1734.1 (Jan. 2021), p. 012058. DOI: 10.1088/1742-6596/1734/1/012058. URL: https://dx.doi.org/10.1088/1742-6596/1734/1/012058.

[8] Padraig Cunningham, Matthieu Cord, and Sarah Delany. "Supervised Learning". In: Jan. 2008, pp. 21–49. ISBN: 978-3-540-75170-0. DOI: 10.1007/978-3-540-75171-7{_}2.

[9] Louis Capitaine, Robin Genuer, and Rodolphe Thiébaut. "Random forests for high-dimensional longitudinal data". In: (2019). arXiv: 1901.11279 [stat.ME].

[10] Jitendra Kumar Jaiswal and Rita Samikannu. "Application of Random Forest Algorithm on Feature Subset Selection and Classification and Regression". In: *2017 World Congress on Computing and Communication Technologies (WCCCT)*. 2017, pp. 65–68. DOI: 10.1109/WCCCT.2016.25.

[11] Tianqi Chen and Carlos Guestrin. "XGBoost". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Aug. 2016. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145%2F2939672.2939785.

[12] John Gerald Taylor. "Neural networks and their applications". In: (1996).

[13] Ameya Jagtap and George Karniadakis. "How important are activation functions in regression and classification? A survey, performance comparison, and future directions". In: (Sept. 2022). DOI: 10.48550/arXiv.2209.02681.

[14] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted Boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[15] Qi Deng, Yi Cheng, and Guanghui Lan. *Optimal Adaptive and Accelerated Stochastic Gradient Descent*. 2018. arXiv: 1810.00553 [stat.ML].

[16] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2017. arXiv: 1609.04747 [cs.LG].

[17] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization." In: *Journal of machine learning research* 13.2 (2012).

[18] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. “Practical Bayesian Optimization of Machine Learning Algorithms”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012. URL: `https://proceedings.neurips.cc/paper_files/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf`.

[19] David E. Goldberg. “1”. In: *Genetic algorithms in search, optimization, and machine learning*. Pearson, 1989.

[20] July 2022. URL: `https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc#:~:text=An%20ROC%20curve%20(receiver%20operating,False%20Positive%20Rate`.

[21] Staf Verhaegen, Stefan Cosemans, Mircea Dusa, Pol Marchal, Axel Nackaerts, Geert Vandenberghe, and Wim Dehaene. “Litho variations and their impact on the electrical yield of a 32nm node 6T SRAM cell”. In: *Design for Manufacturability through Design-Process Integration II*. Ed. by Vivek K. Singh and Michael L. Rieger. Vol. 6925. International Society for Optics and Photonics. SPIE, 2008, 69250R. DOI: `10.1117/12.773333`. URL: `https://doi.org/10.1117/12.773333`.

[22] Gary S. May and Costas J. Spanos. “Yield Modeling”. In: *Fundamentals of Semiconductor Manufacturing and process control*. IEEE, 2006.

[23] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning: With applications in R*. Springer, 2022.

[24] Siby Jose Plathottam, Arin Rzonca, Rishi Lakhnori, and Chukwunwike O. Iloeje. “A review of artificial intelligence applications in manufacturing operations”. In: *Journal of Advanced Manufacturing and Processing* n/a.n/a (2023), e10159. DOI: `https://doi.org/10.1002/amp2.10159`. eprint: `https://aiche.onlinelibrary.wiley.com/doi/pdf/10.1002/amp2.10159`. URL: `https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/amp2.10159`.

[25] Jp Colinge. *Silicon-on-Insulator Technology: Materials to VLSI*. Jan. 2004. ISBN: 978-1-4757-2613-8. DOI: `10.1007/978-1-4757-2611-4`.

[26] G. Nicolae, Andi Buzo, C. Feuerbaum, C.V. Diaconu, Horia Cucu, G. Pelz, and C. Burileanu. “Metamodel-based prediction of On Resistance for microelectronic power switches”. In: Dec. 2021, pp. 1–3. DOI: `10.1109/EDAPS53774.2021.9656996`.

[27] Bo Liu, Slawomir Koziel, and Qingfu Zhang. "A Multi-Fidelity Surrogate-Model-Assisted Evolutionary Algorithm for Computationally Expensive Optimization Problems". In: *Journal of Computational Science* 12 (Nov. 2015). DOI: `10.1016/j.jocs.2015.11.004`.

[28] Y.K. JAIN and S.K. BHANDARE. "Min Max normalization based data perturbation method for privacy protection". In: *International Journal of Computer and Communication Technology* (2013).

[29] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* 2015. arXiv: `1502.03167 [cs.LG]`.

[30] J. Sola and J. Sevilla. "Importance of input data normalization for the application of neural networks to complex industrial problems". In: *IEEE Transactions on Nuclear Science* 44.3 (1997), pp. 1464–1468. DOI: `10.1109/23.589532`.

[31] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. "Linear Regression". In: *An introduction to statistical learning: With applications in R.* Springer, 2022.

[32] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32. ISSN: 1573-0565. DOI: `10.1023/A:1010933404324`. URL: `https://doi.org/10.1023/A:1010933404324`.

[33] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* Adaptive Computation and Machine Learning series. MIT Press, 2016. ISBN: 9780262337373. URL: `https://books.google.de/books?id=omivDQAAQBAJ`.

[34] Rainer Storn and Kenneth Price. "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces". In: *Journal of Global Optimization* 11.4 (Dec. 1997), pp. 341–359. ISSN: 1573-2916. DOI: `10.1023/A:1008202821328`. URL: `https://doi.org/10.1023/A:1008202821328`.

[35] Swagatam Das and Ponnuthurai Suganthan. "Differential Evolution: A Survey of the State-of-the-Art". In: *IEEE Trans. Evolutionary Computation* 15 (Jan. 2011), pp. 4–31.

[36] Dilber Uzun Ozsahin, Mubarak Taiwo Mustapha, Auwalu Saleh Mubarak, Zubaida Said Ameen, and Berna Uzun. "Impact of feature scaling on machine learning models for the diagnosis of diabetes". In: *2022 International Conference on Artificial Intelligence in Everything (AIE)*. 2022, pp. 87–94. DOI: 10.1109/AIE57029.2022.00024.