People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University M'Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering
Department of Electronics

Final Year Project Report Presented in Partial Fulfilment of the Requirements for the Degree of

MASTER

In **Electronic**

Option: **Computer Engineering**

Entitled

Optimized AI-based Real-time State of Charge (SOC) Estimation of Lithium-ion Batteries

Presented by

- **Mr. BOUCHIKH Mohamed Amin**
- **Mr. TEDJANI Mohammed Ridha**

Supervisor

- **Dr. TOUZOUT Walid**

Registration Number:……../2024

This modest work is dedicated to our beloved parents and family

For their endless love, support and encouragement

Abstract

Lithium-ion (Li-ion) batteries are highly valued for their ability to extend battery lifespan and enhance power energy density due to their chemical properties. The battery State of Charge (SOC) is a crucial parameter for monitoring battery health and estimating its lifespan, indicating how much longer the battery can be used and when it needs to be charged. Therefore, accurate SOC predictions are essential to prevent overcharging or over-discharging, and can be determined using conventional methods or data-driven approaches. In this report, we present a novel approach for SOC estimation of Li-ion batteries using optimized machine learning-based SOC estimation in both charging and discharging modes. The models are trained, tested, and optimized using a prognostic Li-ion battery dataset provided by the National Aeronautics and Space Administration (NASA) with five main inputs: load voltage, load current, measured voltage, measured current, and primarily battery temperature, the report is key for preferring data-driven approaches over conventional methods. Initially, nine different methods were evaluated: LASSO, K-Neighbor Regressor, CAT Boost Regressor, Extra Tree Regressor, Random Forest Regressor, XGB Regressor, Decision Tree Regressor, and Gradient Boosting Regressor, These methods were assessed in terms of their accuracy and the evaluation metrics R2, MAE, RMSE, with Four approaches showing promising results according to state-of-the-art applications namely: Extra Tree, XGB,DTR and Gradient Boot Regressors. Moreover, the novelty of this report involves hyperparameter tuning, including learning rate, maximum tree depth, number of trees and more, to find the optimal parameters for the three best models. Thus, GridSearchCV optimization method demonstrated significant improvement in terms of model evaluation metrics. Finally, the best approach (Extartree regressor) for charging and discharging was deployed onto an ESP32 microcontroller with an OLED interconnected with current, voltage, and temperature sensors for real-time battery SOC display and monitoring.

Acknowledgments

First and Foremost, we would like to thank Allah, the Most Gracious, the Most Merciful. Without His guidance, wisdom, and blessings, none of this would have been possible. Through the highs and lows, the challenges and triumphs, it was His infinite mercy and guidance that provided us with the strength, patience and perseverance to see this journey through and this humble work achieved, Alhamdulillah.

Warmest thanks to our supervisor, Dr. TOUZOUT Walid, who made this work possible. His guidance and advice carried us through all the stages of writing this project. We are deeply grateful for his patience, insights and dedication, which have been instrumental in the successful completion of this thesis.

We would like to extend our deepest gratitude to to our parents and family, whose endless love, prayers, support, caring and encouragement have been the bedrock of our academic journey.

Lastly, we express our heartfelt gratitude to all the professors and colleagues who assisted us, as well as to everyone who contributed, both directly and indirectly, to the completion of this work.

Contents

List of Figures

List of Tables

Introduction

Lithium-ion (Li-ion) batteries are rechargeable power sources in a wide range of modern devices which they stand out with their lightweight and compact design and offer the highest energy density on the market. Their ability to extend battery lifespan and enhance power energy density due to their chemical properties have made them the most popular and the predominant commercial rechargeable battery chemistry used today. This has led to their widespread use in mobile computing devices such as smartphones and smartwatches, as well as in automotive systems like hybrid and electric vehicles.

However, lithium batteries are prone to damage from extreme temperatures, overcharging or over-discharging. these conditions can significantly reduce battery lifespan and increase the risk of safety hazards. To ensuring battery safe usage, maximizing performance, extending battery life, improve user experience and accomplish effective energy management, an essential parameter in the energy storage devices called State of Charge (SOC) should be accurately monitored and maintained [1]. SOC indicates how much longer the battery can be used and when it needs to be charged.

There has always a significant concern to develop an efficient and reliable methods/algorithms to accurately estimate the State of Charge of Li-ion batteries. SOC estimation is a challenging task; it's a non-linear function of temperature and current and voltage. Several SOC estimation approaches have been proposed, each with its own set of issues and challenges such as non-linear behavior, aging, dynamic usage conditions and mainly temperature effects [2]. For this report, five approaches was presented: conventional, adaptive filter algorithm, data-driven (learning algorithm), nonlinear observer and others.

Data-driven techniques which involve machine learning (ML) or deep Learning (DL) for SOC estimation offer significant advantages over conventional and statistical methods for SOC estimation as they excel in handling complexity, adapting to various conditions, input-output flexibility, improving accuracy with large datasets, and providing robust, scalable and real-time performance [3]. These benefits make ML approaches particularly suitable for modern and data-rich applications in SOC prediction. Although DL-based SOC estimation offers significant advantages over ML-based approaches in terms of handling complex nonlinear relationships and au-

tomatic feature extraction [4], the deployment can be more challenging, requiring more computational resources and infrastructure for ongoing model maintenance and updates, especially for real-time systems [5–7]. Therefore, ML-based approaches can address this issue by offering distinct advantages in terms of simplicity, interpretability, computational efficiency, and ease of use and deployment onto real-time systems [7–9]. Furthermore, optimizing machine learning models for SOC estimation leads to better performance, safety, and user experience while enhancing the overall efficiency and lifespan of battery-powered systems [10, 11]. As battery technology continues to evolve, the need for precise and reliable SOC estimation becomes even more critical [12], making optimization an essential aspect of battery management by selecting the right features, tuning hyperparameters, choosing appropriate algorithms, etc. These strategies enhance the model's performance, robustness, and generalization ability, making it better suited for practical applications, mainly battery SOC estimation.

In this report, we present a novel approach for SOC estimation of 2000∼1400-mAh Li-ion batteries using optimized machine learning in both charging and discharging modes. The models were trained, tested and optimized using prognostic Li-ion battery dataset provided by the National Aeronautics and Space Administration (NASA) [13], with five main inputs: load voltage, load current, battery voltage (or measured voltage), battery current (measured current) and primarily battery temperature, the latter being key for preferring data-driven approaches over conventional methods. Initially, nine different ML-based methods were evaluated: Linear Regression (LR), Least Absolute Shrinkage and Selection Operator (LASSO), K-Nearest Neighbor Regressor (KNNR), Categorical Boosting Regressor (CatBoostR), Extra Trees Regressor (ETR), Random Forest Regressor (RFR), Extreme Gradient Boosting Regressor (XGBR), Decision Tree Regressor (DTR), and Gradient Boosting Regressor (GBR). These methods were assessed in terms of their accuracy and the evaluation metrics R-squared (R^2), Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). Among these, Four approaches showed promising results according to state-of-the-art applications: ETR,XGBR,DTR and GBR. The novelty of this report involves hyperparameter tuning, including learning rate, maximum tree depth, number of trees and more, to find the optimal parameters for the three best models. The Grid-Search Cross-Validation (GridSearchCV) optimization method demonstrated significant improvement in terms of model evaluation metrics. Additionally, the nine ML models were retrained and reevaluated with the same dataset but without temperature input, resulting in less accurate results, as demonstrated in the results chapter of this report, proving the advantage of ML-based methods over conventional methods. Finally, the best approach for charging and discharging (Extra tree regressor) was deployed onto an ESP32 OLED V3.0 microcontroller interfaced with current, voltage, and temperature sensors for real-time battery SOC display and monitoring. To address all these issues effectively, this report is organized as follows: Chapter 1 delves into the theoretical background required to understand the presented methodology and interpret the results, As well as reviews related works on state-of-charge estimation, detailing

their features, structures and drawbacks. Chapter 2 outlines the workflow, methodology, system configurations, data preparation and processing, and detailed models. Finally, Chapter 3 discusses the results.

Chapter 1

Theoretical Background

Introduction

In the realm of energy storage, understanding the State of Charge (SOC) of lithium-ion batteries is essential for their efficient utilization and management. SOC represents the amount of energy remaining in a battery, crucial for determining its performance and lifespan. Accurate SOC estimation helps prevent problems like overcharging or completely draining the battery, both of which can shorten its lifespan.

To set the stage for the subsequent exploration and implementation, this chapter provides the theoretical background needed to understand the SOC and the vast methods used to estimate it. We will start by explaining what SOC is and then explore the different approaches for SOC estimation before expand on the conventional methods, such as Open Circuit Voltage (OCV) and Electrochemical Impedance Spectroscopy (EIS). Due to the lack real-time accuracy for the traditional methods, we will delve into the realm of machine learning for real-time SOC estimation and introduce the basics of machine learning, discussing different types of learning and various algorithms, and how to evaluate our models. Finally, we will cover the basics of optimization, focusing on how we can fine-tune machine learning models to get the best performance.

1.1 SOC Background

1.1.1 State of charge (SOC)

SOC estimation is a challenging task; it's a non-linear function of temperature, current and voltage. There has always been a big concern to estimate the SOC for all energy storage devices. SOC estimation with high accuracy not only gives us information about remaining useful energy, but also it evaluates the reliability of batteries. In addition, an accurate and efficient SOC estimation gives an idea about charging/discharging strategies, which have a significant impact on battery application where each cell may have different capacities due to aging, temperature, self-discharge and manufacture difference.

Several methods to estimate SOC have been introduced since the 1980s, however, a proper definition has yet to explain as the understanding of SOC needs further analytical tasks, such as prediction of remaining useful life and estimation of capacity. The most classical method to estimate SOC is current integration, which expresses the ratio of the available current capacity to the nominal capacity [14] is shown in Equation 1.1.

$$SOC = SOC_{init} - \frac{\int idt}{C_n} \tag{1.1}$$

where SOC_{init} is the initial State of Charge of the battery which represents the SOC at the start of the time period being considered, i is the battery current; C_n is the nominal capacity; t is time. Due to variation in external load and the internal chemical

reaction of the battery, the nominal capacity decreases gradually over time, which leads to non-stationary, non-linear battery degradation characteristics. In addition, large SOC errors may occur due to accumulation in terminal measurements, thus need to recalibrate the value from time to time [15]. Another way to define SOC with the effect of coulombic efficiency is expressed in Equation 1.2.

$$SOC = SOC_{init} - \frac{\int i.\eta dt}{C_n} \tag{1.2}$$

where η is the coulombic efficiency defined as the ratio of energy require for charging to the discharging energy needed to regain the original capacity.

1.1.2 SOC estimation methods

Several methods exist for determining the State of Charge (SOC) of batteries or their Remaining Useful Life (RUL), which can be classified into five categories or approaches as presented in the paper referenced [16]: empirical (conventional), adaptive filter algorithm, data-driven (learning algorithm) using artificial intelligence (AI), nonlinear observer and others, as shown in Fig 1.1. Selecting the most appropriate method depends on the specific application requirements and battery characteristics. The conventional method uses the physical properties of the battery, which includes voltage, charging/discharging current, resistance, and impedance. The adaptive filter algorithm uses various models and algorithms to calculate the SOC. The learning algorithm requires a large amount of training data and heavy computation to describe the nonlinear characteristics of lithium-ion to estimate the SOC. The nonlinear observer is designed to handle with the highly nonlinear system. The other methods include MARS, BI, IR and hybrid method. MARS, BI, and IR use extended linear model, two linear interpolations, and linear time invariant system respectively. The hybrid method combines two or three SOC algorithms to estimate SOC. It takes the advantage of each method to obtain optimal performance, which improves the estimation accuracy.

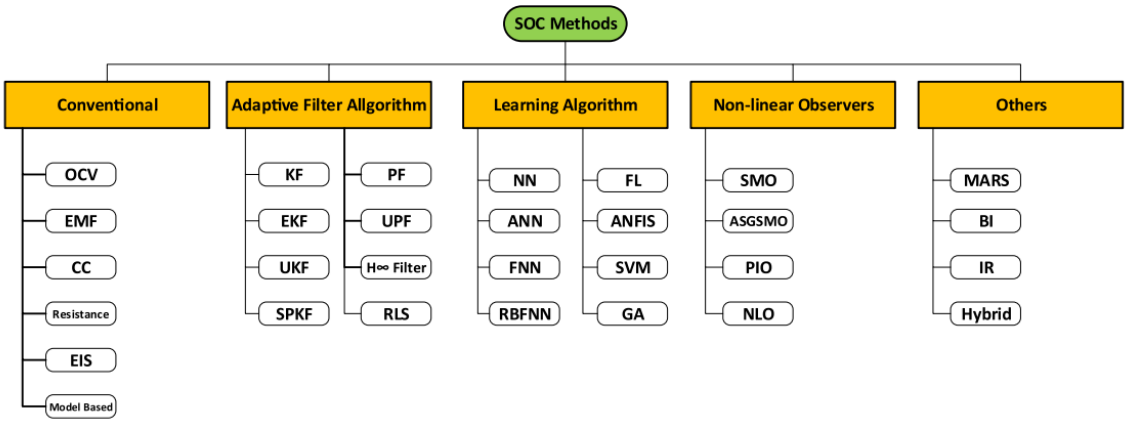


Figure 1.1: Classification of SOC estimation methodologies [16].

Conventional method

1. **Open Circuit Voltage (OCV) method**

Since SOC in a lithium-ion battery is connected to embedding quantity in the active material, an open circuit voltage can be considered to estimate SOC after the battery gets sufficient resting to reach balance [17]. Usually, an approximate linear relationship exists between SOC and OCV. However, the relationship between SOC and OCV is not exactly same for all types of batteries. The relationship depends on capacity and material of the battery [18]. For instance, a lead-acid battery has a linear relationship between SOC and OCV while a lithium-ion battery does not hold that relationship [19].

It is a simple method and has high precision. However, the main drawback of OCV method is that it takes long rest time to reach equilibrium condition [20]. The duration of time to reach from operating state to stable state depends on SOC states, temperature and so on. For instance, at low temperature, $C/LiFePO_4$ takes more than two hours to reach equilibrium. Furthermore, careful observations are required to measure the charge and discharge voltage since batteries have hysteresis characteristics which result in high OCV when the battery is charged and low OCV when the battery is discharged [21], as shown in Fig 1.2.

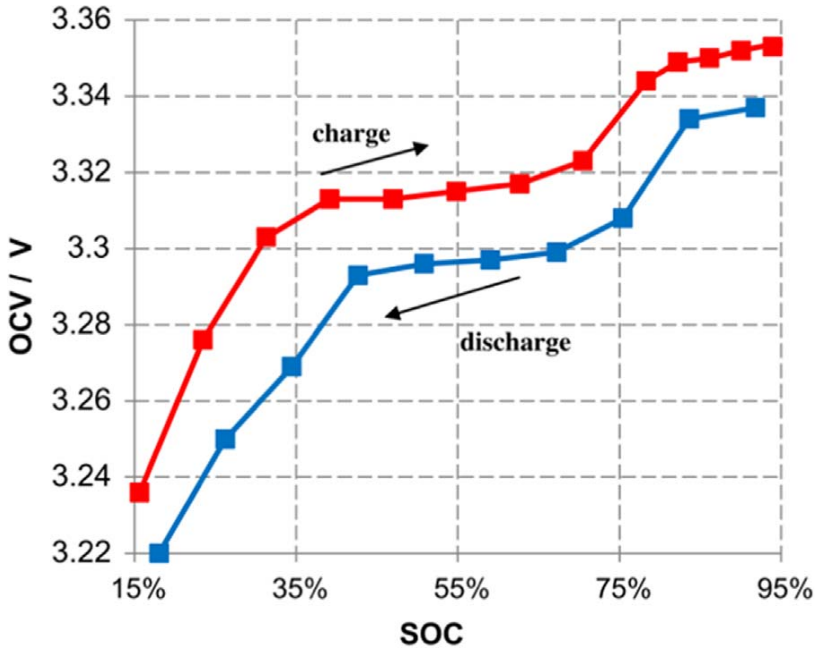


Figure 1.2: OCV/V vs SOC charge and discharge profile of $C/LiFePO_4$ battery tested under 25 °C, for 3 h [16].

2. **Electromotive Force (EMF) method**

The relationship between battery EMF and SOC is widely used for determination of battery capacity as shown in Fig 1.3. Battery EMF can be measured as an equivalent to OCV in equilibrium condition when the substantial time has passed after the current interruption takes place. Several researchers have used OCV relaxation to observe and predict EMF. OCV relaxation process occurs when the battery gets charged and discharged with frequent current disruption, as shown in Fig 1.4. The OCV relaxation may need several hours to reduce the effect of diffusion overvoltages.

In [22], adaptive methods were developed to model OCV relaxation process using EMF and exponential functions. The aim is to observe the OCV relaxation when the current is interrupted each time. The benefit of this model is its simplicity to determine parameters; nevertheless, it predicts the relaxation process inaccurately. Waag and Sauer [23] proposed an advanced adaptive approach to compare the online fitting of an OCV relaxation model with the measured OCV relaxation curve. This model has an equivalent circuit consisting of a voltage source (represents the EMF) in series with the resistances connected in parallel and a constant phase element (CPE). EMF is estimated depending on the fitting model parameters. The new values of the voltage and current of a battery are measured at each time. Once battery current reaches to zero, the algorithm to measure EMF is started. The data for battery open circuit voltage is stored as long as the battery current remains zero. When OCV samples get sufficient number, it is fitted to the measured EMF. As a result, all four model parameters (EMF, ΔOCV, impedance, weighting factor) are determined.

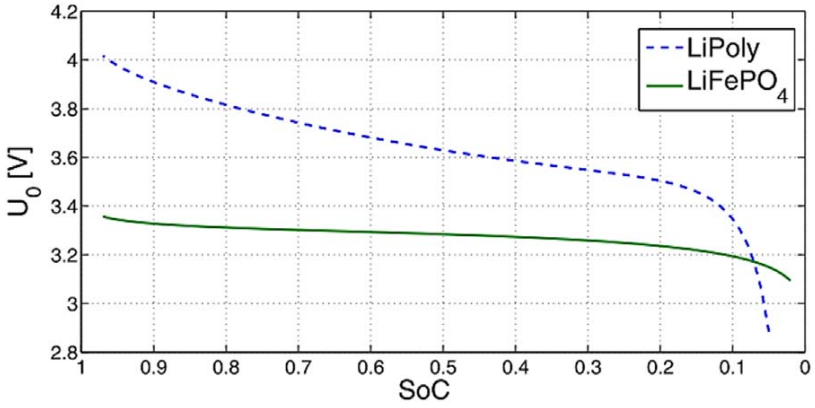


Figure 1.3: EMF voltage U_0 as function of SOC [16].

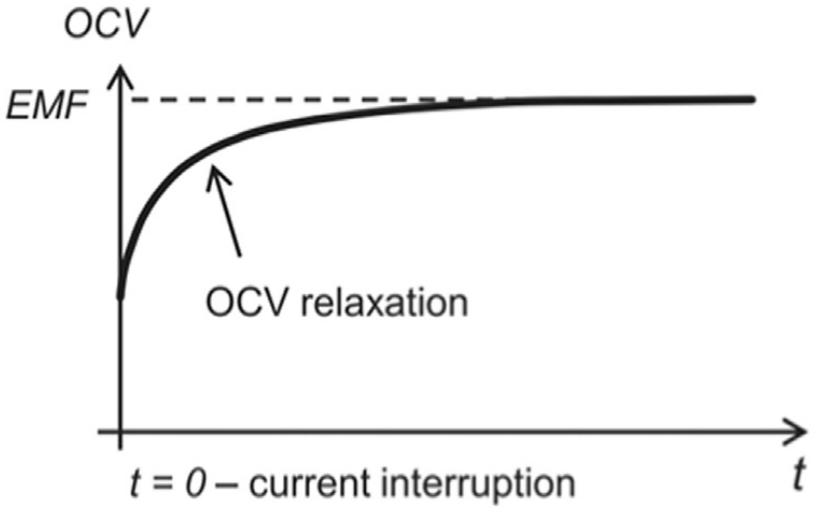


Figure 1.4: Voltage relaxation after the battery is discharged and the current is switched off [16].

3. **Coulomb Counting method**

   Coulomb counting method is the easiest method to estimate the battery SOC. The method is easy to implement with low power computation. It is based on the integration of battery current with respect to time while the battery is charging/discharging. The mathematical expression to measure SOC is denoted in Equation 1.2. Nevertheless, it is an open-loop algorithm and could result in significant inaccuracies due to uncertain disturbances and variables such as noise, temperature, current, etc. Also, there are difficulties in determining the initial value of SOC which causes a cumulative effect [24]. In addition, the estimation accuracy depends highly on the current sensors used which may be affected by measurement error, which also result in cumulative effect [25]. Furthermore, the method needs complete discharge of the cell and periodic capacity calibration to obtain maximum capacity, which shorten the battery lifespan [26].

4. **Internal resistance method**

   The method uses battery voltage and current to measure the internal resistance of the battery. Voltage is measured with the variation of current change during small duration (< 10 ms). The ratio of voltage and current variation results in DC resistance, which represents the capacity of the battery in DC. A small interval less than 10 ms is needed not only to capture the ohmic effect, but also to reduce the effect of transfer reaction and acid diffusion [27]. However, the value of estimated resistance contains an error if the time is longer. In addition, the method has good adaptability as well as the high accuracy of SOC estimation only during the end period of discharging. Furthermore, due to its low value (in milliohm range), the accuracy to get internal resistance is

very hard to obtain. The internal resistance changes slightly with a wide range of SOC which is difficult to observe [28], as indicated in Fig 1.5. Due to this shortcoming, the DC internal resistance is hardly used to estimate SOC.

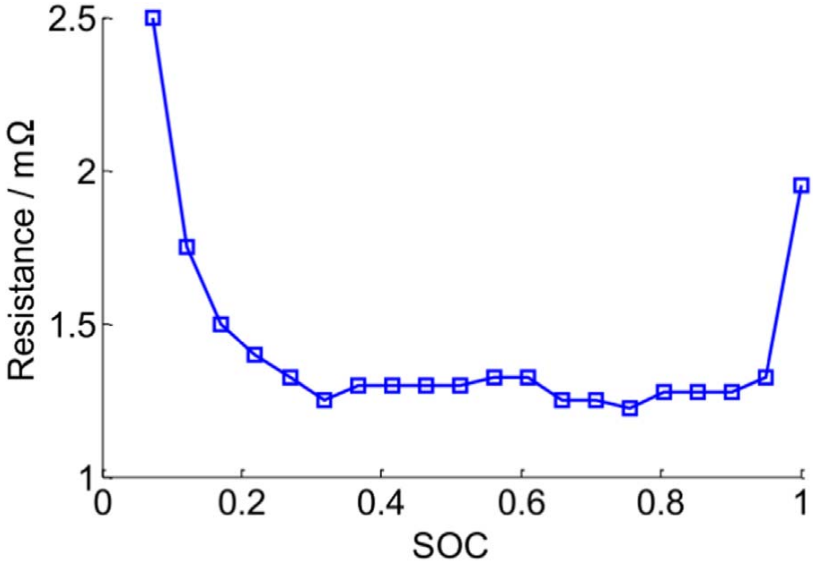


Figure 1.5: Variation of internal resistance with respect to SOC in lithium ion batteries (tested under 25 °C, using the Hybrid Pulse Power Characterization (HPPC) test procedure) [16].

5. **Electrochemical Impedance Spectroscopy (EIS)**

   EIS has been extensively used in order to obtain an understanding of the electrochemical reactions occurred inside the batteries and for determination of SOC. A proper electrochemical model is necessary to implement EIS. Then, EIS estimates the battery impedance using inductances and capacitances over a wide range of frequencies [29]. Ran et al. [30] established an equivalent circuit which included an inductive arc operated at high-frequency and two capacitive arcs operated at low-frequency. A non-linear least-squares fitting method is used under a different state of charging values to calculate the model impedances. However, EIS results are difficult to reproduce if the system is not operated in a steady state condition. Coleman et al. [31] estimated battery EMF voltage using impedance, terminal voltage and discharge current under load. The approach has low cost, achieves good accuracy and can operate online if the value of impedance is updated with a normalized value. However, the influence of battery aging and temperature variation could differ the estimated results from the real values, which result in a lack of accuracy.

These conventional methods share some drawbacks in terms of their accuracy and efficiency compared to other approaches, this is because they do not involve some sig-

nificant parameters mainly: battery temperature, impedance, etc. As a consequence, advanced approaches should be carried out, such as the learning algorithm approach for the problem in hand by developing machine learning models, taking into account other parameters.

1.2 Fundamentals of Machine Learning

1.2.1 Overview of machine learning

Machine Learning is the science (and art) of programming computers so they can learn from data. (past experience to inform future decisions)

Here is a slightly more general definition:

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

—Arthur Samuel, 1959

And a more engineering-oriented one:

A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

—Tom Mitchell, 1997

For example, your spam filter is a Machine Learning program that can learn to flag spam given examples of spam emails (e.g., flagged by users) and examples of regular (nonspam, also called "ham") emails. The examples that the system uses to learn are called the training set. Each training example is called a training instance (or sample). In this case, the task T is to flag spam for new emails, the experience E is the training data, and performance measure P needs to be defined; for example, you can use the ratio of correctly classified emails. This particular performance measure is called accuracy, and it is often used in classification tasks [32].

1.2.2 Types of machine learning

Machine Learning systems can be classified according to the amount and type of supervision they get during training. There are four major categories: supervised learning, unsupervised learning, semi-supervised learning, and Reinforcement Learning.

However, the approach varies from one problem to another. We will explain each type briefly then expand on supervised learning along with its algorithms for relevance to the problem at hand (Estimation).

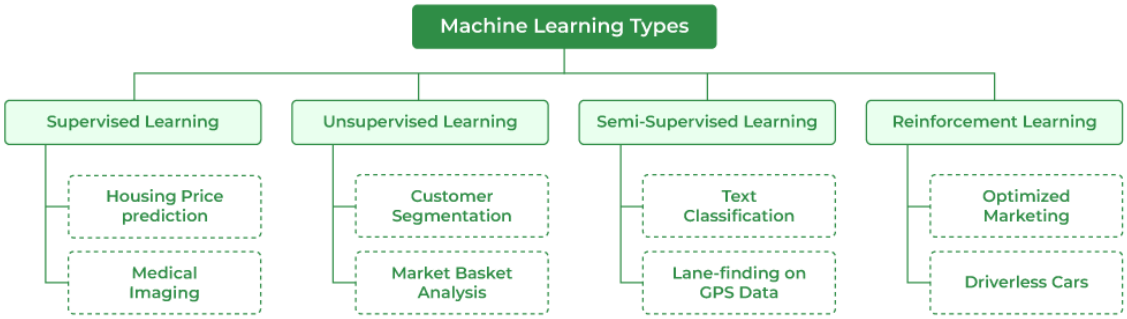


Figure 1.6: Types of machine learning [33].

1. **Supervised Learning:** In supervised learning, models are trained on a labeled dataset where both input and output parameters are provided. The algorithms learn to map points between inputs and correct outputs, making predictions based on the training data.

2. **Unsupervised Learning:** Unsupervised learning involves training models on an unlabeled dataset, where the machine predicts the output without any supervision. Models are trained with data that is neither classified nor labeled, allowing the machine to discover patterns and categories within the data.

3. **Semi-Supervised Learning:** Semi-supervised learning is a combination of supervised and unsupervised learning. It involves training models with a mix of labeled and unlabeled data, where the machine learns from both the labeled and unlabeled data to make predictions.

4. **Reinforcement Learning:** In reinforcement learning, agents learn from experiences without labeled data. The learning process is akin to how humans learn from experiences, where agents receive feedback in the form of rewards and punishments to optimize their actions in an environment.

These types of machine learning play crucial roles in various applications, from image recognition and natural language processing to autonomous systems and predictive analytics, shaping the future of technology and data-driven decision-making.

1.2.3 Supervised learning

Supervised machine learning is a type of machine learning where machines are trained using labeled training data, allowing them to predict outputs accurately. In this process, the training data consists of input data paired with correct output data, serving as a guide for the machine to learn and predict outcomes correctly. The labeled data acts as a supervisor, similar to a teacher guiding a student, helping the machine learn to associate inputs with the correct outputs.

The main goal of the supervised learning technique is to map the input variable(x) with the output variable(y). This approach is fundamental in building accurate machine learning models for various real-world applications Some real-world applications of supervised learning such as Risk Assessment, Fraud Detection, Spam filtering, etc. Supervised learning can be further categorized into two main types: regression and classification.

Regression

Regression is a type of supervised machine learning where the goal is to predict a continuous numerical output variable based on one or more input variables. The algorithm learns a function that maps the input features (also known as predictors, independent variables, or features) to the output variable (also known as the dependent variable or response variable).
Some key characteristics of regression problems:

- The output variable is a real-valued number, such as a person's height, a house's price, or a stock's price.
- The goal is to learn a function that can accurately predict the output given new input data.
- Regression algorithms try to minimize the error between the predicted output and the true output.

Regression methods comprise a diverse set of algorithms, including linear regression, polynomial regression, decision tree regression, random forest regression, and support vector regression, among others. Each algorithm has its own assumptions, advantages, and limitations, making them appropriate for various data types and problem areas.

Classification

Classification is a supervised learning method used to label a sample based on its features [34]. This technique assigns data instances to predefined classes or categories according to their characteristics. It is mainly used when the target variable has discrete or categorical outcomes. The goal of classification is to develop a model that can accurately categorize new, unseen instances into the correct classes [35].
The model analyzes the patterns and relationships among the input variables to create decision boundaries or rules that differentiate between different classes. These boundaries or limits are based upon some classification algorithms which will explore further.

Email spam detection, medical diagnostic tests, fraud detection, image classification and speech recognition are some of the common applications of classification. depending on the data's features, the problem's complexity, and the need for interpretability, a classification algorithm is selected.

1.2.4 Linear regression

Linear regression is a type of supervised machine learning algorithm that estimates the linear relationship between a scalar dependent variable (Y) and one or more independent variables (X) by fitting a linear equation to observed data. The goal of the algorithm is to find the best Fit Line equation that can predict the values (targets) based on the independent variables(features).

For a single independent variable, also known as Simple Linear Regression, the linear regression model can be represented as:

$$Y = \beta_0 + \beta_1 X + \epsilon \tag{1.3}$$

where:

- Y is the dependent variable
- X is the independent variable
- β_0 is the intercept
- β_1 is the slope
- ϵ is the error term

For multiple independent variables, also known as Multiple Linear Regression, the model extends to:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n + \epsilon \tag{1.4}$$

where:

- Y is the dependent variable
- $X_1, X_2, \ldots, X_n$ are the independent variables
- β_0 is the intercept
- $\beta_1, \beta_2, \ldots, \beta_n$ are the slopes
- ϵ is the error term

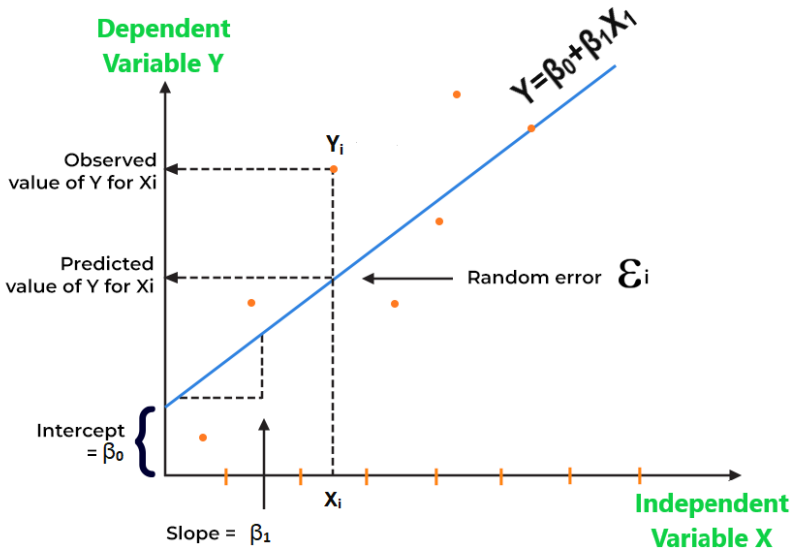


Figure 1.7: Visualization of Equation for Linear Regression [36].

The method aims to minimize the sum of the squared residuals (the differences between the observed and predicted values), effectively fitting the best possible line through the data points [34]. This approach is widely used due to its simplicity and interpretability, making it a popular choice for predicting outcomes and understanding the influence of predictor variables. However, its assumptions of linearity, independence, homoscedasticity, and normality must be met to ensure reliable results, and it can be sensitive to outliers and multicollinearity among the predictors.

Unlike Lasso regression, linear regression does not include any form of regularization, which can lead to overfitting, especially when there are many features, and it uses all provided features without any form of automatic feature selection.

1.2.5 Lasso regression

Least Absolute Shrinkage and Selection Operator Regression (simply called Lasso Regression) is a regularized version of Linear Regression. It aims to identify the variables and corresponding regression coefficients that lead to a model that minimizes the prediction error. This is achieved by imposing a constraint on the model parameters, which 'shrinks' the regression coefficients towards zero, that is by forcing the sum of the absolute value of the regression coefficients to be less than a fixed value [37].

Lasso adds a regularization term to the cost function, but it uses the ℓ_1 norm of the weight vector which encourages the shrinkage of some coefficients to exactly zero, effectively performing feature selection by excluding irrelevant features from the model, hence reducing overfitting. This makes Lasso regression particularly useful for

high-dimensional datasets where feature selection is necessary (see Equation 1.5).

Equation 1.5. Lasso Regression cost function

$$J(\theta) = MSE(\theta) + \alpha \sum_{i=1}^{n} |\theta_i| \tag{1.5}$$

where α is the hyperparameter (regularization parameter) that controls the model's complexity and performance. A higher α value increases the penalty on the absolute values of the coefficients (θ_1, θ_2,..., θ_n) resulting in more coefficients being shrunk to zero, effectively performing feature selection and reducing overfitting. Conversely, a lower α reduces the regularization effect, making the model more similar to standard linear regression, which can capture more data complexity but also risk overfitting. Selecting the optimal α typically involves cross-validation, grid search, and consideration of the bias-variance tradeoff to achieve a balance between model simplicity and predictive accuracy.

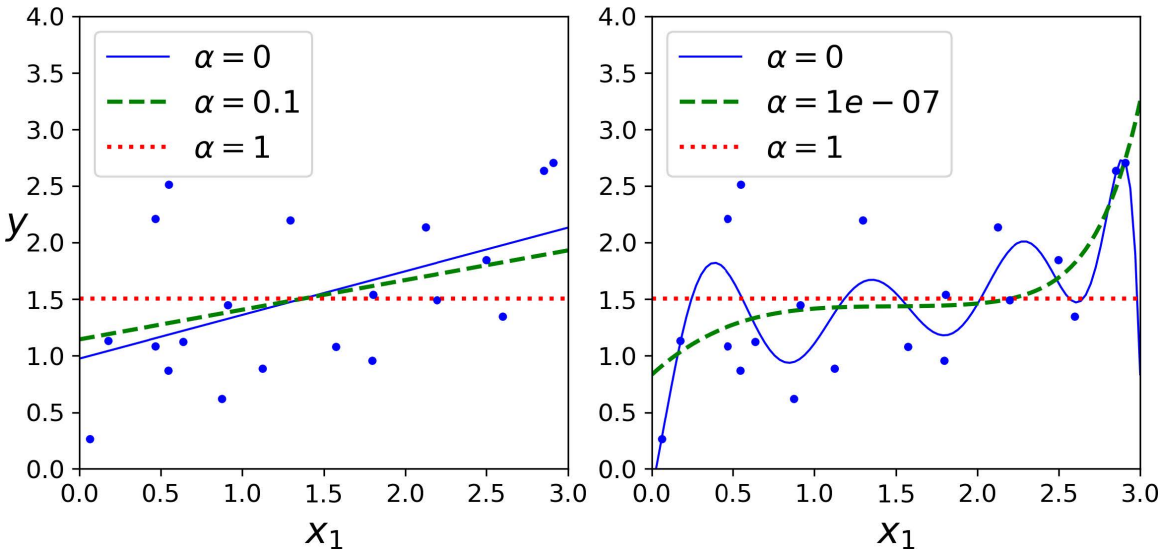


Figure 1.8: Lasso regression [32].

An important characteristic of Lasso Regression is that it tends to completely eliminate the weights of the least important features (i.e., set them to zero). For example, the dashed line in the right plot on Figure 1.8 (with $\alpha = 10^{-7}$) looks quadratic, almost linear: all the weights for the high-degree polynomial features are equal to zero. In other words, Lasso Regression automatically performs feature selection and outputs a sparse model (i.e., with few nonzero feature weights) [32]. By reducing model complexity and addressing multicollinearity, Lasso Regression can lead to more robust and interpretable models.

1.2.6 Decision trees

Decision trees are a popular machine learning method used for both classification and regression tasks. This approach builds a tree-like model where internal nodes represent feature tests, branches signify decision rules, and leaf nodes indicate the predicted class or value.

The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in a recursive manner called recursive partitioning. This flowchart-like structure helps you in decision-making. It is visualized like a flowchart diagram which easily mimics human level thinking. That is why decision trees are easy to understand and interpret [38].

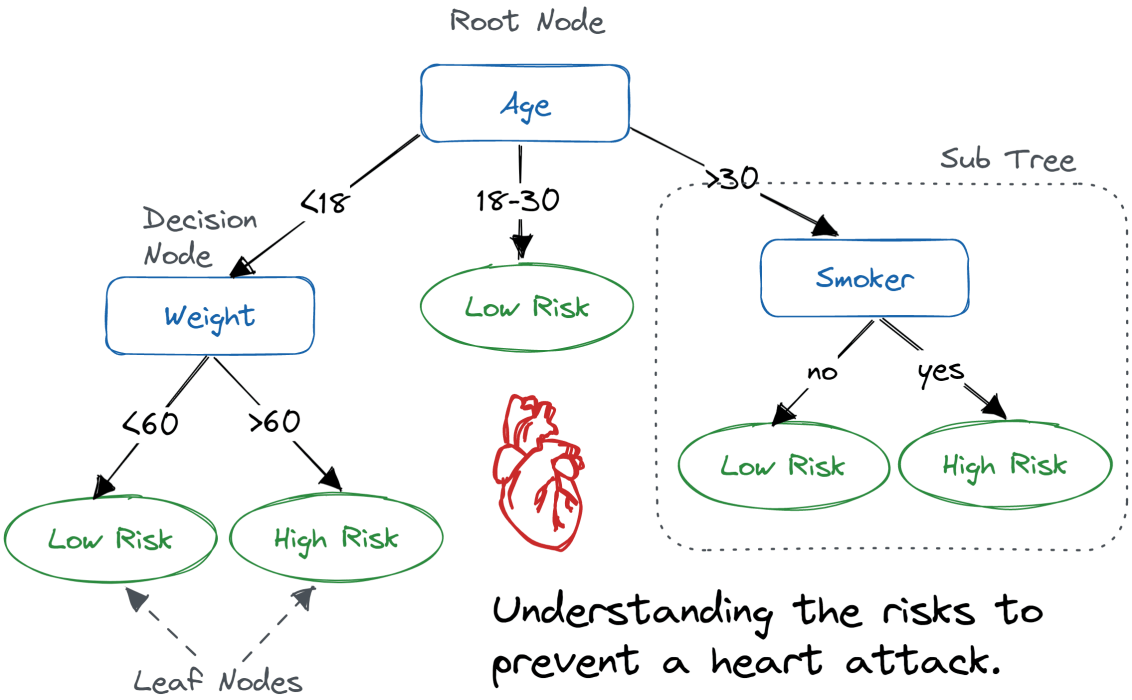


Figure 1.9: Decision tree example classification example of heart failure [38].

A decision tree is an easily understandable type of machine learning algorithm. It clearly shows how decisions are made, unlike neural networks, which are more complex and opaque. Additionally, decision trees train faster than neural networks [39].

The time complexity of decision trees is a function of the number of records and attributes in the given data. The decision tree is a distribution-free or non-parametric method which does not depend upon probability distribution assumptions. Decision trees can handle high-dimensional data with good accuracy [38].

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of the

root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node [40]. For the next node, the algorithm again compares the attribute value with the other sub-nodes and moves further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

1. Find the best attribute in the dataset using Attribute Selection Measure (ASM).
2. Make that attribute a decision node and break the dataset into smaller subsets.
3. Generate the decision tree node, which contains the best attribute.
4. Recursively make new decision trees using the subsets of the dataset created in step -2. Continue this process until a stage is reached where you cannot further classify the nodes and call the final node as a leaf node.
5. To stop the process one of these three conditions must be met:
    - All the tuples belong to the same attribute value.
    - There are no more remaining attributes.
    - There are no more instances.

The following schematic explains the process of making a decision tree algorithm:

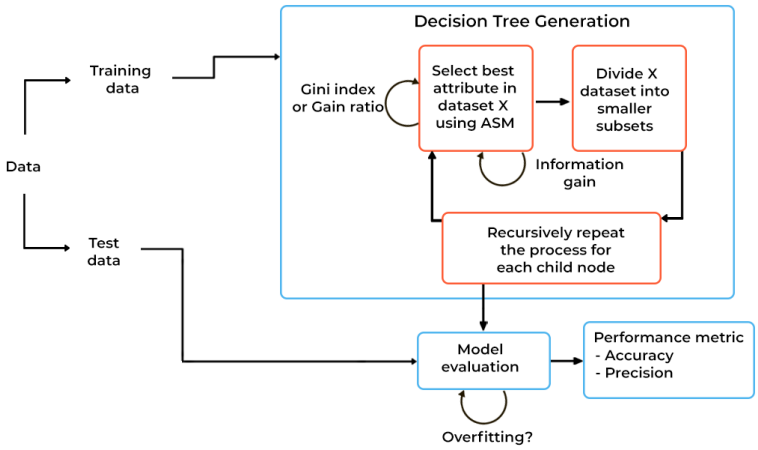


Figure 1.10: Broad view of training decision tree models [41].

The decision tree relies on multiples measures to do its attribute selection process otherwise known as attribute selection measures or ASM, the most critical measures are:

- **Information gain:** It represents the information gain of attribute a_i with respect to the dataset S. It is an impurity-based criterion that uses the entropy

measure (origin from information theory) as the impurity measure (s.t: En is the entropy).

$$infoGain(a_i, S) \;=\; En(y, S) \;-\; \sum_{v_i,j \in\, dom(a_i)} \frac{|\sigma a_i = v_{i,j} S|}{|S|} \,.\, En(y, \sigma a_i = v_{i,j} S) \tag{1.6}$$

Or simply:

$$infoGain(a_i, S) \;=\; En(S) \;-\; [\,(Wa) \times En(eachfeature)\,] \tag{1.7}$$

Such that:

$$Wa \;=\; WeightedAverage \;=\; \frac{|\sigma a_i = v_{i,j} S|}{|S|} \tag{1.8}$$

Where:

- a_i: is the i-th attribute (or feature) in the dataset. Attributes are used to split the data at each node in the decision tree.
- S: is the current dataset (or subset of the data) being evaluated for splitting.
- $En(y, S)$: is the entropy of the target variable y in the dataset S. Entropy is a measure of the impurity or disorder in the dataset with respect to the target variable. It quantifies the amount of uncertainty or randomness in the target variable.
- $dom(a_i)$: represents the domain (or set) of possible values that attribute a_i can take. For example, if a_i is a binary attribute, $dom(a_i)$ could be {0,1}.
- $v_{i,j}$: is the $j - th$ value in the domain of attribute a_i. For example, if a_i can take values {0, 1, 2}, then $v_{i,j}$ could be 0, 1, or 2.
- $\sigma a_i = v_{i,j} S$: is the subset of S that satisfies the condition $a_i = v_{i,j}$.
- $|\sigma a_i = v_{i,j} S|$: is the number of instances in the subset $\sigma a_i = v_{i,j} S$
- $|S|$: is the number of instances in the original dataset S
- $En(y, \sigma a_i = v_{i,j} S)$: is the entropy of the target variable y in the subset $\sigma a_i = v_{i,j} S$. It measures the impurity or disorder of the target variable within the subset of the data where a_i equals $v_{i,j}$.

- **Gini index:** Gini index is an impurity-based criterion that measures the divergences between the probability distributions of the target attribute's values. The Gini index has been used in various works such as (Breiman et al) [40] and (Gelfand et al., 1991) and it is defined as:

$$Gini(y, S) = 1 - \sum_{c_j \in dom(y)} \left(\frac{|\sigma y = c_j S|}{|S|}\right)^2 \tag{1.9}$$

In the case of a discrete-valued attribute, the subset that gives the minimum gini index for that chosen is selected as a splitting attribute. In the case of continuous-valued attributes, the strategy is to select each pair of adjacent values as a possible split point, and a point with a smaller gini index is chosen as the splitting point [38].

Consequently the evaluation criterion for selecting the attribute a_i is defined as:

$$GiniGain(a_i, S) = Gini(y, S) - \sum_{v_i, j \in dom(a_i)} \frac{|\sigma a_i = v_{i,j} S|}{|S|} . Gini(y, \sigma a_i = v_{i,j} S) \tag{1.10}$$

And thus, the attribute with the minimum Gini Index is chosen as the splitting attribute.

1.2.7 Random forests

Random Forests were introduced by Leo Breiman [42] who was inspired by earlier work by Amit and Geman [43]. Random Forests can be used for either a categorical response variable, (classification), or a continuous response, (regression). Similarly, the predictor variables can be either categorical or continuous [44]. In short, it is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting [45].

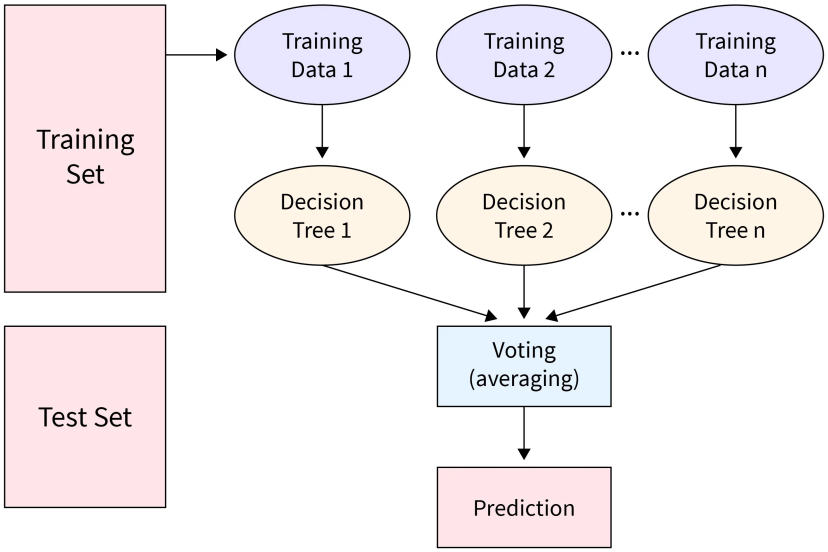


Figure 1.11: Ensemble learning for random forests [46].

Random forests work in few steps:

1. Data sampling: Random Forests randomly select subsets of the training data with replacement (known as bootstrap samples) to create multiple training sets.

2. Building decision trees: For each bootstrap sample, a decision tree is constructed by recursively partitioning the data based on different features and split points. The splitting is done by maximizing information gain or minimizing impurity measures like Gini index or entropy.

3. Random feature selection: At each node of the decision tree, a random subset of features is considered for determining the best split. This random feature selection helps to decorrelate the trees and reduce overfitting.

4. Tree ensemble: After building a set of decision trees, predictions from each tree are combined using majority voting for classification or averaging for regression. This ensemble approach improves the accuracy and generalization of the model.

5. Prediction: To make predictions on unseen data, the input data is passed through each decision tree in the forest, and the final prediction is obtained by aggregating the individual tree predictions.

Random forests are capable of performing both Classification and Regression tasks and are capable of handling large datasets with high dimensionality, while its ensemble learning nature helps enhance its accuracy and combat overfitting.

1.2.8 Boosting

Boosting is a powerful ensemble technique used in machine learning to improve the performance of weak learners (models that are slightly better than random guessing) by combining them to form a strong learner. The primary idea behind boosting is to train weak learners sequentially, each focusing more on the mistakes made by the previous ones, thereby reducing errors and improving accuracy. There are several types of boosting algorithms, including Adaptive Boosting (AdaBoost), Gradient Boosting, and LogitBoost. Each type has its own strengths and weaknesses.

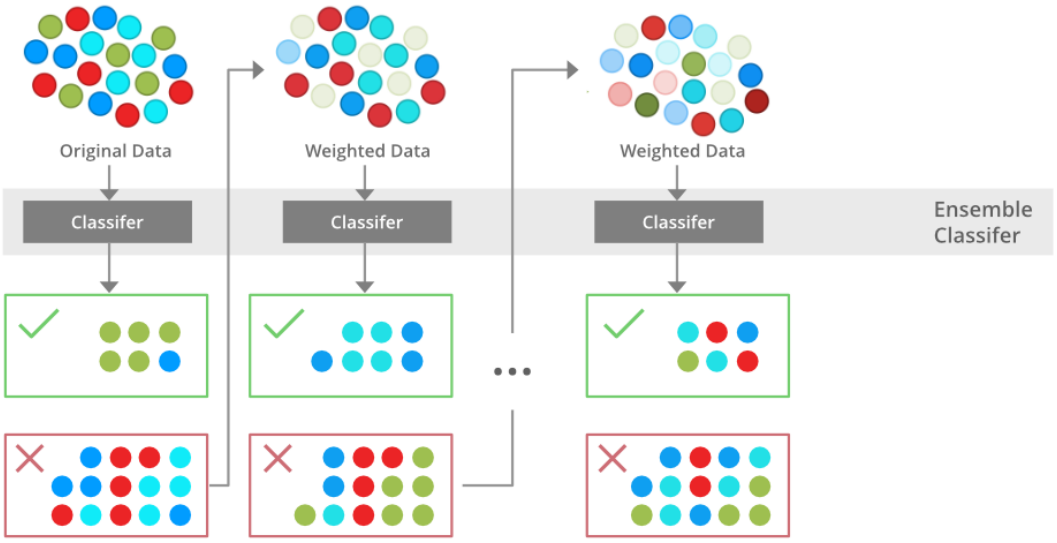


Figure 1.12: Training a boosting model [47].

Advantages of Boosting

- Improved Accuracy – Boosting can improve the accuracy of the model by combining several weak models' accuracies and averaging them for regression or voting over them for classification to increase the accuracy of the final model.
- Robustness to Overfitting – Boosting can reduce the risk of overfitting by reweighting the inputs that are classified wrongly.
- Better handling of imbalanced data – Boosting can handle the imbalance data by focusing more on the data points that are misclassified
- Better Interpretability – Boosting can increase the interpretability of the model by breaking the model decision process into multiple processes.

1.2.9 K-nearest neighbor (KNN)

K-Nearest Neighbors (KNN) is a powerful algorithm that can be used for both classification and regression tasks. In KNN, the class or value of an unseen data point is determined by considering its k nearest neighbors in the training set [35]. The algorithm operates by calculating the distance between the new data point and all other data points in the training set using a distance metric such as Euclidean distance. The k nearest neighbors are then selected based on the shortest distance [48]. For classification tasks, the class label of the new data point is determined through majority voting among its k nearest neighbors. In the context of regression, KNN is often referred to as "K-Nearest Neighbors Regression" or "KNN Regression". It is a simple and intuitive algorithm that makes predictions by finding the K nearest data points to a given input and averaging their target values [49].

Here is an overview of how KNN Regression works [50]:

1. Data Collection: Starting with a dataset that includes both input features and target values. In regression tasks, the target values are continuous and represent the output you want to predict.

2. Choosing the Number of Neighbors (K): The number of nearest neighbors, K, that will be used to make predictions should be chosen properly. This is a hyperparameter that you can tune based on the characteristics of the data. A small K (e.g., 1 or 3) may lead to noisy predictions, while a large K may lead to overly smoothed predictions.

3. Distance Metric: KNN relies on a distance metric (e.g., Euclidean distance) to measure the similarity between data points. Different distance metrics can be used depending on the nature of the data.

4. Prediction: To predict for a new input data point, KNN calculates the distance between this point and all other data points in the dataset. It then selects the K data points with the smallest distances.

5. Regression Prediction: For regression, the predicted value for the new data point is the average of the target values of the K nearest neighbors. This could be a simple arithmetic mean.

KNN Regression is simple to implement and understand, but it can be computationally expensive, especially for large datasets, because it requires calculating distances between the new data point and all existing data points. Additionally, choosing the right value for K and the appropriate distance metric can impact the quality of the predictions. Cross-validation and hyperparameter tuning can help in selecting suitable values for K and the distance metric [50].

1.2.10 Evaluation metrics

Root Mean Squared Error (RMSE)

RMSE is a typical performance measure for regression problems. It gives an idea of how much error the system typically makes in its predictions, with a higher weight for large errors. Equation 1.11 shows the mathematical formula to compute the RMSE.

Equation 1.11. Root Mean Square Error (RMSE)

$$\text{RMSE}(X, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h(x^{(i)}) - y^{(i)}\right)^2} \tag{1.11}$$

Where:

- m is the number of instances in the dataset you are measuring the RMSE on.
- $x^{(i)}$ is a vector of all the feature values (excluding the label) of the i^{th} instance in the dataset, and $y^{(i)}$ is its label (the desired output value for that instance).
- X is a matrix containing all the feature values (excluding labels) of all instances in the dataset. There is one row per instance and the i^{th} row is equal to the transpose of $x^{(i)}$, noted $(x^{(i)})^T$.
- h is your system's prediction function, also called a hypothesis. When your system is given an instance's feature vector $x^{(i)}$, it outputs a predicted value $\hat{y}^{(i)} = h(x^{(i)})$ for that instance ($\hat{y}$ is pronounced "y-hat").
- RMSE(X, h) is the cost function measured on the set of examples using your hypothesis h.

[32]

Mean Absolute Error (MAE)

MAE is a loss function used in regression tasks to measure the average absolute differences between predicted values from a machine learning model and the actual target values. Equation 1.12 shows the mathematical formula to compute the MAE [51].

Equation 1.12. Mean Absolute Error (MAE)

$$\text{MAE}(X, h) = \frac{1}{m}\sum_{i=1}^{m}\left|h(x^{(i)}) - y^{(i)}\right| \tag{1.12}$$

R-squared (R^2)

The R^2 (R-squared), also known as the coefficient of determination, is a common regression metric and statistical measure used to evaluate the performance of regression models. It represents the proportion of the variance in the dependent variable that is predictable from the independent variables in the model. The R^2 score ranges from 0 to 1, with 1 indicating a perfect fit where the model explains all the variability of the target variable, and 0 indicating that the model does not explain any of the variability [52]. R-squared can be calculated using the following formula.

Equation 1.13. R-squared (R^2)

$$R^2 = 1 - \frac{SS_{residual}}{SS_{total}} \tag{1.13}$$

Where:

- The total sum of squares (SS_{total}): Represents the total variation in the dependent variable y from its mean.

$$SS_{total} = \sum_{i=1}^{n} (y_i - \bar{y})^2 \tag{1.14}$$

  where n is the number of observations, y_i is the i^{th} observed value of y and $\overline{y}$ is the mean of y.

- The residual sum of squares ($SS_{residual}$): Represents the unexplained variation in y after fitting the regression model.

$$SS_{residual} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{1.15}$$

  where $\hat{y}_i$ is the predicted value of y_i from the regression model.

1.3 Overview of Optimization

In this section, we explore the concept of hyperparameter tuning (also known as hyperparameter optimization) in machine learning, witch is an essential process to optimize the performance of machine learning models. Proper tuning can well improve the performance of the model, as the performance of a model can be highly dependent on the choice of hyperparameters.

1.3.1 Model hyperparameters

In contrast to model parameters which are learned during training, model hyperparameters are set ahead of training and control implementation aspects of the model.

The weights learned during training of a linear regression model are parameters while the number of trees in a random forest is a model hyperparameter, other examples of hyperparameters include learning rate, number of trees in a random forest, regularization strength. Hyperparameters can be thought of as model settings. These settings need to be tuned for each problem because the best model hyperparameters for one particular dataset will not be the best across all datasets. The process of hyperparameter tuning means finding the combination of hyperparameter values for a machine learning model that performs the best - as measured on a validation (test) dataset - for a problem.

1.3.2 Hyperparameter tuning

It is helpful to think of hyperparameter tuning as having four parts:

1. **Objective function:** A function that takes in hyperparameters and returns a score we are trying to minimize or maximize, in our case we want to maximize the accuracy.

2. **Domain:** The domain, or search space, is all the possible values for all the hyperparameters that we want to search over.For grid search, the domain is a hyperparameter grid and usually takes the form of a dictionary with the keys being the hyperparameters and the values are lists of values for each hyperparameter.

3. **Algorithm**: A method for selecting the next set of hyperparameters to evaluate in the objective function, in our case it is the GridSearch .

4. **Results history**: Data structure containing each set of hyperparameters and the resulting score from the objective function.

Switching from one optimization method to another will only require making minor modifications to these four parts.

Hyperparameter tuning techniques

Manual: select hyperparameters based on intuition/experience/guessing, train the model with the hyperparameters, and score on the validation data. Repeat the process until you are satisfied with the results.

Grid Search: set up a grid of hyperparameter values and for each combination, train a model and score on the validation data. In this approach, every single combination of hyperparameters values is tried.

Random search: set up a grid of hyperparameter values and select random combinations to train the model and score. The number of search iterations is set based on time/resources.

Bayesian optimization: It is an advanced method for hyperparameter tuning in machine learning models. Unlike traditional methods like Grid Search or Random Search. Bayesian optimization builds a probabilistic model to predict the performance of different hyperparameter combinations. This approach intelligently selects the next set of hyperparameters to evaluate based on past results, aiming to find the optimal parameters with fewer iterations and reducing the computational cost associated with exhaustive search methods.

Bayesian methods differ from random or grid search in that they use past evaluation results to choose the next values to evaluate [53].

1.4 Related Work

Li-ion batteries can be considered prime candidates for grid-storage applications, portable electrical devices, electric vehicles, etc. due to their higher cycle life and high-power density compared to other alternatives [54, 55]. A BMS is an integral part of a Li-ion battery pack, designed to monitor and manage the battery's operating conditions. The primary functions of a BMS include SOC estimation and monitoring [56]. Accordingly, numerous studies have explored various techniques and optimization models to improve battery SOC estimation accuracy and efficiency over the last decades [57]. Besides, machine learning (ML) has become indispensable in applied engineering for its ability to analyze vast amounts of data and extract valuable insights for prediction (regression) or classification. For instance, ML helps predict equipment failures, reducing downtime and costs in predictive maintenance [58]. Process optimization also benefits from ML's capacity to identify patterns and trends, leading to increased efficiency and improved results quality [59, 60]. Furthermore, regression using machine learning involves building models that predict continuous outcomes based on input features [61]. These models can be trained on labeled data, where each application has input features and a corresponding continuous target variable [62].

Beyond that, as introduced in the previous section, SOC estimation in battery-powered devices can be categorized into three main types: conventional methods, statistical methods, and data-driven methods. In this section, we provide the recent state-of-the-art applications for each approach along with their drawbacks and advantages. Ng, Kong Soon, et al. [63] proposed an enhanced estimating SOC and SoH of lithium-ion batteries using coulomb counting method, this later is also known as ampere-hour (Ah) counting, the technique involves measuring the current flowing into and out of the battery over time and integrating this current to determine the

total charge that has been added to or removed from the battery [64, 65]. Similarly, Mohammadi, F [66] and Lee J et al [67] proposed an improved Coulomb-Counting algorithm for Li-ion battery SOC estimation and uncertainty evaluation, which verify the high accuracy of SOC estimation compared to other analytical and heuristic approaches. Zheng, et al [68] studied the influence of the open-circuit voltage (OCV) tests on SOC Li-ion battery real-time estimation, which is also a conventional method that demonstrates the effectiveness of the OCV method for SOC estimation where the temperature dependency of the SOC-OCV relationship was examined. Moreover, other conventional methods have been investigated such as Electromechanical impedance spectroscopy [69, 70], this approach enables more accurate battery modeling, leading to improved SOC estimation. Although conventional methods for battery SOC estimation are easy to implement, and provide real-time monitoring in many cases [71], these methods can accumulate errors over time [57], requiring the battery to be at rest for more precise and accurate measurements [72], which cannot not be applicable to all battery types for dynamic real-time applications [73]. Additionally, these conventional methods for SOC estimation may struggle to account for temperature variations that significantly affect battery performance [74]. As a result, these methods often yield inaccurate SOC estimates. To address this, statistical and data-driven approaches can be used to incorporate temperature effects and provide more reliable SOC estimations [75, 76].

Shrivastava, P et [77] proposed an overview of real-time battery SOC estimation employing Kalman Filter (KF) variants, the proposed method incorporate statistical approaches to estimate the SOC by modeling the dynamic behavior of Li-ion battery system with state-space representations, the prediction can be adapted based on real-time measurements, similar approaches have been proposed in [78, 79]. However, Standard KF assumes linearity, which may not be accurate for complex battery systems [80], and might not hold true in all scenarios as it relies on Gaussian noise assumptions [78]. Additionally, Shen, J et al [81] proposed an accurate SOC estimation using Moving Horizon Estimation approach, a statistical optimization framework used to estimate the SOC over a sliding time window, taking into consideration the current and the previous measurements to provide an accurate estimation, conversely, solving an optimization problem at each step over a horizon is computationally complicated [82]. Although many statistical approach-based SOC estimation have been proposed offering powerful SOC estimation [83–85], they come with trade-offs in terms of computational complexity, sensitivity to noise and model assumptions mainly the battery temperature, and the need for careful parameter tuning. Thus, incorporation ML in battery SOC estimation can leverage large datasets, learn complex nonlinear relationships, and adapt to varying operating conditions, making them powerful tools for battery SOC estimation [86, 87].

Li J et al [88] proposed an improved SOC estimation using the Least Squares Support Vector Machine (LSSVM) to establish the battery model, but this approach

requires careful tuning of hyperparameters (e.g., kernel type, regularization); thus, optimization is required to address the issue as presented in our report. Yang, N et [89] suggested a Li-ion battery SoH using Radom Forest (RF) and Convolutional Neural Network (CNN), these approaches may be slow to train on large datasets, as well as a significant amount of memory for storing multiple trees is needed [57]. Talluri T et al [90] studied battery SOC with KNN regression model due to its advantages in terms of simplicity and speed; however, its performance may deteriorate with increasing number of features and samples. Ipek, E et al [91] presented in their paper the SOC estimation of Li-ion battery using SVM and Gradient Boosting (GB) techniques; however, GB is more difficult to tune compared to other models [92]. Also, DL-based methods can provide promising SOC estimation according to some related work [93–96], they also come with significant drawbacks in terms of large data requirement, high computational cost, integration and deployment complexity, maintenance, etc. [97, 98]. As a consequence, addressing these drawbacks is essential for the practical deployment and widespread adoption of SOC estimation in real-world applications. This can be achieved by using machine learning, which forms the right balance between conventional methods and deep learning techniques.

Meng J et al. [99] used Support Vector Machine (SVM) and KF for SOC estimation of Li-ion batteries, but the goal of their work was actually to establish the battery model with limited initial training samples. Lipu M et al. [100] suggested an optimized RF regression algorithm for enhanced SOC estimation accuracy. However, their results were obtained under constant temperature and environmental conditions. Deb S et al. [101] incorporated RF and GB methods for SOC estimation, conducting a comparative study between both approaches, but only in battery discharge mode and without hyperparameter tuning. Additionally, many papers involve either the combination or hybridization of ML-based techniques for Li-ion battery SOC estimation [1, 5, 57, 102–104]; however, based on these cited review papers, the common drawback is that the battery SOC was estimated under constant environmental conditions and in either charging or discharging mode only, as well as hyperparameter tuning is not significantly taken into account. As a result, the present work contributes to SOC estimation for Li-ion batteries by incorporating nine recommended machine learning models: LR, LASSO, KNNR, CatBoostR, ETR, RFR, XGBR, DTR, GBR, in both charging and discharging modes under several battery conditions along with hyperparameter tuning using Gridsearch optimization model for the best three approaches, resulting in more promising outcomes and more accurate evaluation metrics.

Conclusion

In this chapter, we have established the theoretical groundwork for understanding SOC estimation and machine learning. We started by explaining the importance of SOC and reviewed the different approaches used for its estimation including the conventional methods. Then, we provided an overview of machine learning, discussing

various algorithms that can be used for developing models to estimate the SOC, along with the evaluation metrics for assessing these ML models. Lastly, we introduced the concept of optimization and hyperparameter tuning, which are crucial for improving the performance of machine learning models.

This theoretical foundation is essential as we move forward into the next chapters, where we will apply these concepts to develop and optimize machine learning models for accurate real-time SOC estimation.

Chapter 2

Methodology

Introduction

In this chapter, we describe the methodology used for training, evaluating and comparing various regression models to predict the state of charge (SOC) of lithium-ion batteries. Our goal is to identify the most effective model for this prediction task. We employed a systematic approach encompassing several key steps: data preprocessing, model training, evaluation and optimization. This structured methodology ensures that we test and improve our models to achieve the best possible performance.

2.1 Tools

Visual Studio Code (VS Code)

Visual Studio Code (VS Code) is a lightweight and highly popular source code editor developed by Microsoft. It is known for its versatility, extensibility, and support for a wide range of programming languages. VS Code provides a rich set of features such as code completion, debugging, version control integration, and customizable user interface, making it a preferred choice for many developers.

When it comes to Python development, Visual Studio Code provides excellent support. It offers features specifically tailored for Python development, such as syntax highlighting, code formatting, code navigation, linting and debugging capabilities. Overall, VS Code is a popular choice among Python developers due to its efficiency, ease of use and robust Python-specific features.

Python

Python is a computer programming language often used to build websites and software, automate tasks and conduct data analysis. Python is a general-purpose language, meaning it can be used to create a variety of different programs and is not specialized for any specific problems. This versatility along with its beginner-friendliness, has made it one of the most used programming languages today [105]. Stack Overflow's 2022 Developer Survey revealed that Python is the fourth most popular programming language, with respondents saying that they use Python almost 50 percent of the time in their development work. Survey results also showed that Python is tied with Rust as the most-wanted technology, with 18% percent of developers who are not using it already saying that they are interested in learning Python [106].

The Libraries

In the field of coding, a library refers to a collection of pre-compiled code and resources that provide specific functionality, features or tools for developers. Libraries

are designed to be reused, making development more efficient and enabling programmers to leverage existing code rather than building everything from scratch.

There are several libraries used in our program, some of which are part of the Python standard library, and some are not, meaning that they need to be preinstalled.

The libraries and modules used are:

- **Pandas:** The pandas library is a popular data manipulation and analysis tool. It provides data structures like DataFrames for efficient handling of structured data, as well as functions for data cleaning, filtering, merging and more.
- **NumPy:** A fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.
- **Scikit-Learn:** Also known as (SkLearn), is a comprehensive machine learning library in Python. It provides a wide range of algorithms and tools for tasks such as classification, dimensionality reduction and model selection.
- **pyplot:** It is a module within the Matplotlib library that provides a MATLAB-like interface for creating and manipulating static, animated and interactive visualizations in Python. It produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.
- **Tabulate:** A Python library which allows creating formatted tables for display purposes. It supports multiple output formats including plain text, grid, HTML, LaTeX, and more. Also, it is easy to use and integrate into existing codebases, providing functions for printing lists of lists, dictionaries and pandas DataFrames in tabular form.
- **tqdm:** The tqdm library is a fast, extensible progress bar for Python and CLI. It adds progress bars to loops and other iterable operations with minimal code changes. It provides an easy way to visualize the progress of the code execution, making it particularly useful for long-running operations like data processing and training machine learning models. The library supports various output formats, including the console, Jupyter notebooks and graphical interfaces. Its name is derived from the Arabic name taqaddum, which means 'progress'.
- **Warnings:** A standard Python module for issuing and controlling warning messages. It provides functions to issue warnings, filter warnings, and control how warnings are displayed. The library supports different categories of warnings, such as DeprecationWarning and UserWarning, and is used to alert the programmer of conditions that are not necessarily exceptions but might need attention, like deprecated features or potential issues in the code.

All these mentioned libraries and modules either will be necessary for the right functioning of our program, or would help in enhancing its capabilities, where they are imported using the command “import”.

2.2 Data Acquisition & Description

The dataset utilized in this project is the "NASA Battery Dataset" made available by the "NASA Prognostics Center of Excellence". This dataset consists of a set of Li-ion batteries subjected to various operational profiles, including charge, discharge and impedance measurements, conducted at different temperatures. The impedance measurements were performed using electrochemical impedance spectroscopy (EIS) frequency, providing detailed insights into the internal battery parameters as they change with aging.

The dataset captures repeated charge and discharge cycles, which result in accelerated aging of the batteries. The experiments continued until the batteries reached their end-of-life (EOL) criteria, ensuring comprehensive data on battery degradation over time. This dataset is organized into batches of six experiments, with data provided in MATLAB (.mat) files and detailed experiment descriptions in asSOCiated README.txt files.

The rich and detailed nature of this dataset enables it to be used for predicting both the remaining charge for a given discharge cycle and the remaining useful life (RUL) of the batteries. The availability of this dataset on Kaggle facilitates easy access and usability for researchers and developers, making it a valuable resource for advancing battery technology and prognostics.

AI techniques require large datasets for learning, mapping, and predicting. Only appropriate and relevant data produce accurate predictions. So for that, we have made some modifications:

- The flies of the dataset are converted from mat format to CSV format.
- The impedance cycles are removed (we used only charge and discharge cycles).
- "Capacity" column is removed from the discharge cycles.
- "RemT" column is added, which is calculated mathematically based on the "time" column:

$$RemT = Time[max] - Time[current] \tag{2.1}$$

- "SOC" column is also added, which is calculated mathematically:

$$SOC = \frac{RemT[current]}{RemT[max]} \times 100 \tag{2.2}$$

Furthermore, the following features are considered for each cycle:

For charge the features are:

- Voltage_measured: Battery terminal voltage (Volts)
- Current_measured: Battery output current (Amps)
- Temperature_measured: Battery temperature (degree C)
- Current_charge: Current measured at charger (Amps)
- Voltage_charge: Voltage measured at charger (Volts)
- Time: Time vector for the cycle (secs)
- SOC: State of charge (%)
- RemT: Remaining time for charge

For discharge the features are:

- Voltage_measured: Battery terminal voltage (Volts)
- Current_measured: Battery output current (Amps)
- Temperature_measured: Battery temperature (degree C)
- Current_load: Current measured at load (Amps)
- Voltage_load: Voltage measured at load (Volts)
- Time: Time vector for the cycle (seconds)
- SOC: State of charge (%)
- RemT: Remaining time for discharge

Summary:

The final dataset used for this project consists of 5609 CSV files; 2794 represent load cycles (discharging modes), and 2815 represent charge cycles (charging modes).

2.3 Data Visualization and Analysis

This section introduces the key features of the battery dataset and provides an analysis of the observed patterns. The features include voltage measured, current measured, temperature measured, current load/charge and voltage load/charge over time.

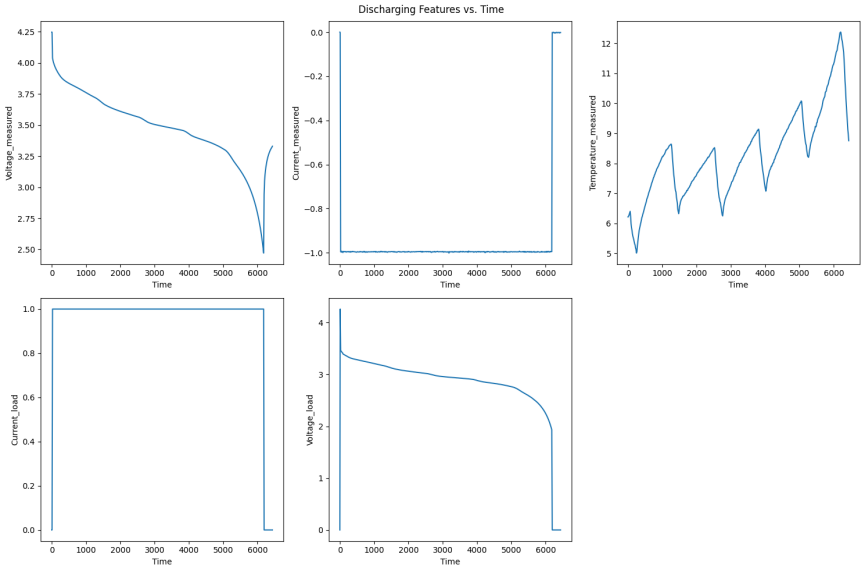


Figure 2.1: Discharging Features vs Time (one cycle)

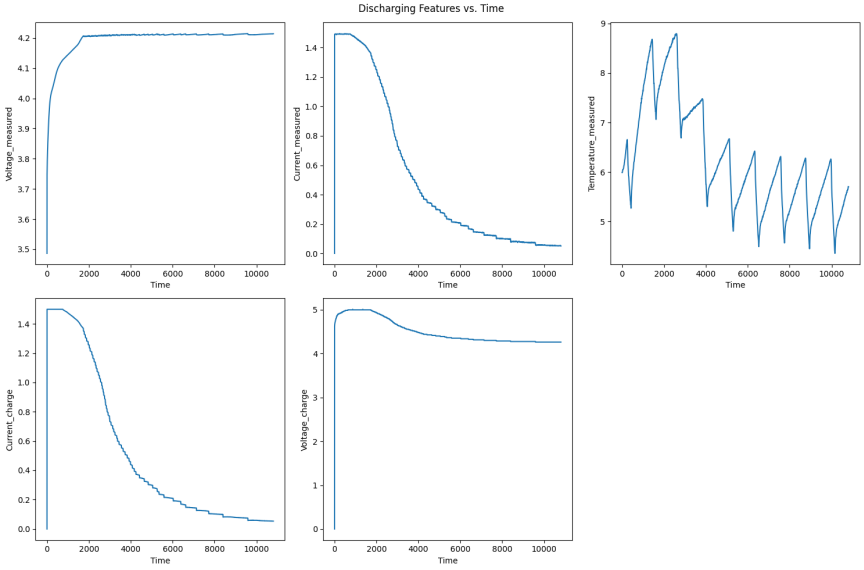


Figure 2.2: Charging Features vs Time (one cycle)

Graph Analysis:
The temperature graphs indicate clearly significant fluctuations during the charging and discharging cycles. These variations in temperature are critical as they impact the remaining charge time and SOC predictions, **highlighting the importance of accounting for thermal effects in battery performance assessments.**

2.4 Data Pre-Processing

2.4.1 Data collection and sampling

For our project, we are working with a dataset of discharge/charge cycles of Li-ion batteries. To ensure that our data is representative of various conditions and changes applied to the discharge/charge batteries, we have implemented a manual sampling strategy. Specifically, we sampled the discharge CSV(s) files by selecting **one file out of every six files** and for charge CSV(s) we selected **one file out of every twelve files**. This approach helps us cover the full spectrum of different samples and conditions present in the dataset, providing a comprehensive view of the battery discharge/charge behavior.

This sampling method is particularly useful in scenarios where the data is sequential or where changes in the conditions over time need to be accounted for. By systematically sampling every sixth/twelve file, we reduce the volume of data to a manageable size while still capturing the variability in the discharge/charge cycles. This careful selection process ensures that our analysis and any modeling efforts will be based on a well-rounded and diverse dataset.

Next, we will delve into the specific steps of our data pre-processing workflow, including concatenation CSV(s), handling missing values and preparing it for analysis and modeling.

2.4.2 Data concatenation

After loading the sampled files, we concatenated them into a single DataFrame called **discharge_df/charge_df**. This involved stacking the rows of each sampled file one after the other. This DataFrame encompasses the entirety of the discharge/charge data, resulting in a dataset with **128,634 rows (for discharge)/544,808 rows(for charge), and 8 columns.** This step is critical for efficient data analysis and model training, as it centralizes all relevant information in one place, making it easier to manipulate and analyze.

2.4.3 Data cleaning

Any dataset consists of duplicates, imbalanced and corrupt samples that can lead to inconsistency during training and prediction, so such samples are excluded. Hence, our dataset was processed and cleaned so that it would be ready to feed for the training of an ML model. It was made sure that there were no null values in any row (masking) and the precision of the data points was uniform throughout. This step was achieved using the function "dropna" from python's data analysis library; Pandas. Data cleaning was an important measure because null values and non-uniform data points adversely affect the accuracy of any ML algorithm [107].

2.4.4 Data splitting

Traditionally, to employ machine learning algorithms, the datasets are randomly split into two parts: training and test sets. The data is assigned randomly to any of the two sets, but according to a particular ratio which The rough standard for train-validation-test splits is (60-80):(40-20) [108] where the training set contains 60-80% of the data and the test set contains 40-20% of the data. The purpose of this split is to evaluate the ML model and check its accuracy against the test set. The model is trained only on training data and then its performance is tested on the test set to gain insight into the validity of the model. For this purpose, we split the dataset into randomized 70:30 training and test sets using python.

The feature "SOC" is then chosen to be the target (output). Because of their linear relationship with "SOC," the "RemT" and "Time" features are removed from the dataset, leaving "SOC" to represent the three features.

With the following lines of code, we perform the train-test split on our dataset using the train_test_split function from the module sklearn.model_selection and choosed the target:

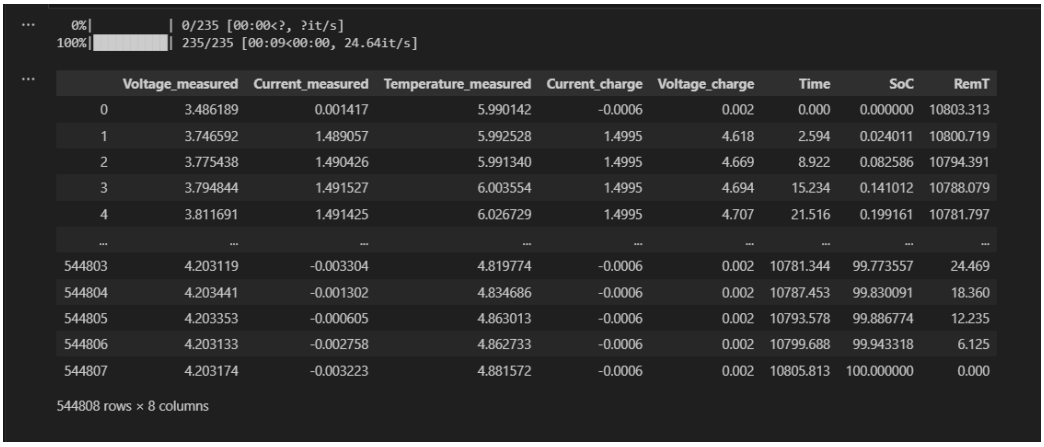
```
from sklearn.model_selection import train_test_split


y = discharge_df['SoC']
X = discharge_df.drop([ 'Time','SoC' ,'RemT'], axis=1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)


print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

Figure 2.3: Data splitting

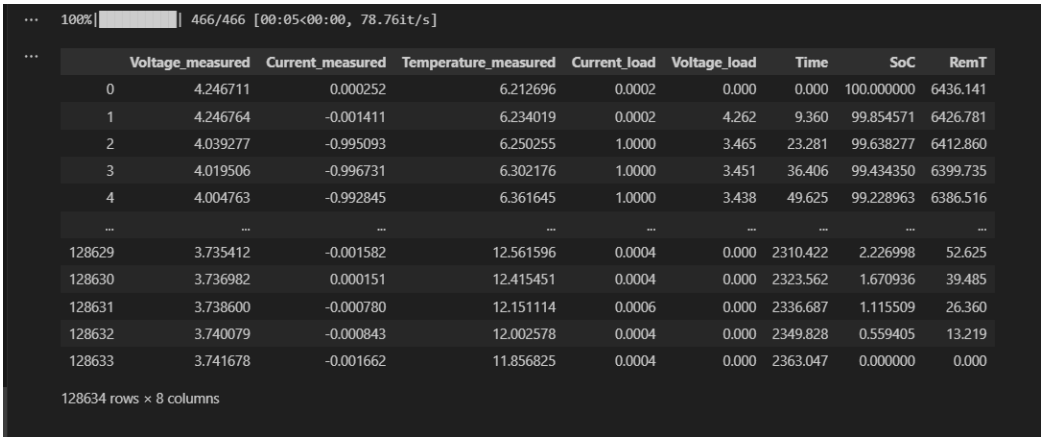


```
  0%|          | 0/235 [00:00<?, ?it/s]
100%|██████████| 235/235 [00:09<00:00, 24.64it/s]
```

	Voltage_measured	Current_measured	Temperature_measured	Current_charge	Voltage_charge	Time	SoC	RemT
0	3.486189	0.001417	5.990142	-0.0006	0.002	0.000	0.000000	10803.313
1	3.746592	1.489057	5.992528	1.4995	4.618	2.594	0.024011	10800.719
2	3.775438	1.490426	5.991340	1.4995	4.669	8.922	0.082586	10794.391
3	3.794844	1.491527	6.003554	1.4995	4.694	15.234	0.141012	10788.079
4	3.811691	1.491425	6.026729	1.4995	4.707	21.516	0.199161	10781.797
...	...	...	...	...	...	...	...	...
544803	4.203119	-0.003304	4.819774	-0.0006	0.002	10781.344	99.773557	24.469
544804	4.203441	-0.001302	4.834686	-0.0006	0.002	10787.453	99.830091	18.360
544805	4.203353	-0.000605	4.863013	-0.0006	0.002	10793.578	99.886774	12.235
544806	4.203133	-0.002758	4.862733	-0.0006	0.002	10799.688	99.943318	6.125
544807	4.203174	-0.003223	4.881572	-0.0006	0.002	10805.813	100.000000	0.000

544808 rows × 8 columns

Figure 2.4: Pre-Processed Charge dataset



```
100%|██████████| 466/466 [00:05<00:00, 78.76it/s]
```

	Voltage_measured	Current_measured	Temperature_measured	Current_load	Voltage_load	Time	SoC	RemT
0	4.246711	0.000252	6.212696	0.0002	0.000	0.000	100.000000	6436.141
1	4.246764	-0.001411	6.234019	0.0002	4.262	9.360	99.854571	6426.781
2	4.039277	-0.995093	6.250255	1.0000	3.465	23.281	99.638277	6412.860
3	4.019506	-0.996731	6.302176	1.0000	3.451	36.406	99.434350	6399.735
4	4.004763	-0.992845	6.361645	1.0000	3.438	49.625	99.228963	6386.516
...	...	...	...	...	...	...	...	...
128629	3.735412	-0.001582	12.561596	0.0004	0.000	2310.422	2.226998	52.625
128630	3.736982	0.000151	12.415451	0.0004	0.000	2323.562	1.670936	39.485
128631	3.738600	-0.000780	12.151114	0.0006	0.000	2336.687	1.115509	26.360
128632	3.740079	-0.000843	12.002578	0.0004	0.000	2349.828	0.559405	13.219
128633	3.741678	-0.001662	11.856825	0.0004	0.000	2363.047	0.000000	0.000

128634 rows × 8 columns

Figure 2.5: Pre-Processed Discharge dataset

2.5 Applying Machine Learning Models

The SOC prediction models employs machine learning algorithms. A total of 9 different algorithms were tested to find the model that works best. As this was a regression problem, some traditional algorithms were tried out first, then the analysis moved to much more powerful Regressors. Finally, a comparative analysis of these algorithms is performed to select the best model.

2.5.1 Model building

We selected a diverse set of regression models (9 Models) to compare their performance. The chosen models include:

a. LinearRegression

b. KNeighborsRegressor

c. Lasso

d. RandomForestRegressor

e. DecisionTreeRegressor

f. ExtraTreeRegressor

g. XGBRegressor

h. GradientBoostingRegressor

i. CatBoostRegressor

Evaluation Metrics

To evaluate the performance of the various regression models, we employed three commonly used metrics: R-squared (R^2), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).

- **R-squared (R^2):** This metric indicates how well the independent variables explain the variability of the dependent variable. An R^2 value closer to 1 suggests a better fit. **Mean Absolute Error (MAE):** MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It provides a straightforward measure of prediction accuracy.

- **Root Mean Squared Error (RMSE):** RMSE is the square root of the average of squared differences between prediction and actual observation. It penalizes larger errors more than MAE, providing a clear view of the prediction error magnitude.

2.5.2 Training and evaluation process

The process for training and evaluating each model involves the following steps:

1. **Model Training:** A function **evaluate_and_print** is defined to train a given list of models, make predictions, evaluate the model's performance, and visualize the results. The function takes four parameters: **model**, **model_name**, **X_train_param** and **y_train_param**
   The **model** parameter is the machine learning model to be trained and evaluated. **X_train_param** and **y_train_param** are the training data and corresponding labels respectively. **model_name** is a string representing the name of the model.

2. **Prediction:** Inside the **evaluate_and_print** function, the **model** is first trained using the **fit** method on the training data. Then, predictions are made on the test data using the **predict** method.

3. **Evaluation:** The predictions are evaluated using the metrics described above.

4. **Visualization:** A subset of the predictions (first 5% of the test data) is visualized (predicted values against the actual values) to inspect model performance visually.

Results

The tables 2.1 and 2.2 summarizes the Results.

model/evaluation	R2	MAE	RMSE
Lasso	0.717477	11.811579	15.527722
KNeighborsRegressor	0.911580	3.733305	8.686714
LinearRegression	0.717931	11.800901	15.515255
CatBoostRegressor	0.929583	4.443967	7.752089
ExtraTreesRegressor	**0.975732**	1.545410	4.550950
RandomForestRegressor	**0.973361**	1.712858	4.768001
XGBRegressor	0.932529	4.296816	7.588190
DecisionTreeRegressor	**0.947810**	1.744184	6.673812
GradientBoostingRegressor	0.885620	5.892310	9.880000

Table 2.1: Results of the Charge Cycles

model/evaluation	R2	MAE	RMSE
Lasso	0.440520	17.516225	21.634235
KNeighborsRegressor	0.918980	4.695400	8.232780
LinearRegression	0.440387	17.442982	21.636800
CatBoostRegressor	**0.964902**	3.438097	5.418612
ExtraTreesRegressor	**0.981254**	1.998865	3.960077
RandomForestRegressor	**0.978734**	2.192992	4.217876
XGBRegressor	**0.966208**	3.344424	5.316899
DecisionTreeRegressor	**0.961061**	2.598255	5.707416
GradientBoostingRegressor	0.904167	6.565596	8.953772

Table 2.2: Results of the Discharge Cycles

The results of these optimizations will be discussed in detail in chapter 3 (Results and Discussion)

2.6 Model Optimization

Hyperparameter tuning is a crucial step in developing machine learning models, as the default parameters of a model might not always give the best performance. This section is about optimizing these parameters to improve the model's performance.

In our project, GridSearchCV is the method used for optimization, it searches over specified, predefined range values. It trains the model multiple times on that range of parameters that we specified and gives out the best parameter combination, which can then be used for training the final model.

We applied the GridSearchCV method to various models and obtained different results. For some models, we successfully identified the optimal parameters, while for others, we were unable to derive optimal parameters from the specified range of values.

2.7 Pseudocode

1: **Import necessary libraries numpy, sklearn, matplotlib ..**

2: **Load/sample the dataset**

dataset ← `load dataset containing Charge/Discharge data`

3: **Split the dataset into Charge and Discharge datasets**

chargeDataset ← `split dataset to get Charge data`

dischargeDataset ← `split dataset to get Discharge data`

4: **Handle missing values**

`handle missing values in chargeDataset`

5: **Split the data into training and testing sets**

`split data where SOC is the target feature`

6: **Define machine learning models for prediction**

models ← [`list of machine learning models`]

7: **Initialize an empty dictionary for results**

results ← {}

8: **function** EVALUATE_AND_PRINT(model, X_train, Y_train, model_name)

model.fit(X_train, y_train)

y_pred ← $model.predict(X\_test)$

Calculate Evaluation metrics (MSE , RMSE , MAE, R2)

Store error metrics in results dictionary

results[model_name] ← {`MSE , RMSE , MAE, R2`}

Create a scatter/line plot

`plot(Y_test ,Y_pred )`

9: **end function**

10: **for all** model **in** models **do**

    EVALUATE_AND_PRINT(model, X_train, Y_train, model_name)

11: **end for**

12: **Print results of each model's prediction**

`print(results)`

13: **Plot actual vs predicted SOC for each model**

For All model_name in results

    `plot(actual vs predicted SOC)`

End For

14: **Save the best performing model for future use**

best_model ← `get model with best performance from results`

15: **Import GridSearchCV for optimization**

16: **for all** model **in** models **do**

    param_grid ← `define parameter grid`

    gridSearch ← `GridSearchCV(model, param_grid, 'r2', cv strategy)`

    gridSearch.fit(X_{train}, y_{train})

    best_params ← `gridSearch.best_params_`

    best_estimator ← `gridSearch.best_estimator_`

17: **end for**

2.8 Model Deployment

in this section, we detail the hardware implementation of our project, which focuses on the real-time monitoring of the state of charge (SOC) of lithium-ion batteries during the discharging cycles. This involves selecting the best-performing model, deploying it on ESP32, and visualizing the results on a web page.

2.8.1 Selecting the best performing model

Following a comprehensive evaluation of various regression models, the Extra Tree Regressor was selected due to its superior performance in predicting the SOC for discharge cycles, making it the ideal choice for real-time monitoring applications.

2.8.2 Saving the trained model

To enable real-time predictions, the trained Extra Tree Regressor model was saved in the (.sav) format using the Pickle library. Pickle allows us to serialize the model, preserving its state and weights. This ensures the model can be easily loaded and used for predictions without the need for retraining.

2.8.3 Creating the server using Flask

A server was developed using Flask, a lightweight web framework for Python. This server receives data related to the battery's current, voltage, and temperature captured from various sensors to the esp32 (see circuit in fig 2.7) which was programmed using Thony IDE, and then fits this data to the pre-trained model (in .sav format) to predict the corresponding SOC.

2.8.4 Displaying results on a web page

The results of the SOC estimation are displayed on a web page. The web page compares the predicted SOC value with the actual SOC value, providing a clear and user-friendly visualization of the model's performance in both discharging modes. Fig 2.6 shows the monitoring of the results of state of charge during discharge mode in a web page.

2.8.5 Conclusion

In conclusion, this hardware implementation showcases the practical application of our research findings. By selecting the best-performing model and deploying it onto an ESP32 microcontroller for real-time monitoring. This system provides a reliable and user-friendly method for real-time battery monitoring.

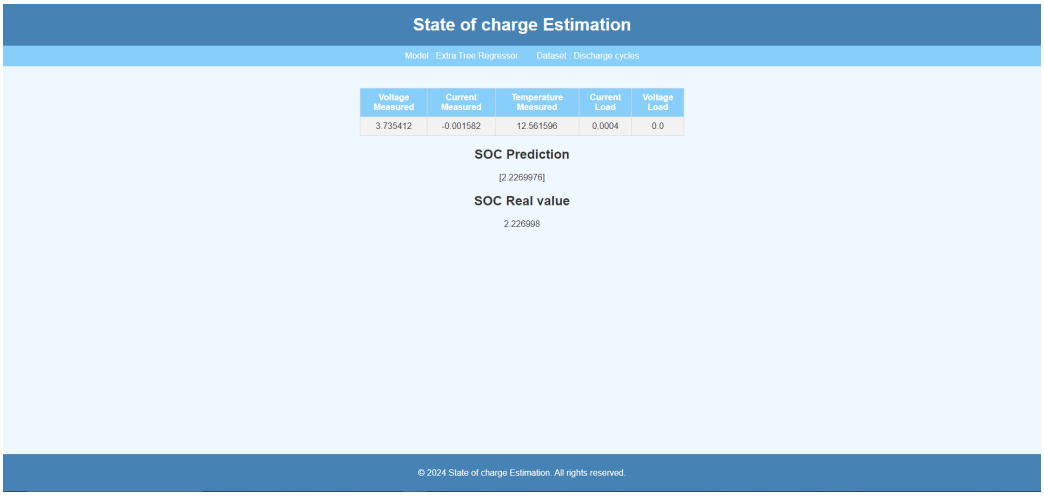


Figure 2.6: Discharging mode results presented on a web page.

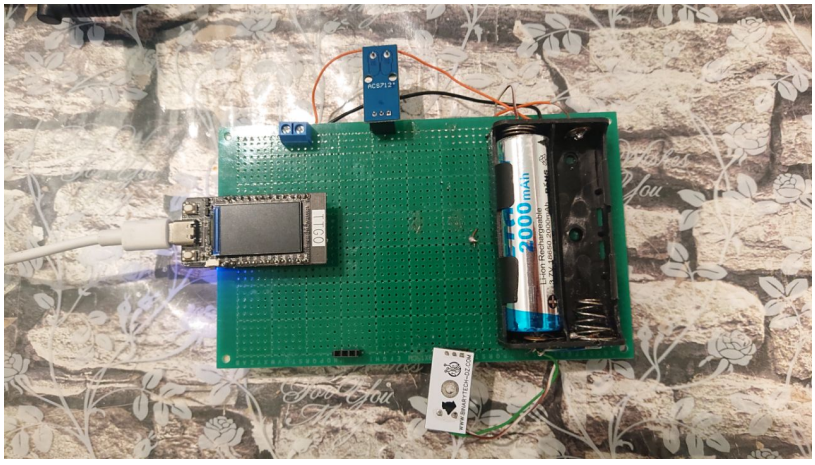


Figure 2.7: Battery data collecting circuit.

Conclusion

In this chapter, we have outlined the overall methodology employed to train, evaluate, and compare various regression models for predicting the state of charge (SOC) of lithium-ion batteries. By following a systematic approach including data preprocessing, model training, evaluation, and hyperparameter optimization, we aimed to identify the most effective model for our prediction task. The utilization of GridSearchCV for hyperparameter tuning enabled us to develop our models, although it yielded mixed results across different models. The findings from these methodological steps lay the groundwork for the detailed analysis and discussions presented in the following chapter.

Chapter 3

Results and Discussion

Introduction

In this chapter, we will present and discuss the obtained results, as well as a detailed analysis of the performance of various machine learning models in predicting the state of charge (SOC) of lithium-ion batteries. Our focus is on comparing machine learning models' performance, and exploring the impact of sampling, optimization and including temperature as a feature.

3.1 Presentation of Results

We start by showcasing the performance metrics of each model. These metrics include the coefficient of determination (R^2), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE). These metrics provide a comprehensive understanding of how well each model captures the underlying patterns in the data and predicts the target variable.

3.1.1 Results of the disharge cycles

model/evaluation	R^2	MAE	RMSE
Lasso	0.440520	17.516225	21.634235
KNeighborsRegressor	0.918980	4.695400	8.232780
LinearRegression	0.440387	17.442982	21.636800
CatBoostRegressor	**0.964902**	3.438097	5.418612
ExtraTreesRegressor	**0.981254**	1.998865	3.960077
RandomForestRegressor	**0.978734**	2.192992	4.217876
XGBRegressor	**0.966208**	3.344424	5.316899
DecisionTreeRegressor	**0.961061**	2.598255	5.707416
GradientBoostingRegressor	0.904167	6.565596	8.953772

Table 3.1: Evaluation metrics of Discharge Cycles

Analysis:

The **Extra Trees Regressor** is the most effective model for the **Discharge dataset**, followed closely by the **Random Forest Regressor** and **XGBRegressor**. These models show the highest R^2 values and the lowest error metrics, making them the best choices for accurate predictions. Traditional models like Lasso Regression and Linear Regression perform poorly in comparison, indicating that more complex ensemble methods are better suited for this problem, and that the output "SoC" not have a liner relationship with the inputs (voltage, current, temperature).

3.1.2 Results of the charge cycles

model/evaluation	R^2	MAE	RMSE
Lasso	0.717477	11.811579	15.527722
KNeighborsRegressor	0.911580	3.733305	8.686714
LinearRegression	0.717931	11.800901	15.515255
CatBoostRegressor	0.929583	4.443967	7.752089
ExtraTreesRegressor	**0.975732**	1.545410	4.550950
RandomForestRegressor	**0.973361**	1.712858	4.768001
XGBRegressor	0.932529	4.296816	7.588190
DecisionTreeRegressor	**0.947810**	1.744184	6.673812
GradientBoostingRegressor	0.885620	5.892310	9.880000

Table 3.2: Evaluation metrics of Charge Cycles

Analysis:

The **ExtraTreesRegressor** and **RandomForestRegressor** demonstrated the best overall performance across all metrics, making them the most suitable models for the **Charge dataset**. they provide the most accurate and reliable predictions, followed closely by the **DecisionTreeRegressor**, and as said for discharge dataset liner regression an lasso performed poorly because there was no liner relationship between the input and the output. For future development and deployment, the recommended models are ExtraTrees, RandomForest and DecisionTree Regressors.

3.2 Graphical Representation

To present a clearer view of the models' performances, visual representations of the results are included. Graphs comparing predicted values with actual values help illustrate the accuracy and precision of each model. These visuals are crucial for detecting patterns or systematic errors in the predictions.

Discharge Cycles

Below are scatter plots of the discharge dataset, showing predicted values (blue points) versus actual values (red line) of 1929 samples out of a total of 38591 samples. Please note that only 5% of the values are displayed to enhance the graph's readability without compromising the information provided.

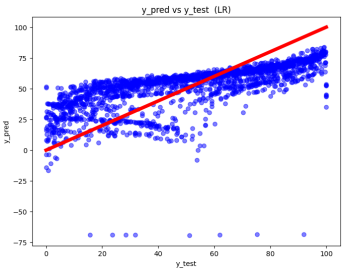


Figure 3.1: LR

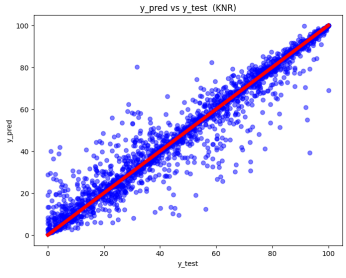


Figure 3.2: KNR

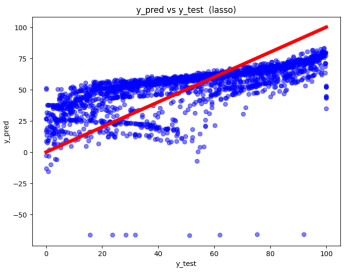


Figure 3.3: lasso

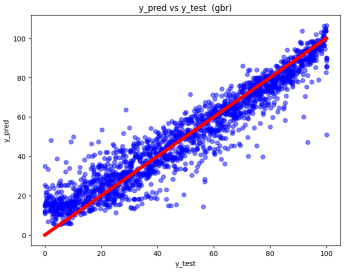


Figure 3.4: gbr

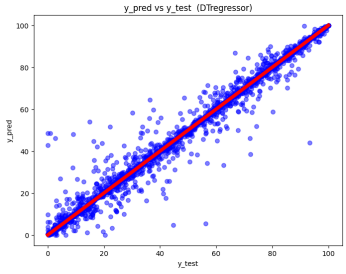


Figure 3.5: DTregressor

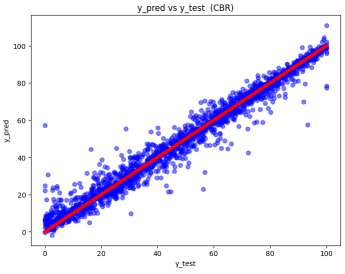


Figure 3.6: CBR

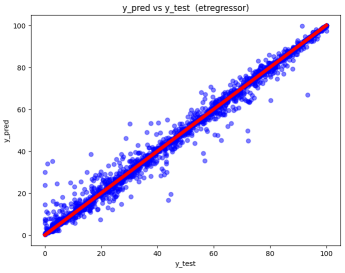


Figure 3.7: etregressor

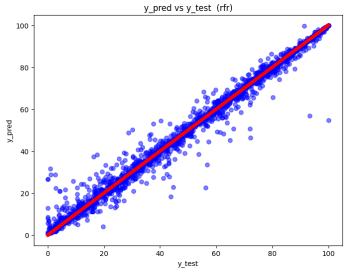


Figure 3.8: rfr

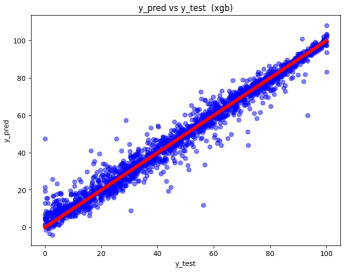


Figure 3.9: xgboost

Analyze scatter plots of discharge cycles:
The information deduced form the graphs are compatible from what is mentioned in the table. This consistency indicates that the models with high error rates consistently deviate significantly from the actual values, while those with lower error rates demonstrate predictions that closely align with the actual data.

Charge Cycles

Below are scatter plots of the Charge dataset, showing predicted values (blue points) versus actual values (red line) of 8172 samples out of a total of 163443 samples. Please note that only 5% of the values are displayed to enhance the graph's readability without compromising the information provided.

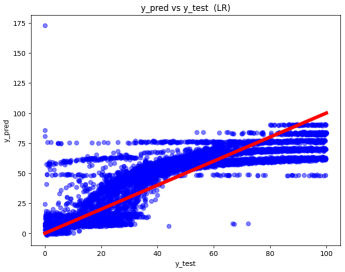


Figure 3.10: LR

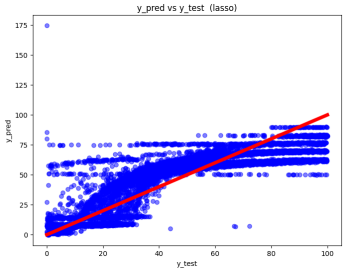


Figure 3.11: lasso

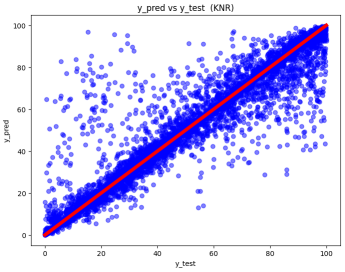


Figure 3.12: KNR

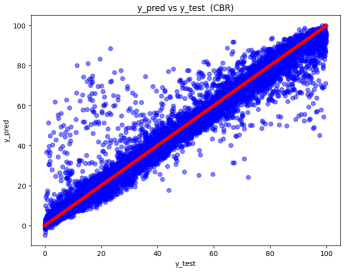


Figure 3.13: CBR

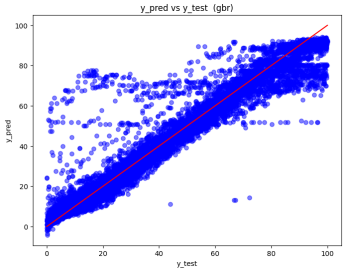


Figure 3.14: gbr

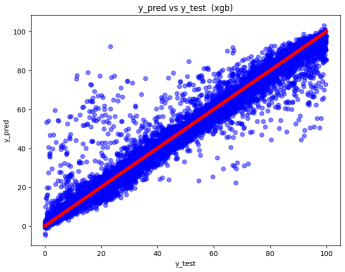


Figure 3.15: etregressor

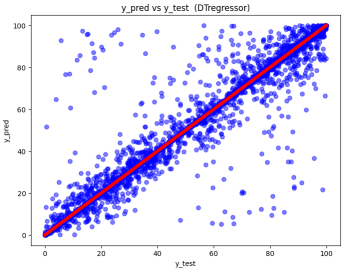


Figure 3.16: DTregressor

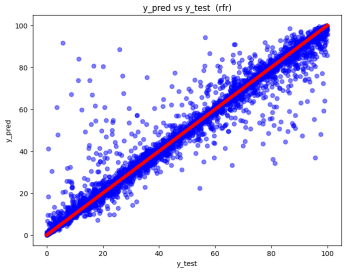


Figure 3.17: rfr

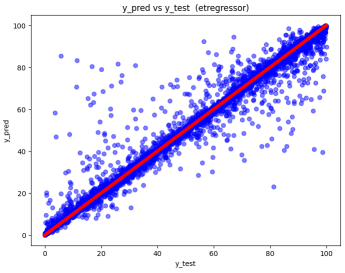


Figure 3.18: xgboost

Analyze scatter plots of charge cycles:
As mentioned in the Analyze of discharge cycles the information deduced form the graphs are compatible with what is mentioned in the table ,where this information indicates that the models with high error rates consistently deviate significantly from the actual values, while those with lower error rates demonstrate predictions that closely align with the actual data, although is not as precise and accurate as the discharge results .

3.3 Optimization

Hyperparameter tuning is a crucial step in building machine learning models, as the default parameters of a model might not always give the best performance. This section is about optimizing these parameters to improve the model's performance.

In this case, GridSearchCV is used to find the optimal parameters for the our models. This include parameters like the learning rate, maximum depth of the trees, number of trees and so on.

The models are fine-tuned through grid search, resulting in various improvements. Below are the models that were enhanced after optimization, along with their optimal parameters.

model/evaluation	R^2	MAE	RMSE
xgb_opt	**0.9713**	2.416603	4.649071
DTregressor_opt	**0.96405**	2.686923	5.146051
gbr_opt	**0.9734**	2.394721	4.433793

Table 3.3: optimization results of Discharge cycles

```
xgb :
Best parameters {'learning_rate': 0.1, 'max_depth': 11, 'n_estimators': 500}
Mean cross-validated accuracy score of the best_estimator: 0.95538

DTregressor :
Best parameters {'max_depth': 19, 'min_samples_leaf': 5, 'min_samples_split': 3}
Mean cross-validated accuracy score of the best_estimator: 0.95119

gbr :
Best parameters {'learning_rate': 0.1, 'max_depth': 13, 'n_estimators': 100}
Mean cross-validated accuracy score of the best_estimator: 0.96326
```

Figure 3.19: optimal parameters of the enhanced models (discharge cycles)

visual representations of discharge cycles after optimization:

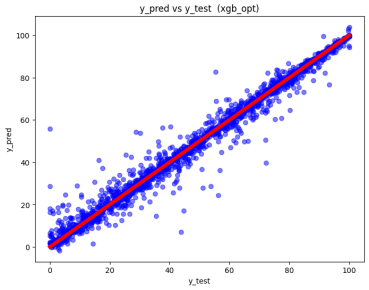


Figure 3.20: gbr

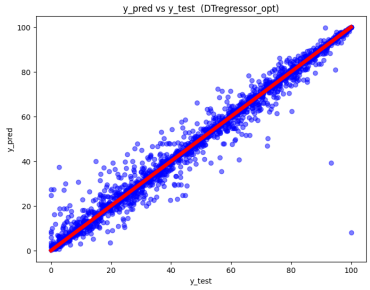


Figure 3.21: DTregressor

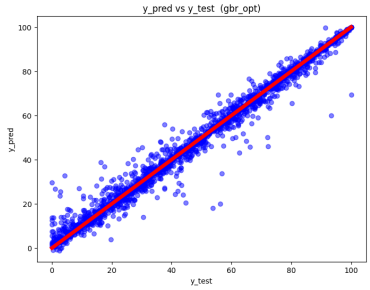


Figure 3.22: xgboost

model/evaluation	R^2	MAE	RMSE
xgb_opt	0.95538	2.434957	6.049739
DTregressor_opt	0.9511	2.328652	6.132173
gbr_opt	0.966635	2.587810	5.336135

Table 3.4: optimization results of Charge cycles

```
xgb :
Best parameters {'learning_rate': 0.1, 'max_depth': 11, 'n_estimators': 500}
Mean cross-validated accuracy score of the best_estimator: 0.97132

DTregressor :
Best parameters {'max_depth': 19, 'min_samples_leaf': 6, 'min_samples_split': 3}
Mean cross-validated accuracy score of the best_estimator: 0.96405

gbr :
Best parameters {'learning_rate': 0.1, 'max_depth': 13, 'n_estimators': 100}
Mean cross-validated accuracy score of the best_estimator: 0.97347
```

Figure 3.23: optimal parameters of the enhanced models (charge cycles)

visual representations of charge cycles after optimization:

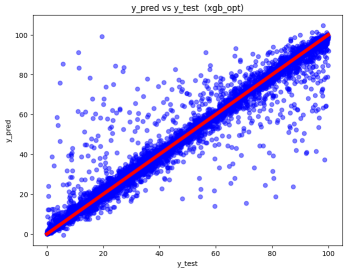


Figure 3.24: xgb

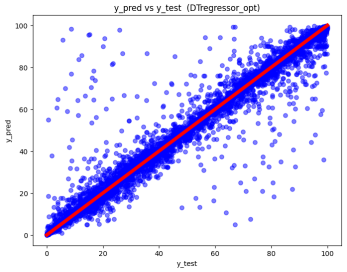


Figure 3.25: DTregressor

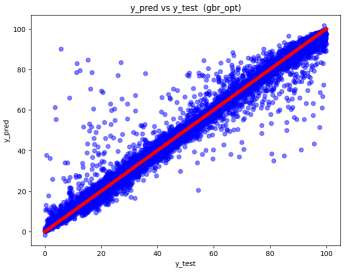


Figure 3.26: gbr

Assessing the Impact of Temperature Parameter on Model Performance

	R2	**MAE**	**RMSE**
Lasso	0.697837	12.260779	16.058376
KNeighborsRegressor	0.899474	5.270680	9.262309
LinearRegression	0.698184	12.243111	16.049155
CatBoostRegressor	0.888151	6.180162	9.770035
ExtraTreesRegressor	0.897591	5.200063	9.348677
RandomForestRegressor	0.902601	5.209244	9.117138
XGBRegressor	0.889185	6.126000	9.724773
DecisionTreeRegressor	0.829624	6.199070	12.058288
GradientBoostingRegressor	0.863098	6.959878	10.809012

Table 3.5: Discharge cycles (without Temperature parameter)

	R2	**MAE**	**RMSE**
Lasso	0.398428	18.123801	22.433294
KNeighborsRegressor	0.836216	7.394032	11.705353
LinearRegression	0.398318	18.056626	22.435349
CatBoostRegressor	0.864462	7.146515	10.648281
ExtraTreesRegressor	0.879684	6.028250	10.032559
RandomForestRegressor	0.878334	6.149669	10.088648
XGBRegressor	0.867518	6.977616	10.527543
DecisionTreeRegressor	0.785730	7.379432	13.388448
GradientBoostingRegressor	0.787143	10.148881	13.344207

Table 3.6: Charge cycles (without Temperature parameter)

Analysis of Tables Demonstrating the Impact of Temperature Parameter

It is clear that the temperature parameter has a significant impact on model performance. When the temperature parameter is removed, the accuracy of discharge cycles decreases by more than 7%, and for charge cycles, it decreases by more than 10%.

3.4 Discussion of Results

In this section, we dig deeper into the findings. We interpret the results, explore the reasons behind the models' performances, and discuss any unexpected outcomes. This discussion provides a critical analysis of the results.

Evaluation Metrics Relationship

this part shows the relationship between the evaluation metrics R^2, RMSE, and MAE and whether they represent the same aspects of model performance ,and does the models' performance consistently represented across different metrics.

While R^2 measures how well the model explains the variance in the data, RMSE and MAE assess the actual prediction errors. A high R^2 does not necessarily imply low RMSE or MAE. However, our models **consistently achieved high R^2 values along with low RMSE and MAE**, indicating their reliability and demonstrating that these evaluation metrics represent consistent aspects of model performance.

Superior Performance of Tree-Based and Boost-Based Models

Our results clearly indicate that tree-based and boost-based models outperform linear models in predicting the SOC. This can be attributed to the capabilities of these models to handle complex, non-linear relationships and interactions between features, which are common in our dataset.

- Tree-Based Models: Models such as ExtraTreesRegressor, RandomForestRegressor, and DecisionTreeRegressor excel in capturing the intricate patterns within the data. These models can split the data into homogenous subgroups, making them particularly effective for our application where the relationship between the input variables and the SOC is non-linear and multifaceted.
- Boost-Based Models: CatBoostRegressor, XGBRegressor, and GradientBoostingRegressor showed remarkable performance due to their ability to sequentially correct errors of the previous models. The iterative boosting process enhances their accuracy, making them robust against overfitting and capable of providing superior predictions.
- In contrast, linear models like Lasso and LinearRegression assume a linear relationship between the dependent and independent variables. Given the complexity of our dataset, these models failed to capture the intricate dependencies, resulting in lower predictive accuracy.

Superior Results for Discharge Cycles

For discharge cycles, six models achieved an accuracy range of 96% to 98%, whereas for charge cycles, five models attained an accuracy range of 96% to 97%. Notably, despite the smaller dataset size used for discharge cycles compared to charge cycles, the models used on Discharge Cycles demonstrated superior performance.

Comprehensive Dataset Coverage

This section highlights the work done on the dataset and its relationship with the results. The dataset was sampled by selecting CSV files at regular intervals to cover a wide range of details, ensuring that our models are trained and tested on diverse data points. **This approach guarantees that our results are reliable and representative**, and It does not express only a small part of the dataset.

Consideration of Temperature Parameter

It is worth noting that a key aspect of our study is the inclusion of temperature as a feature in predicting the State of Charge (SOC). **By incorporating temperature as an input (feature), we have achieved accurate SOC estimation results, ensuring accounting for its effect**, while Traditional methods often overlook the temperature parameter, focusing only on current and voltage.

Optimization using GridSearch

To further improve our models performance, we employed GridSearch for hyperparameter optimization. This process involved searching through a predefined grid of hyperparameters to identify the optimal settings for each model. Notably, the Gradient Boosting Regressor and Decision Tree Regressor showed significant improvement after optimization. **For instance, the Gradient Boosting Regressor achieved an R^2 increase of 7.8% and a reduction in RMSE and MAE by 4.5 and 2.8, respectively.** Similarly, the Decision Tree Regressor exhibited enhanced accuracy. These improvements demonstrate the importance of hyperparameter tuning in maximizing the accuracy of machine learning models.

Conclusion

In summary, our analysis shows that advanced machine learning models, particularly tree-based and boost-based approaches, significantly outperform linear models in predicting the SOC of lithium-ion batteries. The models demonstrated superior performance in discharge cycles, achieving high accuracy despite the dataset's complexity. The inclusion of temperature as a feature proved crucial in capturing its effect on SOC. Our data sampling method ensured comprehensive coverage, leading to reliable and generalizable results. The evaluation metrics consistently reflected model performance and hyperparameter tuning was essential in optimizing model effectiveness. These findings highlight the potential of machine learning techniques in enhancing dealing with one of the important parameters of Li-ion batteries (state of charge).

Conclusion

This thesis presents a novel solution for predicting the state of charge (SOC) of lithium-ion batteries, which is a crucial parameter for their efficient performance, extending life time and safe operation. Utilizing a data-driven approach (learning algorithms), we addressed the limitations of conventional methods, which often fail to account for effective variables like temperature, where this parameter (ie: temperature) is considered by including it as a feature of the used ML models and proved to have a significant impact on the model's performance.

Nine different regression models were initially evaluated, among of them were tree-based methods including Decision Tree, Random Forest, and ExtraTree Regressors and boost-based such as XGB, Gradient Boosting and Cat Boost Regressors. Also, linear-based models like Linear Regressor and Lasso were considered. Notably, the advanced machine learning models (tree-based and boost-based) demonstrated superior performance compared to linear models. Furthermore, the models were optimized using Hyperparameter tuning, particularly through GridSearchCV method ,yielding a significantly performance improvement especially for XGB and Gradient Boosting Regressors for both discharge/charge cycles, then XGB model was selected to be deployed onto an ESP32 microcontroller for real-time monitoring, showcasing practical application and implementation of the research findings.

These findings illustrate the superiority of advanced machine learning models (tree-based and boost-based) in handling the complexities of battery SoC prediction, especially under varying operational conditions. Through this research, we have demonstrated that data-driven approaches not only provide high accuracy but also offer a robust and reliable solution for SOC estimation in lithium-ion batteries.

Bibliography

[1] M. Kurucan, M. Özbaltan, Z. Yetgin, and A. Alkaya, "Applications of artificial neural network-based battery management systems: a literature review," *Renewable and Sustainable Energy Reviews*, vol. 192, p. 114262, 2024.

[2] T. A. Fagundes, G. H. F. Fuzato, R. F. Q. Magossi, L. J. R. Silva, J. C. Vasquez, J. M. Guerrero, and R. Q. Machado, "A modified redundancy-based energy management system for microgrids: An soc enhancement approach," *IEEE Transactions on Industrial Electronics*, 2024.

[3] D.-J. Lim, J.-H. Ahn, D.-H. Kim, and B. K. Lee, "A mixed soc estimation algorithm with high accuracy in various driving patterns of evs," *Journal of Power Electronics*, vol. 16, no. 1, pp. 27–37, 2016.

[4] A. Adel, O. Hand, G. Fawzi, T. Walid, R. Chemseddine, and B. Djamel, "Gear fault detection, identification and classification using mlp neural network," in *Recent Advances in Structural Health Monitoring and Engineering Structures: Select Proceedings of SHM and ES 2022*. Springer Nature Singapore, 2022, pp. 221–234.

[5] Y. Liu, Y. He, H. Bian, W. Guo, and X. Zhang, "A review of lithium-ion battery state of charge estimation based on deep learning: Directions for improvement and future trends," *Journal of Energy Storage*, vol. 52, p. 104664, 2022.

[6] M. H. Lipu, S. Ansari, M. S. Miah, S. T. Meraj, K. Hasan, A. S. M. Shihavuddin, ..., and A. Hussain, "Deep learning enabled state of charge, state of health and remaining useful life estimation for smart battery management system: Methods, implementations, issues and prospects," *Journal of Energy Storage*, vol. 55, p. 105752, 2022.

[7] A. Afia, F. Gougam, W. Touzout, C. Rahmoune, H. Ouelmokhtar, and D. Benazzouz, "Spectral proper orthogonal decomposition and machine learning algorithms for bearing fault diagnosis," *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, vol. 45, no. 10, p. 550, 2023.

[8] A. Afia, F. Gougam, C. Rahmoune, W. Touzout, H. Ouelmokhtar, and D. Benazzouz, "Gearbox fault diagnosis using remd, eo and machine learning classifiers," *Journal of Vibration Engineering & Technologies*, vol. 12, no. 3, pp. 4673–4697, 2024.

[9] C. Vidal, P. Malysz, P. Kollmeyer, and A. Emadi, "Machine learning applied to electrified vehicle battery state of charge and state of health estimation: State-of-the-art," *IEEE Access*, vol. 8, pp. 52 796–52 814, 2020.

[10] M. Li, C. Li, Q. Zhang, W. Liao, and Z. Rao, "State of charge estimation of li-ion batteries based on deep learning methods and particle-swarm-optimized kalman filter," *Journal of Energy Storage*, vol. 64, p. 107191, 2023.

[11] A. Afia, F. Gougam, C. Rahmoune, W. Touzout, H. Ouelmokhtar, and D. Benazzouz, "Intelligent fault classification of air compressors using harris hawks optimization and machine learning algorithms," *Transactions of the Institute of Measurement and Control*, vol. 46, no. 2, pp. 359–378, 2024.

[12] V. S. Naresh, G. V. Ratnakara Rao, and D. V. N. Prabhakar, "Predictive machine learning in optimizing the performance of electric vehicle batteries: Techniques, challenges, and solutions," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, p. e1539, 2024.

[13] B. Saha and K. Goebel, "Battery data set," NASA AMES prognostics data repository, 2007.

[14] K. S. Ng, C.-S. Moo, Y.-P. Chen, and Y.-C. Hsieh, "Enhanced coulomb counting method for estimating state-of-charge and state-of-health of lithium-ion batteries," *Applied Energy*, vol. 86, no. 9, pp. 1506–1511, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306261908003061

[15] E. Meissner and G. Richter, "Battery monitoring and electrical energy management: Precondition for future vehicle electric power systems," *Journal of Power Sources*, vol. 116, no. 1, pp. 79–98, 2003, selected Papers Presented at the Eighth European Lead Battery Conference. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378775302007139

[16] M. Rani and S. Jaiswal, "A brief review of different estimation methods of soc for li-ion battery," in *Sustainable Technology and Advanced Computing in Electrical Engineering*, V. Mahajan, A. Chowdhury, N. P. Padhy, and F. Lezama, Eds. Singapore: Springer Nature Singapore, 2022, pp. 543–556.

[17] I. Snihir, W. Rey, E. Verbitskiy, A. Belfadhel-Ayeb, and P. H. Notten, "Battery open-circuit voltage estimation by a method of statistical analysis," *Journal of Power Sources*, vol. 159, no. 2, pp. 1484–1487, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378775305016460

[18] D. Tingting, L. Jun, Z. Fuquan, Y. Yi, and J. Qiqian, "Analysis on the influence of measurement error on state of charge estimation of lifepo4 power battery," in *2011 International Conference on Materials for Renewable Energy Environment*, vol. 1, 2011, pp. 644–649.

[19] X. Tang, Y. Wang, and Z. Chen, "A method for state-of-charge estimation of lifepo4 batteries based on a dual-circuit state observer," *Journal of Power Sources*, vol. 296, pp. 23–29, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378775315300756

[20] L. Zheng, L. Zhang, J. Zhu, G. Wang, and J. Jiang, "Co-estimation of state-of-charge, capacity and resistance for lithium-ion batteries based on a high-fidelity electrochemical model," *Applied Energy*, vol. 180, pp. 424–434, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306261916311035

[21] M. A. Roscher and D. U. Sauer, "Dynamic electric behavior and open-circuit-voltage modeling of lifepo4-based lithium ion secondary batteries," *Journal of Power Sources*, vol. 196, no. 1, pp. 331–336, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378775310010852

[22] Y.-P. Yang, J.-J. Liu, and C.-H. Tsai, "Improved estimation of residual capacity of batteries for electric vehicles," *Journal of the Chinese Institute of Engineers*, vol. 31, no. 2, pp. 313–322, 2008.

[23] W. Waag and D. U. Sauer, "Adaptive estimation of the electromotive force of the lithium-ion battery after current interruption for an accurate state-of-charge and capacity determination," *Applied Energy*, vol. 111, pp. 416–427, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306261913003905

[24] Y. Zhang, W. Song, S. Lin, and Z. Feng, "A novel model of the initial state of charge estimation for lifepo4 batteries," *Journal of Power Sources*, vol. 248, pp. 1028–1033, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S037877531301642X

[25] K. S. Ng, C.-S. Moo, Y.-P. Chen, and Y.-C. Hsieh, "Enhanced coulomb counting method for estimating state-of-charge and state-of-health of lithium-ion batteries," *Applied Energy*, vol. 86, no. 9, pp. 1506–1511, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306261908003061

[26] F. Leng, C. M. Tan, R. Yazami, and M. D. Le, "A practical framework of electrical based online state-of-charge estimation of lithium ion batteries," *Journal of Power Sources*, vol. 255, pp. 423–430, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378775314000354

[27] H. Wang, Y. Liu, H. Fu, and G. Li, "Estimation of state of charge of batteries for electric vehicles," *International Journal of Control and Automation*, vol. 6, no. 2, pp. 185–194, 2013.

[28] L. Lu, X. Han, J. Li, J. Hua, and M. Ouyang, "A review on the key issues for lithium-ion battery management in electric vehicles," *Journal of Power Sources*, vol. 226, pp. 272–288, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378775312016163

[29] M. Li, "Li-ion dynamics and state of charge estimation," *Renewable Energy*, vol. 100, pp. 44–52, 2017.

[30] L. Ran, W. Junfeng, W. Haiying, and L. Gechen, "Prediction of state of charge of lithium-ion rechargeable battery with electrochemical impedance spectroscopy theory," in *2010 5th IEEE Conference on Industrial Electronics and Applications*. IEEE, 2010, pp. 684–688.

[31] M. Coleman, C. K. Lee, C. Zhu, and W. G. Hurley, "State-of-charge determination from emf voltage estimation: Using impedance, terminal voltage, and current for lead-acid and lithium-ion batteries," *IEEE Transactions on industrial electronics*, vol. 54, no. 5, pp. 2550–2557, 2007.

[32] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.", 2022.

[33] "Machine learning algorithms," Available: https://www.geeksforgeeks.org/machine-learning-algorithms/, (Jun 19, 2024), [Accessed: 30-May-2024].

[34] M. Harrison, *Machine learning pocket reference: working with structured data in python*. O'Reilly Media, 2019.

[35] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009, vol. 2.

[36] V. Kanade, "What is linear regression? types, equation, examples, and best practices for 2022," Available: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-linear-regression/, (Apr 3, 2023), [Accessed: 30-May-2024].

[37] J. Ranstam and J. A. Cook, "Lasso regression," *British Journal of Surgery*, vol. 105, no. 10, pp. 1348–1348, 08 2018. [Online]. Available: https://doi.org/10.1002/bjs.10895

[38] A. Navlani, "Decision tree classification in python tutorial," Available: https://www.datacamp.com/tutorial/decision-tree-classification-python, (Feb 2023), [Accessed: 24-May-2024].

[39] T. M. Mitchell, "Artificial neural networks," *Machine learning*, vol. 45, no. 81, p. 127, 1997.

[40] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, Aug. 1996. [Online]. Available: https://doi.org/10.1007/BF00058655

[41] V. Kanade, "What is a decision tree? algorithms, template, examples, and best practices," Available: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-decision-tree/, (May 30, 2022), [Accessed: 30-May-2024].

[42] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[43] Y. Amit and D. Geman, "Shape quantization and recognition with randomized trees," *Neural computation*, vol. 9, no. 7, pp. 1545–1588, 1997.

[44] A. Cutler, D. R. Cutler, and J. R. Stevens, "Random forests," *Ensemble machine learning: Methods and applications*, pp. 157–175, 2012.

[45] "Random forest algorithm," Available: https://www.javatpoint.com/machine-learning-random-forest-algorithm, (2022), [Accessed: 23-May-2024].

[46] Binoy, "Random forest algorithm in machine learning," Available: https://www.scaler.com/topics/machine-learning/random-forest-algorithm/, (May 4, 2023), [Accessed: 30-May-2024].

[47] "Boosting in machine learning — boosting and adaboost," Available: https://www.geeksforgeeks.org/boosting-in-machine-learning-boosting-and-adaboost/, (May 23, 2023), [Accessed: 30-May-2024].

[48] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[49] T. Zhang, *Pattern Recognition and Machine Learning*. Springer, 2010.

[50] N. Verma, "Understanding k-nearest neighbors (knn) regression in machine learning," Available: https://medium.com/@nandiniverma78988/understanding-k-nearest-neighbors-knn-regression-in-machine-learning-c751a7cf516c, (Nov 5, 2023), [Accessed: 27-May-2024].

[51] R. Alake, "Loss functions in machine learning explained," Available: https://www.datacamp.com/tutorial/loss-function-in-machine-learning, (2023), [Accessed: 16-May-2024].

[52] J. Fernando, "R-squared: Definition, calculation formula, uses, and limitations," Available: https://www.investopedia.com/terms/r/r-squared.asp, (May 02, 2024), [Accessed: 17-May-2024].

[53] "Bayesian optimization." Available: https://towardsdatascience.com/automated-machine-learning-hyperparameter-tuning-in-python-dfda59b72f8a, (2018), [Accessed: 07-june-2024].

[54] J. Meng, M. Ricco, G. Luo, M. Swierczynski, D. I. Stroe, A. I. Stroe, and R. Teodorescu, "An overview and comparison of online implementable soc estimation methods for lithium-ion battery," *IEEE Transactions on Industry Applications*, vol. 54, no. 2, pp. 1583–1591, 2017.

[55] R. Li, W. J. Verhagen, and R. Curran, "A comparative study of data-driven prognostic approaches: Stochastic and statistical models," in *2018 IEEE International Conference on Prognostics and Health Management (ICPHM)*. IEEE, Jun. 2018, pp. 1–8.

[56] G. Fathoni, S. A. Widayat, P. A. Topan, A. Jalil, A. I. Cahyadi, and O. Wahyunggoro, "Comparison of state-of-charge (soc) estimation performance based on three popular methods: Coulomb counting, open circuit voltage, and kalman filter," in *2017 2nd International Conference on Automation, Cognitive Science, Optics, Micro Electro-Mechanical System, and Information Technology (ICACOMIT)*. IEEE, Oct. 2017, pp. 70–74.

[57] P. Shrivastava, T. K. Soon, M. Y. I. B. Idris, and S. Mekhilef, "Overview of model-based online state-of-charge estimation using kalman filter family for lithium-ion batteries," *Renewable and Sustainable Energy Reviews*, vol. 113, p. 109233, 2019.

[58] F. Gougam, A. Afia, M. A. Aitchikh, W. Touzout, C. Rahmoune, and D. Benazzouz, "Computer numerical control machine tool wear monitoring through a data-driven approach," *Advances in Mechanical Engineering*, vol. 16, no. 2, p. 16878132241229314, 2024.

[59] Z. Cui, W. Hu, G. Zhang, Z. Zhang, and Z. Chen, "An extended kalman filter based soc estimation method for li-ion battery," *Energy Reports*, vol. 8, pp. 81–87, 2022.

[60] H. S. Ramadan, M. Becherif, and F. Claude, "Extended kalman filter for accurate state of charge estimation of lithium-based batteries: a comparative analysis," *International journal of hydrogen energy*, vol. 42, no. 48, pp. 29 033–29 046, 2017.

[61] I. Baccouche, S. Jemmali, B. Manai, N. Omar, and N. Essoukri Ben Amara, "Improved ocv model of a li-ion nmc battery for online soc estimation using the extended kalman filter," *Energies*, vol. 10, no. 6, p. 764, 2017.

[62] J. N. Shen, J. J. Shen, Y. J. He, and Z. F. Ma, "Accurate state of charge estimation with model mismatch for li-ion batteries: A joint moving horizon

estimation approach," *IEEE Transactions on Power Electronics*, vol. 34, no. 5, pp. 4329–4342, 2018.

[63] Z. Zhang, B. Xue, and J. Fan, "Noise adaptive moving horizon estimation for state-of-charge estimation of li-ion battery," *IEEE Access*, vol. 9, pp. 5250–5259, 2020.

[64] C. Unterrieder, C. Zhang, M. Lunglmayr, R. Priewasser, S. Marsili, and M. Huemer, "Battery state-of-charge estimation using approximate least squares," *Journal of Power Sources*, vol. 278, pp. 274–286, 2015.

[65] Z. Yun, W. Qin, and W. Shi, "State of charge estimation of lithium-ion battery under time-varying noise based on variational bayesian estimation methods," *Journal of Energy Storage*, vol. 52, p. 104916, 2022.

[66] J. M. D. S. Jeewandara, J. P. Karunadasa, and K. T. M. U. Hemapala, "Soc level estimation of lithium-ion battery based on time series forecasting algorithms for battery management system," in *2021 3rd International Conference on Electrical Engineering (EECon)*. IEEE, Sep. 2021, pp. 43–49.

[67] A. J. Hasan, J. Yusuf, and R. B. Faruque, "Performance comparison of machine learning methods with distinct features to estimate battery soc," in *2019 IEEE Green Energy and Smart Systems Conference (IGESSC)*. IEEE, Nov. 2019, pp. 1–5.

[68] F. A. Barrios, J. Di Donato, C. Vidal, N. Chemmanoor, R. Ahmed, A. Emadi, and S. Habibi, "Comparing traditional and machine learning models for battery soc calculation," in *2022 IEEE Transportation Electrification Conference & Expo (ITEC)*. IEEE, Jun. 2022, pp. 125–130.

[69] J. Li, M. Ye, W. Meng, X. Xu, and S. Jiao, "A novel state of charge approach of lithium ion battery using least squares support vector machine," *IEEE Access*, vol. 8, pp. 195 398–195 410, 2020.

[70] N. Yang, Z. Song, H. Hofmann, and J. Sun, "Robust state of health estimation of lithium-ion batteries using convolutional neural network and random forest," *Journal of Energy Storage*, vol. 48, p. 103857, 2022.

[71] T. Talluri, H. T. Chung, and K. Shin, "Study of battery state-of-charge estimation with knn machine learning method," *IEIE Transactions on Smart Processing & Computing*, vol. 10, no. 6, pp. 496–504, 2021.

[72] E. Ipek, M. K. Eren, and M. Yilmaz, "State-of-charge estimation of li-ion battery cell using support vector regression and gradient boosting techniques," in *2019 International Aegean Conference on Electrical Machines and Power Electronics (ACEMP) & 2019 International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*, Aug. 2019, pp. 604–609.

[73] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," *Advances in neural information processing systems*, vol. 31, 2018.

[74] E. Chemali, P. J. Kollmeyer, M. Preindl, and A. Emadi, "State-of-charge estimation of li-ion batteries using deep neural networks: A machine learning approach," *Journal of Power Sources*, vol. 400, pp. 242–255, 2018.

[75] M. A. Hannan, D. N. How, M. H. Lipu, P. J. Ker, Z. Y. Dong, M. Mansur, and F. Blaabjerg, "Soc estimation of li-ion batteries with learning rate-optimized deep fully convolutional network," *IEEE Transactions on Power Electronics*, vol. 36, no. 7, pp. 7349–7353, 2020.

[76] J. Tian, C. Chen, W. Shen, F. Sun, and R. Xiong, "Deep learning framework for lithium-ion battery state of charge estimation: Recent advances and future perspectives," *Energy Storage Materials*, p. 102883, 2023.

[77] X. Fan, W. Zhang, C. Zhang, A. Chen, and F. An, "Soc estimation of li-ion battery using convolutional neural network with u-net architecture," *Energy*, vol. 256, p. 124612, 2022.

[78] M. H. Lipu, M. A. Hannan, A. Hussain, A. Ayob, M. H. Saad, T. F. Karim, and D. N. How, "Data-driven state of charge estimation of lithium-ion batteries: Algorithms, implementation factors, limitations and future trends," *Journal of Cleaner production*, vol. 277, p. 124110, 2020.

[79] H. Yu, L. Zhang, W. Wang, S. Li, S. Chen, S. Yang, and X. Liu, "State of charge estimation method by using a simplified electrochemical model in deep learning framework for lithium-ion batteries," *Energy*, vol. 278, p. 127846, 2023.

[80] J. Meng, G. Luo, and F. Gao, "Lithium polymer battery state-of-charge estimation based on adaptive unscented kalman filter and support vector machine," *IEEE Transactions on Power Electronics*, vol. 31, no. 3, pp. 2226–2238, 2015.

[81] M. H. Lipu, M. A. Hannan, A. Hussain, S. Ansari, S. A. Rahman, M. H. Saad, and K. M. Muttaqi, "Real-time state of charge estimation of lithium-ion batteries using optimized random forest regression algorithm," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 639–648, 2022.

[82] S. Deb, A. K. Goswami, R. L. Chetri, and R. Roy, "Prediction of plug-in electric vehicle's state-of-charge using gradient boosting method and random forest method," in *2020 IEEE international conference on power electronics, drives and energy systems (PEDES)*. IEEE, Dec. 2020, pp. 1–6.

[83] F. Mohammadi, "Lithium-ion battery state-of-charge estimation based on an improved coulomb-counting algorithm and uncertainty evaluation," *Journal of Energy Storage*, vol. 48, p. 104061, 2022.

[84] J. Lee and J. Won, "Enhanced coulomb counting method for soc and soh estimation based on coulombic efficiency," *IEEE Access*, vol. 11, pp. 15 449–15 459, 2023.

[85] F. Zheng, Y. Xing, J. Jiang, B. Sun, J. Kim, and M. Pecht, "Influence of different open circuit voltage tests on state of charge online estimation for lithium-ion batteries," *Applied energy*, vol. 183, pp. 513–525, 2016.

[86] O. von Kessel, T. Deich, S. Hahn, F. Brauchle, D. Vrankovic, T. Soczka-Guth, and K. P. Birke, "Mechanical impedance as a tool for electromechanical investigation and equivalent modeling of lithium-ion batteries," *Journal of Power Sources*, vol. 508, p. 230337, 2021.

[87] Y. Jiang, J. Xu, M. Liu, and X. Mei, "An electromechanical coupling model-based state of charge estimation method for lithium-ion pouch battery modules," *Energy*, vol. 259, p. 125019, 2022.

[88] K. Sarrafan, K. M. Muttaqi, and D. Sutanto, "Real-time estimation of model parameters and state-of-charge of li-ion batteries in electric vehicles using a new mixed estimation model," *IEEE Transactions on Industry Applications*, vol. 56, no. 5, pp. 5417–5428, 2020.

[89] M. Landi and G. Gross, "Measurement techniques for online battery state of health estimation in vehicle-to-grid applications," *IEEE Transactions on Instrumentation and Measurement*, vol. 63, no. 5, pp. 1224–1234, 2014.

[90] Z. Dou, J. Li, H. Yan, C. Zhang, and F. Liu, "Real-time online estimation technology and implementation of state of charge state of uncrewed aerial vehicle lithium battery," *Energies*, vol. 17, no. 4, p. 803, 2024.

[91] T. Talluri, H. T. Chung, and K. Shin, "Study of battery state-of-charge estimation with knn machine learning method," *IEIE Transactions on Smart Processing & Computing*, vol. 10, no. 6, pp. 496–504, 2021.

[92] E. Ipek, M. K. Eren, and M. Yilmaz, "State-of-charge estimation of li-ion battery cell using support vector regression and gradient boosting techniques," in *2019 International Aegean Conference on Electrical Machines and Power Electronics (ACEMP) & 2019 International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*. IEEE, Aug. 2019, pp. 604–609.

[93] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," in *Advances in neural information processing systems*, vol. 31, 2018.

[94] M. A. Hannan, D. N. How, M. H. Lipu, P. J. Ker, Z. Y. Dong, M. Mansur, and F. Blaabjerg, "Soc estimation of li-ion batteries with learning rate-optimized deep fully convolutional network," *IEEE Transactions on Power Electronics*, vol. 36, no. 7, pp. 7349–7353, 2020.

[95] J. Tian, C. Chen, W. Shen, F. Sun, and R. Xiong, "Deep learning framework for lithium-ion battery state of charge estimation: Recent advances and future perspectives," *Energy Storage Materials*, p. 102883, 2023.

[96] X. Fan, W. Zhang, C. Zhang, A. Chen, and F. An, "Soc estimation of li-ion battery using convolutional neural network with u-net architecture," *Energy*, vol. 256, p. 124612, 2022.

[97] M. H. Lipu, M. A. Hannan, A. Hussain, A. Ayob, M. H. Saad, T. F. Karim, and D. N. How, "Data-driven state of charge estimation of lithium-ion batteries: Algorithms, implementation factors, limitations and future trends," *Journal of Cleaner production*, vol. 277, p. 124110, 2020.

[98] H. Yu, L. Zhang, W. Wang, S. Li, S. Chen, S. Yang, and X. Liu, "State of charge estimation method by using a simplified electrochemical model in deep learning framework for lithium-ion batteries," *Energy*, vol. 278, p. 127846, 2023.

[99] J. Meng, G. Luo, and F. Gao, "Lithium polymer battery state-of-charge estimation based on adaptive unscented kalman filter and support vector machine," *IEEE Transactions on Power Electronics*, vol. 31, no. 3, pp. 2226–2238, 2015.

[100] M. H. Lipu, M. A. Hannan, A. Hussain, S. Ansari, S. A. Rahman, M. H. Saad, and K. M. Muttaqi, "Real-time state of charge estimation of lithium-ion batteries using optimized random forest regression algorithm," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 639–648, 2022.

[101] S. Deb, A. K. Goswami, R. L. Chetri, and R. Roy, "Prediction of plug-in electric vehicle's state-of-charge using gradient boosting method and random forest method," in *2020 IEEE international conference on power electronics, drives and energy systems (PEDES)*. IEEE, Dec. 2020, pp. 1–6.

[102] M. K. Tran, S. Panchal, T. D. Khang, K. Panchal, R. Fraser, and M. Fowler, "Concept review of a cloud-based smart battery management system for lithium-ion batteries: Feasibility, logistics, and functionality," *Batteries*, vol. 8, no. 2, p. 19, 2022.

[103] D. N. How, M. A. Hannan, M. H. Lipu, and P. J. Ker, "State of charge estimation for lithium-ion batteries using model-based and data-driven methods: A review," *IEEE Access*, vol. 7, pp. 136 116–136 136, 2019.

[104] Z. Cui, L. Wang, Q. Li, and K. Wang, "A comprehensive review on the state of charge estimation for lithium-ion battery based on neural network," *International Journal of Energy Research*, vol. 46, no. 5, pp. 5423–5440, 2022.

[105] C. Staff, "What is python used for? a beginner's guide," Available: https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python, (Apr 3, 2024), [Accessed: 29-May-2024].

[106] Available: https://www.python.org/, [Accessed: 29-May-2024].

[107] N. Kumar, ""Data Wrangling: Removing Null Values from Dataset in Python using Pandas Library"," Available: https://theprofessionalspoint.blogspot.com/2019/03/data-wrangling-removing-null-values.html, (2019), [Accessed: 23-May-2024].

[108] "Training, Validation, Test Split for Machine Learning Datasets," Available: https://encord.com/blog/train-val-test-split, (June 13, 2023), [Accessed: 11-Jun-2024].