Electrical Machines & Drives
Elective Project & Computation Report

Estimation of State of Charge (SOC) of Lithium-Ion Batteries

Submitted by:
SAYANDEEP DEY

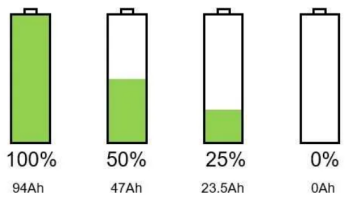
State of Charge (SOC) Estimation Using Machine Learning

Project Overview

Accurate estimation of the State of Charge (SOC) of lithium-ion batteries is essential for the efficient and safe operation of battery-powered systems. The SOC represents the remaining charge in a battery as a percentage of its total capacity. Effective SOC estimation ensures optimal battery usage, extends battery life, and prevents issues such as overcharging or deep discharging.

Traditional SOC estimation methods include:

- ***Coulomb Counting:*** SoC Estimation by Coulomb Counting is based on the measurement of the current and integration of that current over time.



The State of Charge (SoC) of a battery cell is required to maintain it's safe operation and lifetime during charge, discharge and storage. However, SoC cannot be measured directly and is estimated from other measurements and known parameters. This leads to errors in the estimated SoC and that means it is not possible to fully exploit the full capability of the cell.Coulomb counting gives a relative change in SoC and not an absolute SoC. If you measure the current over a given time step you have a measure of the number of Ah that have left or been received by the battery.

$$\boxed{SoC(t) = SoC(t-1) + \frac{I(t)}{Q_n}\Delta t}$$

where:

  - $SoC(t)$ = estimated State of Charge at time, t
  - $SoC(t-1)$ = previous State of Charge at time $t-1$
  - $I(t)$ = charging or discharging current at time, t
  - Q_{n} = battery cell capacity
  - Δt = time step between $t-1$ and t

- ***Open Circuit Voltage (OCV) Method:*** The Open Circuit Voltage (OCV) is a fundamental parameter of the cell. The OCV of a battery cell is the potential difference between the positive and negative terminals when no current flows and the cell is at rest.The typical lithium battery OCV curves versus SoC then looks like:

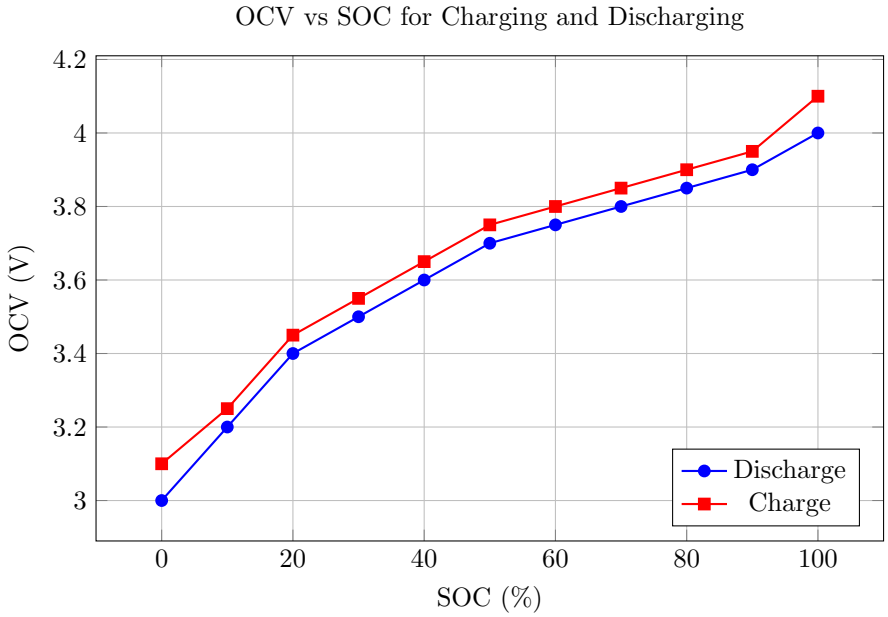


Figure 1: OCV vs SOC (%) for Charge and Discharge

- ***Model-based Methods:*** Use equivalent circuit or electrochemical models, which need detailed parameter tuning and can be complex.

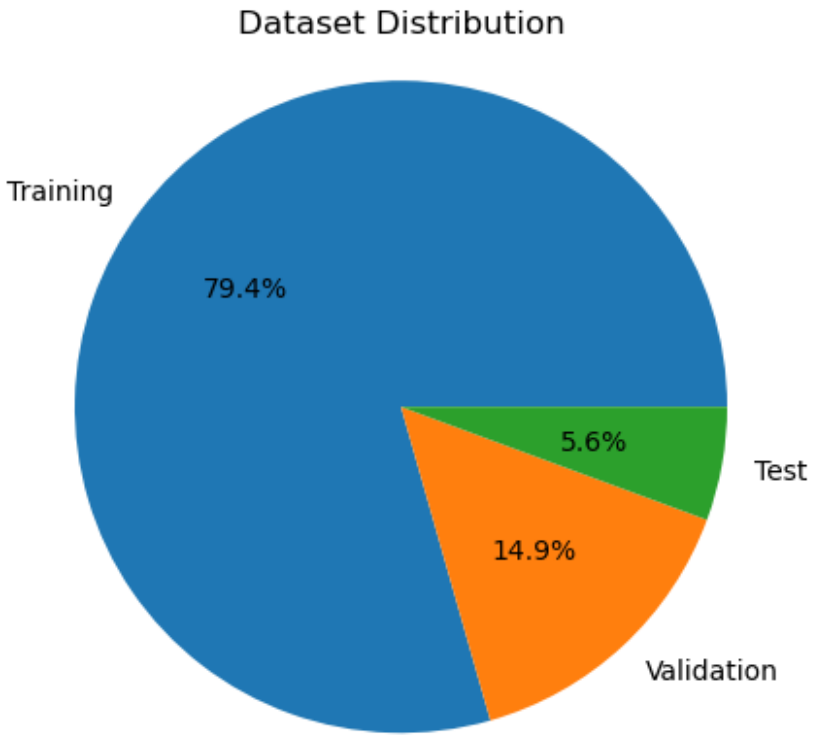
Machine learning offers a promising alternative by utilizing historical and real-time data to learn patterns and relationships that influence SOC. This approach reduces dependence on physical models and can adapt to battery aging and environmental changes.

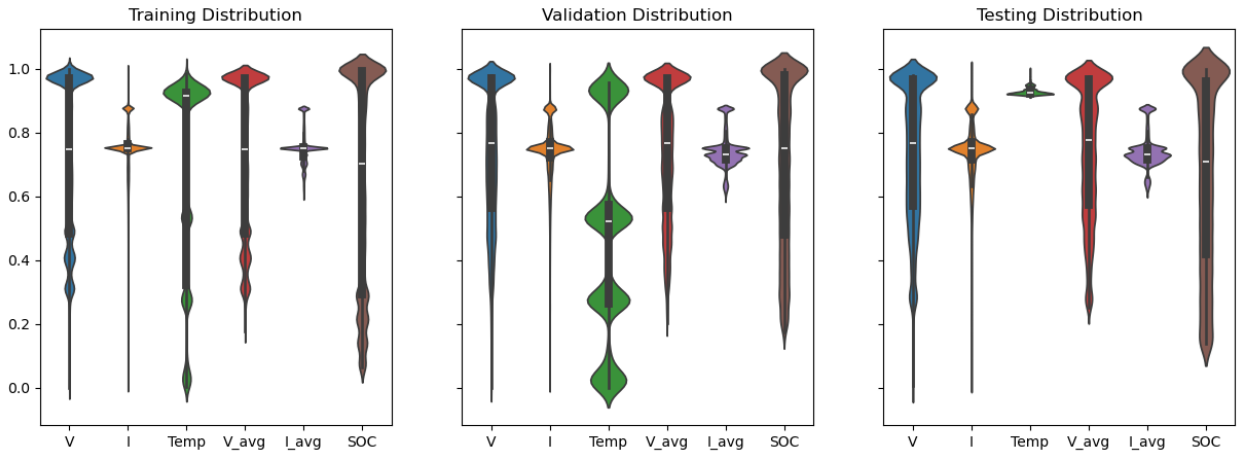
Dataset and Features

V	I	Temp	V_avg	I_avg	SOC
0.966960	0.748900	0.920678	0.966960	0.748900	1.000000
0.966020	0.746992	0.920677	0.966490	0.747946	0.999990
0.965901	0.746992	0.917845	0.966294	0.747628	0.999983
0.965783	0.747098	0.917845	0.966166	0.747496	0.999973
0.965665	0.746992	0.917845	0.966066	0.747395	0.999963
0.965547	0.746992	0.917845	0.965986	0.747313	0.999953
0.965429	0.746992	0.917845	0.965906	0.747255	0.999943
0.965311	0.746992	0.917845	0.965835	0.747210	0.999933
0.965193	0.746992	0.917845	0.965768	0.747174	0.999923
0.965075	0.746992	0.917845	0.965702	0.747145	0.999913
0.964957	0.746992	0.917845	0.965637	0.747120	0.999903
0.964839	0.746992	0.917845	0.965573	0.747099	0.999893
0.964721	0.746992	0.917845	0.965509	0.747082	0.999883
0.964603	0.746992	0.917845	0.965446	0.747068	0.999873
0.964485	0.746992	0.917845	0.965384	0.747056	0.999863
0.964367	0.746992	0.917845	0.965322	0.747045	0.999853
0.964249	0.746992	0.917845	0.965260	0.747035	0.999843
0.964131	0.746992	0.917845	0.965198	0.747026	0.999833
0.964013	0.746992	0.917845	0.965137	0.747017	0.999823
0.963895	0.746992	0.917845	0.965076	0.747008	0.999813

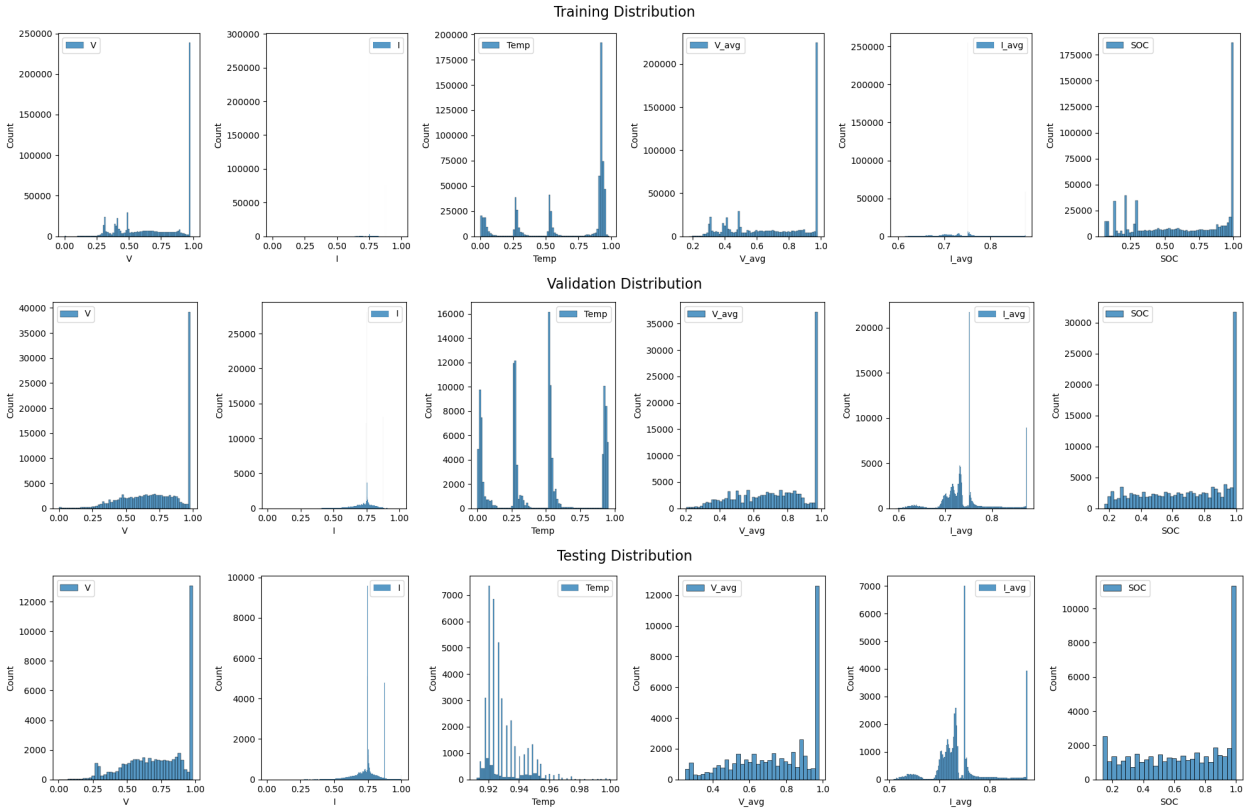
V	I	Temp	V_avg	I_avg	SOC
0.963777	0.746992	0.917845	0.965014	0.746999	0.999803
0.963659	0.746992	0.917845	0.964953	0.746990	0.999793
0.963541	0.746992	0.917845	0.964891	0.746981	0.999783
0.963423	0.746992	0.917845	0.964829	0.746972	0.999773
0.963305	0.746992	0.917845	0.964767	0.746962	0.999763
0.963187	0.746992	0.917845	0.964705	0.746951	0.999753
0.963069	0.746992	0.917845	0.964643	0.746939	0.999743
0.962951	0.746992	0.917845	0.964581	0.746926	0.999733
0.962833	0.746992	0.917845	0.964519	0.746912	0.999723
0.962715	0.746992	0.917845	0.964457	0.746897	0.999713
0.962597	0.746992	0.917845	0.964395	0.746881	0.999703
0.962479	0.746992	0.917845	0.964333	0.746864	0.999693
0.962361	0.746992	0.917845	0.964271	0.746846	0.999683
0.962243	0.746992	0.917845	0.964209	0.746827	0.999673
0.962125	0.746992	0.917845	0.964147	0.746808	0.999663
0.962007	0.746992	0.917845	0.964085	0.746788	0.999653
0.961889	0.746992	0.917845	0.964023	0.746768	0.999643
0.961771	0.746992	0.917845	0.963961	0.746747	0.999633
0.961653	0.746992	0.917845	0.963899	0.746726	0.999623
0.961535	0.746992	0.917845	0.963837	0.746704	0.999613
0.961417	0.746992	0.917845	0.963775	0.746681	0.999603
0.961299	0.746992	0.917845	0.963713	0.746658	0.999593
0.961181	0.746992	0.917845	0.963651	0.746635	0.999583
0.961063	0.746992	0.917845	0.963589	0.746611	0.999573
0.960945	0.746992	0.917845	0.963527	0.746586	0.999563
0.960827	0.746992	0.917845	0.963465	0.746561	0.999553
0.960709	0.746992	0.917845	0.963403	0.746535	0.999543
0.960591	0.746992	0.917845	0.963341	0.746508	0.999533

Table 1: First 50 rows of the dataset





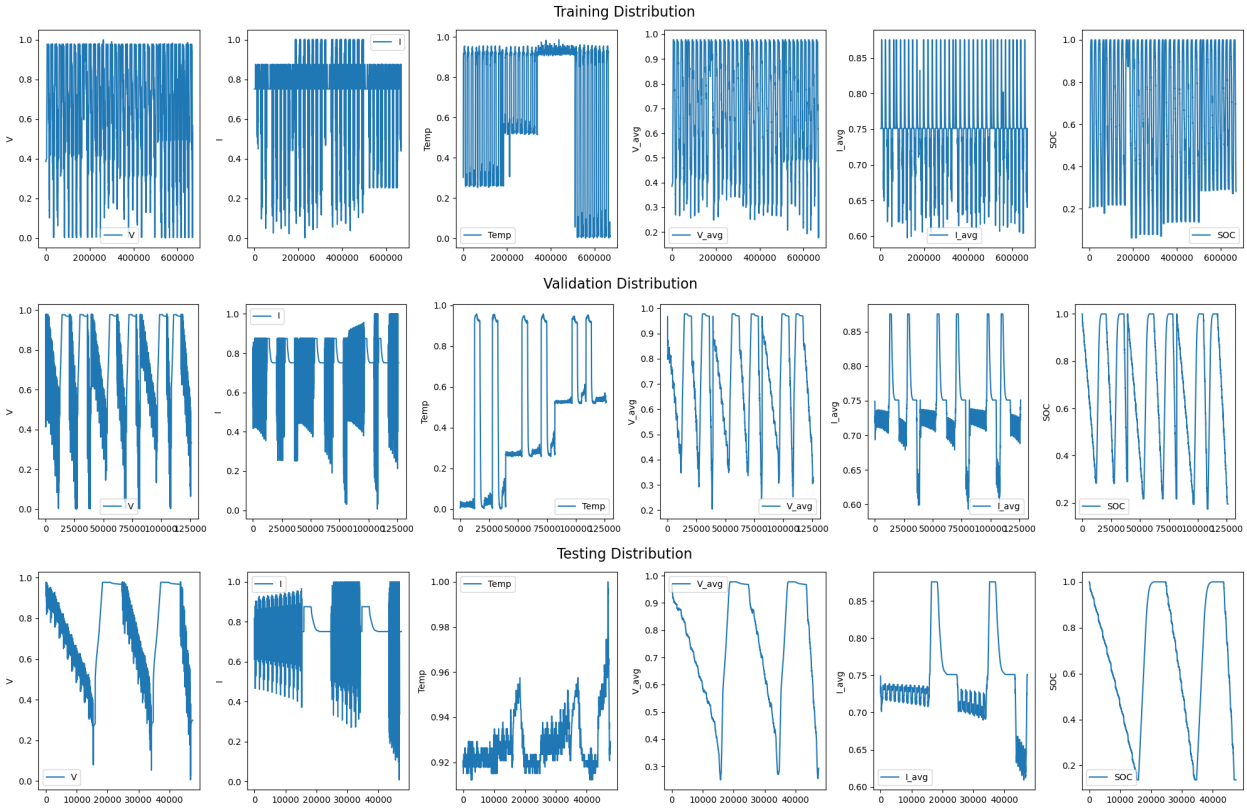
- **Features Used:**
  - `I_avg`: Average current
  - `V_avg`: Average voltage
  - `Temp`: Temperature
  - `SOC`: State of charge (target variable)



Data Preprocessing:

- Normalization and standardization of the input features.
- Splitting the data into training and testing sets.
- Handling missing values and outliers.

Time Series Plots:



Modeling and Evaluation

Metrics used for Model Evaluation

- **Max Error:**

$$\text{Max Error} = \max_i |Y_i - \hat{Y}_i|$$

- **Mean Absolute Error (MAE):**

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i|$$

- **Mean Absolute Percentage Error (MAPE):**

$$\text{MAPE} = 100 \cdot \frac{1}{n}\sum_{i=1}^{n}\left(\frac{|Y_i - \hat{Y}_i|}{Y_i}\right)$$

- **Mean Squared Error (MSE):**

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

- **Root Mean Squared Error (RMSE):**

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}$$

- **Root Mean Squared Log Error (RMSLE):**

$$\text{RMSLE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\log(1+Y_i) - \log(1+\hat{Y}_i)\right)^2}$$

Adaptive H-Infinity Filter

AHIF is a simplified Adaptive H-Infinity Filter based on Kalman gain. It is used later to filter out noisy SOC estimates predicted by trained models.

- The ***process variance*** is initially small because it represents model uncertainty. At the beginning, it is assumed that the model is very accurate and, therefore, the variance is set to a low value. This parameter will be adapted every ***10 measurements*** because 10 residuals are enough statistical information to predict model variance correctly. Specifically, if the residuals are highly variable, the model is inaccurate. If they are low constant values, the model variance can be reduced.
- The ***measurement variance*** is set to a high value because it is assumed that measurements from sensors are very noisy.
- The ***initial error covariance*** is initially set to ***1*** (high uncertainty) because, at first, it is assumed that there is high measurement uncertainty.
- The ***Kalman gain*** represents how much we can trust the measurement versus the prediction. If the measurement is very noisy (high measurement variance/uncertainty), the Kalman gain will be low, favoring the prediction.

  $$KG = \frac{\text{PEC}}{\text{PEC} + \text{MV}}$$

  Where:

  - $KG \rightarrow$ Kalman gain
  - $PEC \rightarrow$ Predicted error covariance
  - $MV \rightarrow$ Measurement variance
- The ***error covariance*** is updated with newly acquired information. The more data collected, the higher the prediction confidence (Bayesian Filter principles).

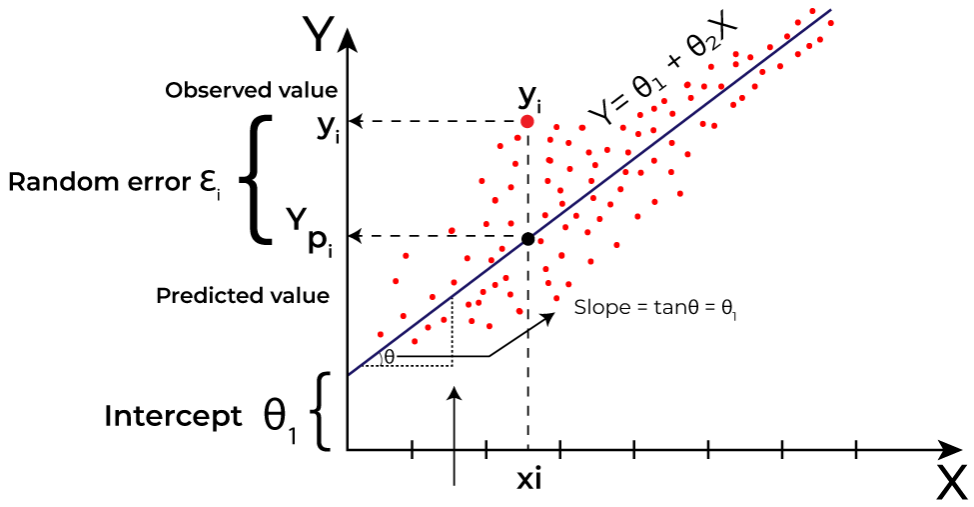
Algorithms Used

Linear Regression

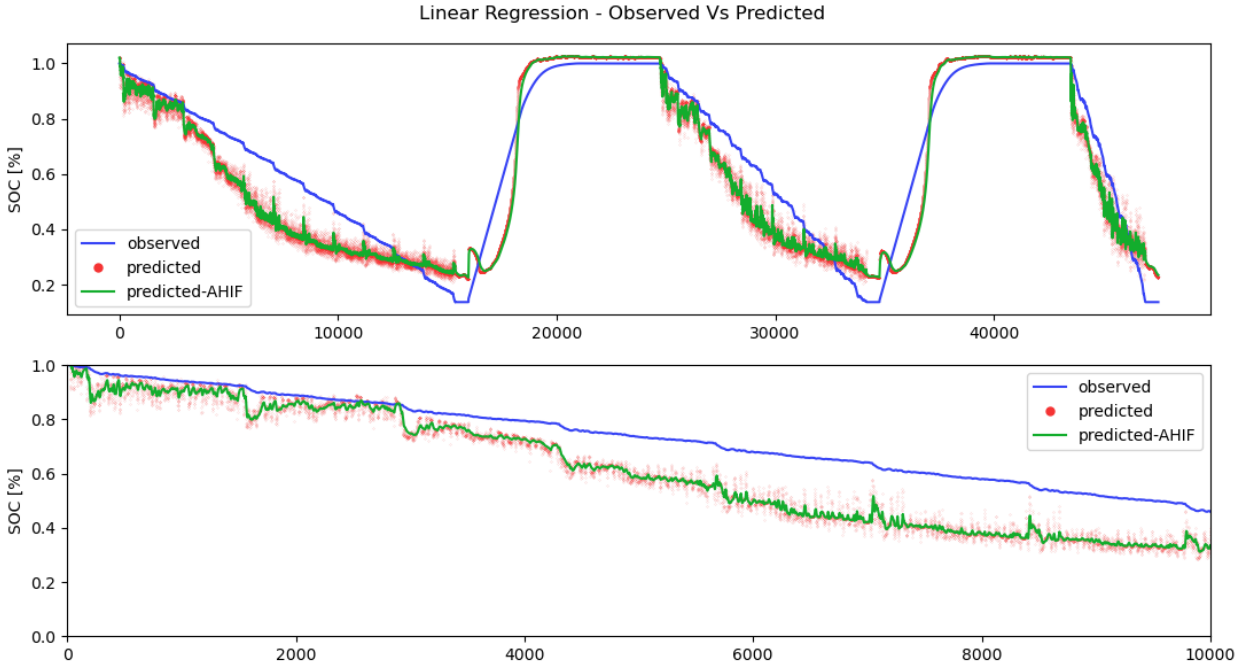
Linear Regression aims to establish a **linear relationship** between a dependent variable y and one or more independent variables x. In other words, the model attempts to find a **linear function** that improves the prediction of y given x.

Mathematical Representation:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon$$



Observed vs Predicted:



Decision Tree Regression

Decision Tree Regression is a *Machine Learning technique* that uses *tree structures* to model the relationship between *independent variables* and a *dependent variable*.

Structure:

A *Decision Tree* consists of:

- *Intermediate nodes*: Split the dataset into subgroups based on specific conditions.
- *Branches*: Represent the possible choices leading to the next nodes.
- *Leaf nodes*: Provide the *final prediction* value.

Splitting Criterion:

- At each node, the dataset is split to minimize the *mean squared error (MSE)* or another metric.
- The formula for *MSE* at a node is:

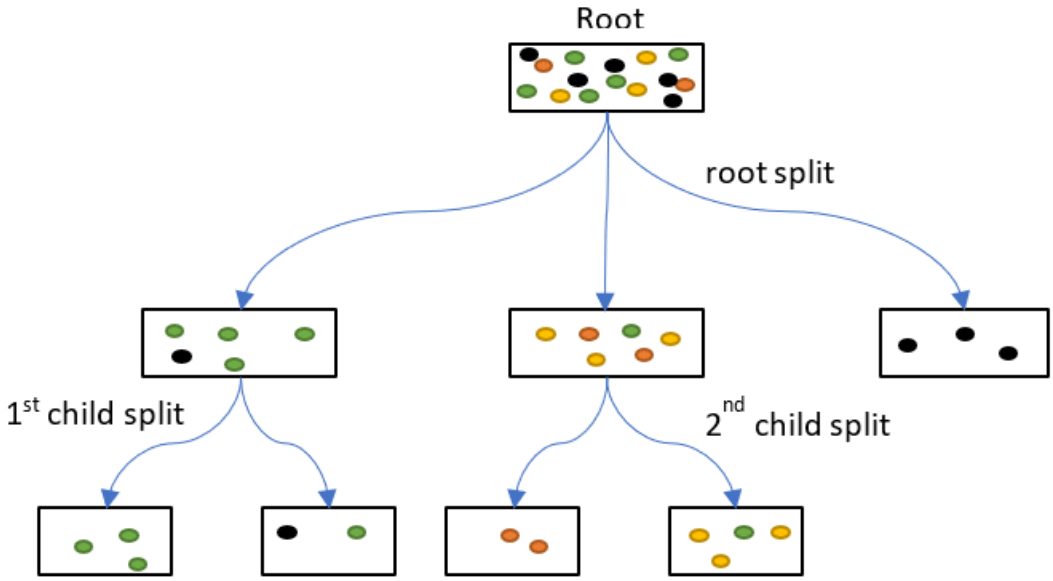
$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2$$

  - $n \rightarrow$ Number of samples at the node
  - $y_i \rightarrow$ Actual values
  - $\hat{y} \rightarrow$ Predicted value

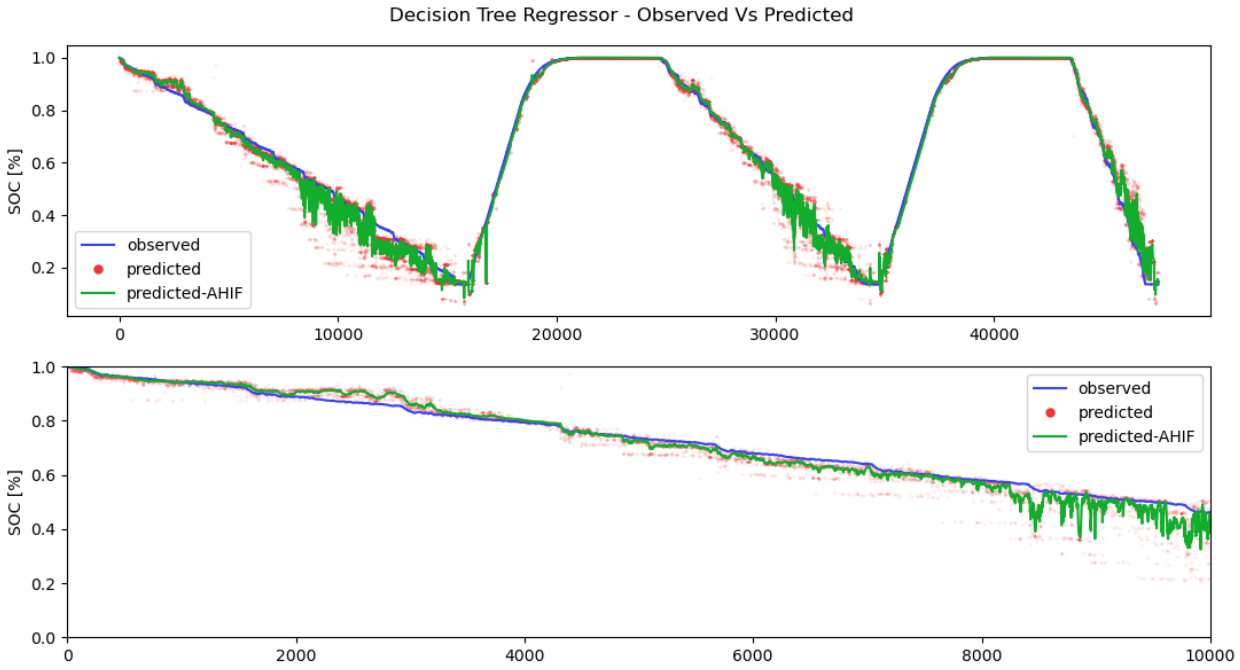
Prediction Formula:

- The predicted value at a *leaf node* is the *average* of the target values in that region:

$$\hat{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$



Observed vs Predicted



K-Nearest Neighbors Regression

K-Nearest Neighbors Regression (KNN) is a machine learning technique used for regression problems.

Distance Calculation:

$$d(x, x') = \sqrt{\sum_{i=1}^{n}(x_i - x_i')^2}$$

How it Works:

1. **Problem Definition:**
   - Start with a dataset containing *independent variables (features)* and a *numeric dependent variable (target).*
   - The goal is to predict the value of the *target* for new data points.
2. **Neighbor Selection:**
   - For a new data point, *KNN regression* identifies the K closest points in the training dataset.
   - Distance is typically calculated using *Euclidean distance:*

$$d(x, x') = \sqrt{\sum_{i=1}^{n}(x_i - x_i')^2}$$

3. **Aggregation of Results:**

- The predicted value for the new point is the *average* (or sometimes a weighted average) of the values of the nearest neighbors:

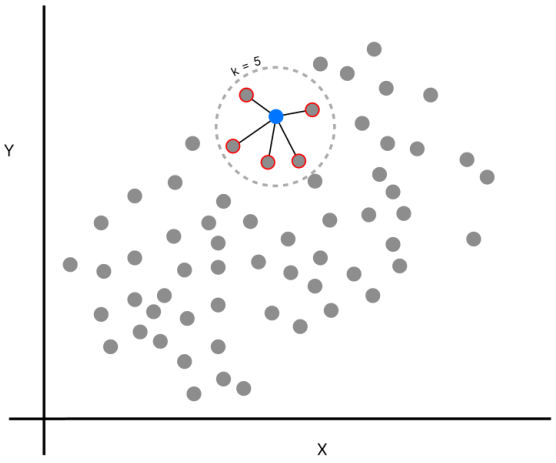
$$\hat{y} = \frac{1}{K} \sum_{i=1}^{K} y_i$$

- $K \rightarrow$ Number of nearest neighbors
- $y_i \rightarrow$ Target values of the K neighbors

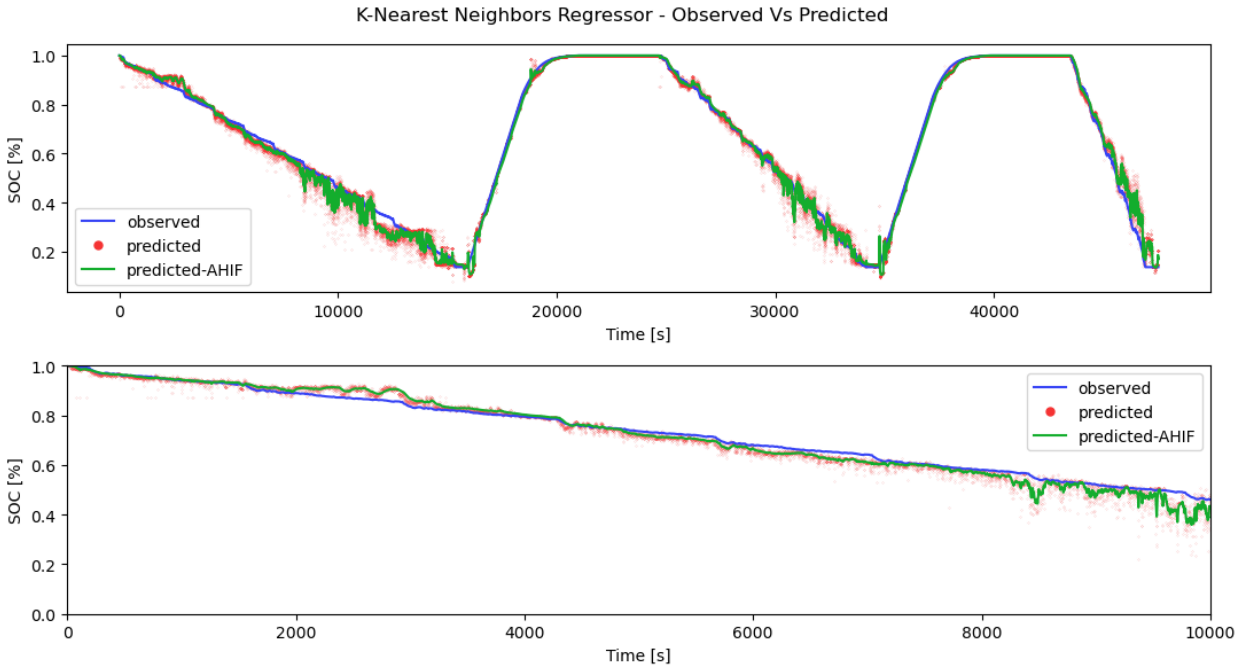
4. **Choice of K:**

- K is a *hyperparameter* that requires careful selection.
- A *low* K value $\rightarrow$ model is sensitive to noise.
- A *high* K value $\rightarrow$ model becomes too general, losing important details.
- The optimal value of K is often selected using *cross-validation* techniques.

Visualization:



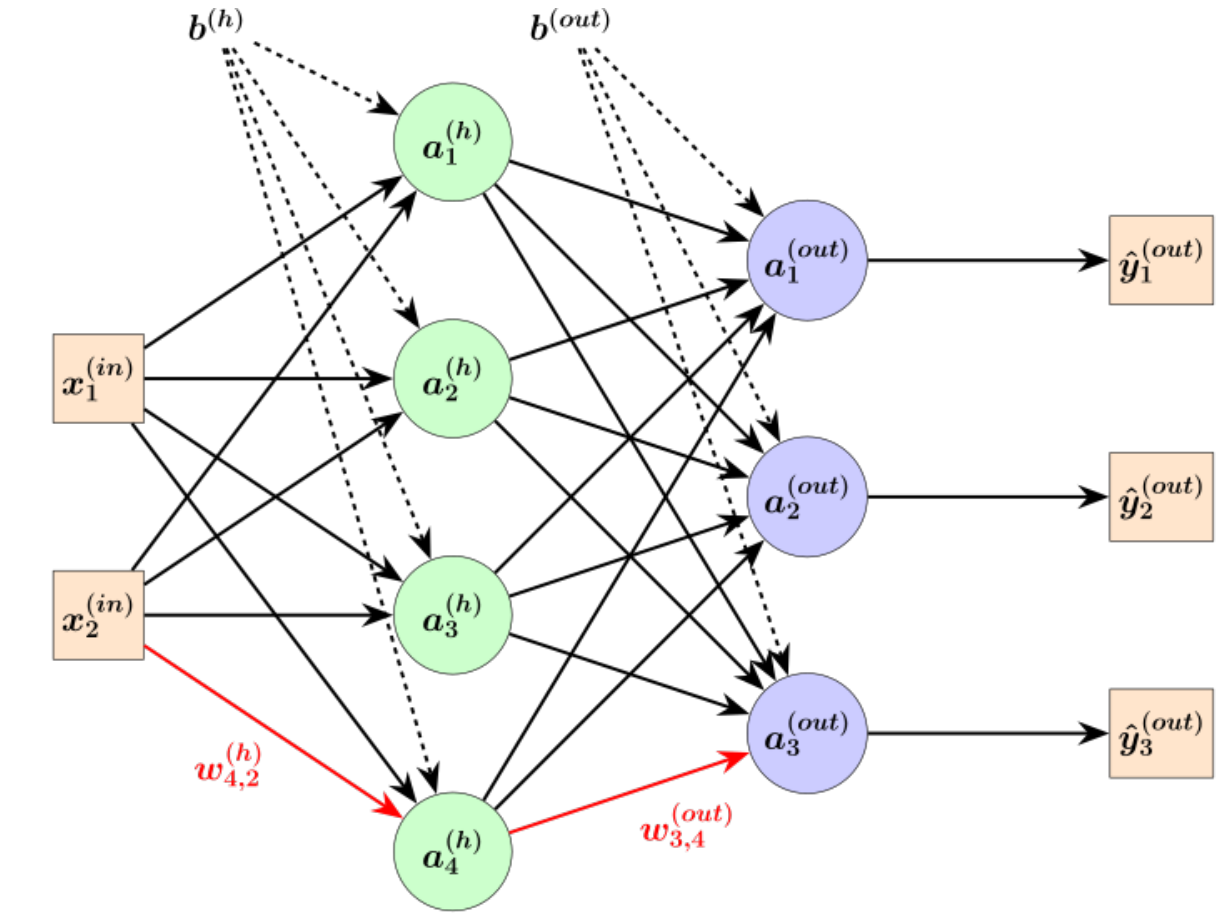
Observed vs Predicted



Multilayer Perceptron-Feedforward Neural Network (MLP-FNN)

A **FeedForward Neural Network (FNN)** is one of the simplest artificial neural networks used in the AI field. It is named '**feed-forward**' because data flows in one direction without loops and feedback. In particular, there are no recurrent connections, which distinguishes it from **RNN (Recurrent Neural Network)** where data can flow backward.

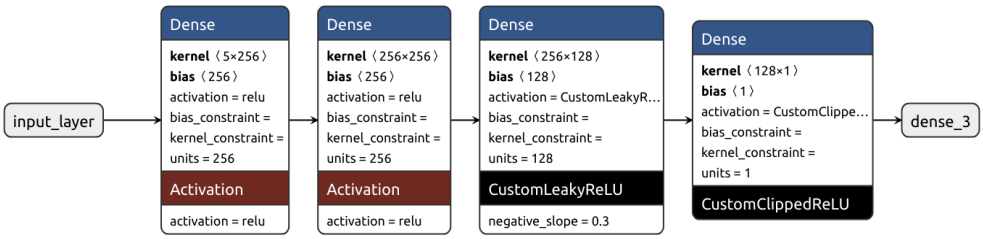
Specifically, a **Multilayer Perceptron (MLP)** is an FNN with at least one hidden layer between the input and output layers. This network is very useful for learning non-linear relationships.



Specifically, a class is designed to instantiate, build, and compile an MLP-FNN model where the input shape is equal to 5, i.e., 5 features (**V**, $\mathbf{V_{avg}}$, **I**, $\mathbf{I_{avg}}$, **Temp**)

In particular, this architecture consists of:

- ***1 input layer*** with 5 neurons
- ***3 hidden layers*** with 256, 256, and 128 neurons, respectively
- ***1 output layer*** with 1 neuron



Activation Functions

The hidden layers use different activation functions:

- ***ReLU (Rectified Linear Unit)*** for the first two hidden layers (256 neurons each)
- ***Leaky ReLU*** for the third hidden layer (128 neurons)
- ***Clipped ReLU*** for the output layer

ReLU:

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

Leaky ReLU:

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha \cdot x & \text{if } x \leq 0 \end{cases}$$

Where:

- α is the negative slope that allows a small gradient to pass through even for negative values, avoiding the ***dying ReLU problem***.

Clipped ReLU:

$$\text{Clipped ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \leq x < 1 \\ 1 & \text{if } x \geq 1 \end{cases}$$

Optimization and Loss Function

Stochastic Gradient Descent (SGD):

The optimizer chosen for the model is **SGD**, which updates architecture weights to decrease the error.

$$w_{ij} = w_{ij} - \eta \frac{\delta L}{\delta w_{ij}}$$

Where:

- η is the learning rate controlling the update rate
- $\frac{\delta L}{\delta w_{ij}}$ is the gradient of the loss function with respect to the weight

Decay Algorithm: Exponential Decay:

The decay algorithm used is **Exponential Decay**, which stabilizes the optimization process. The learning rate at each epoch t is updated according to:

$$\eta_t = \eta_0 \cdot e^{-\lambda t}$$

Where:

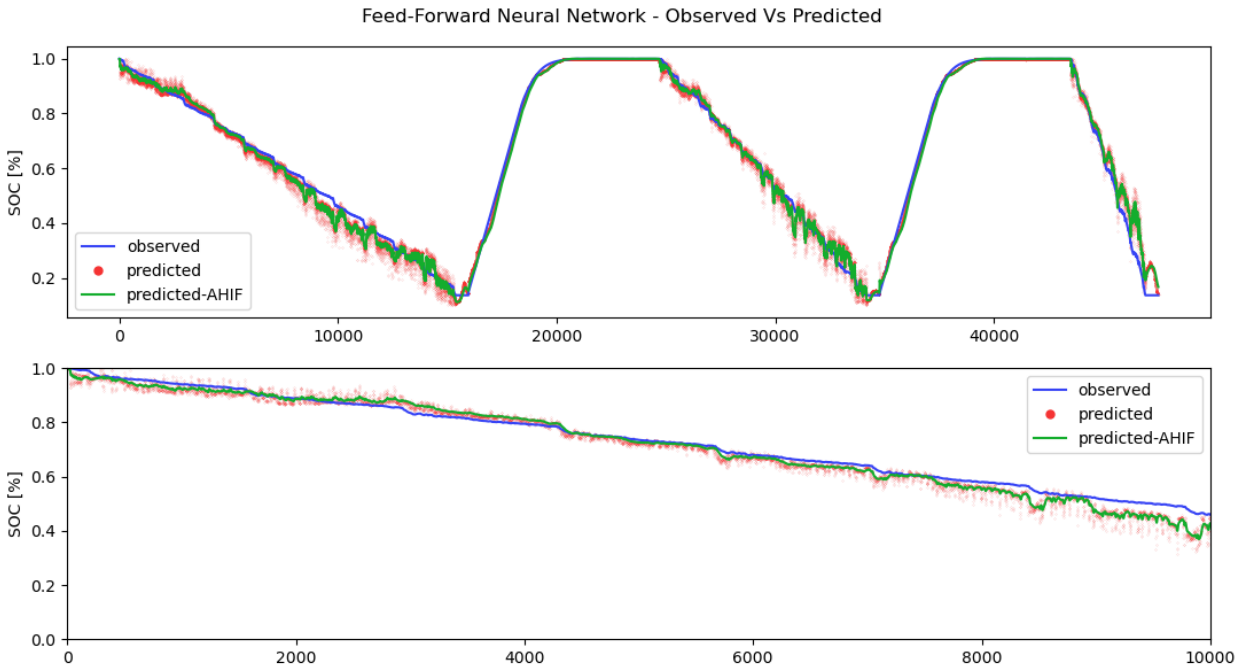
- η_0 is the initial learning rate
- λ is the decay constant
- t is the number of epochs

Loss Function: Mean Squared Error (MSE)

The loss function used to evaluate the model is **MSE**, aiming to minimize the difference between predicted and actual values.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Observed vs Predicted



Visualization and Analysis

Metrics Comparison Among FNN, Linear Regression, Decision Tree Regression, and K-Nearest Neighbors Regression

Best Model (Based on Minimum Error): **K-Nearest Neighbors Regression**

This model achieves the lowest error values across all metrics:

- **Max Error (MA)**: 17.4%
- **Mean Absolute Error (MAE)**: 1.8%
- **Mean Absolute Percentage Error (MAPE)**: 4.9%
- **Mean Squared Error (MSE)**: 0.1%
- **Root Mean Squared Error (RMSE)**: 2.8%
- **Root Mean Squared Log Error (RMSLE)**: 2%

Note: *K-Nearest Neighbors (KNN)* is a **fixed model**, meaning accuracy can only be improved by:

- Increasing the amount of training data

- Choosing a different value of K
  - Increasing K improves generalization but may cause loss of important local patterns

In this context, the **MLP-FNN (Multi-Layer Perceptron - Feedforward Neural Network)** can be considered a **better model**, despite higher error values, due to its **greater flexibility for improvement**.

Tuning Options for MLP-FNN:

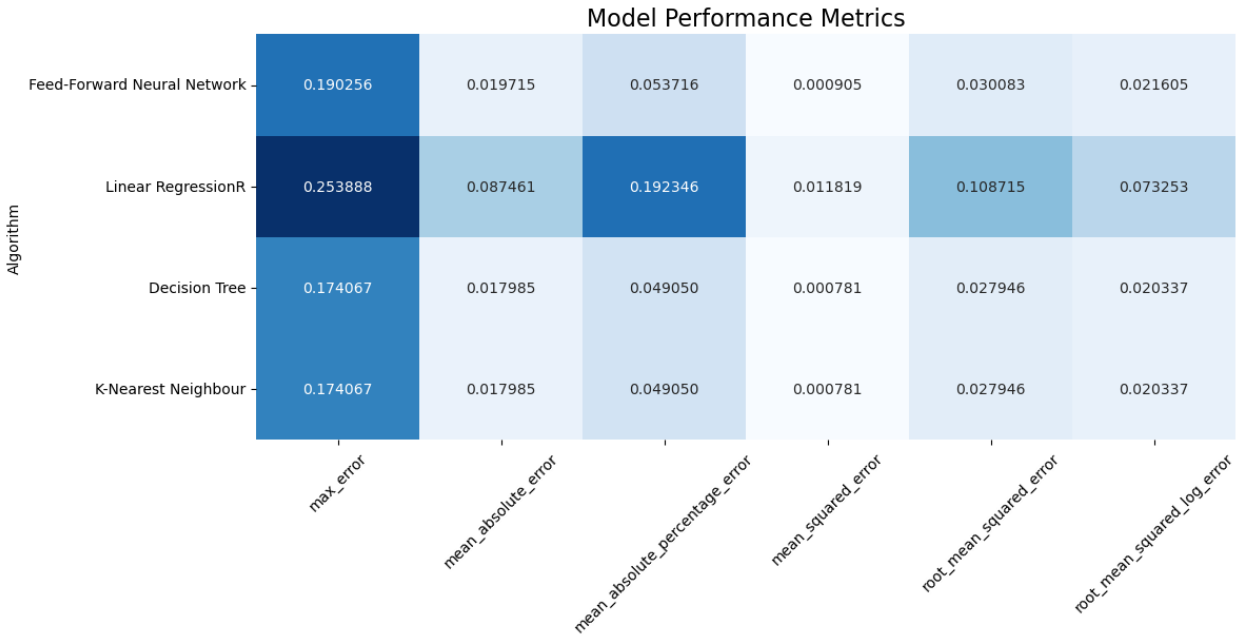
- Increase the number of epochs
- Add more hidden layers
- Try different activation functions
- Increase the number of neurons
- Use different optimization functions
- Apply various learning rate decay functions

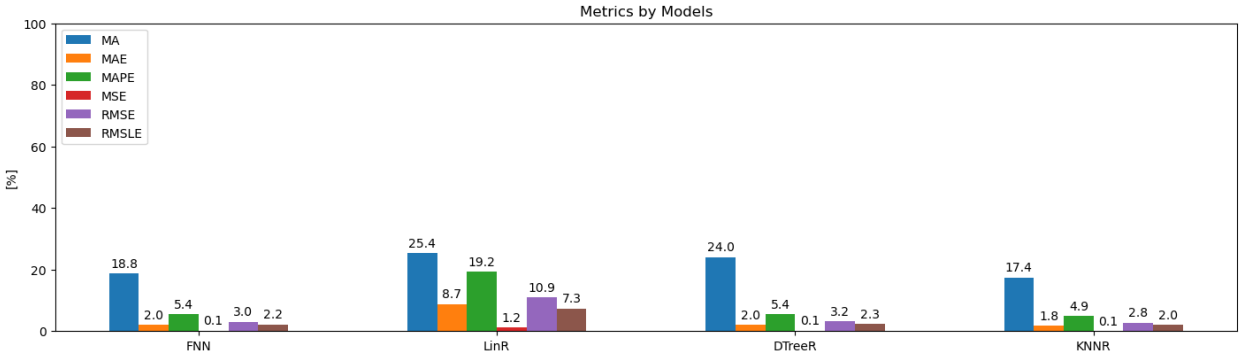
This flexibility makes **MLP-FNN** more adaptable for performance improvements than K-Nearest Neighbors Regression.

Correlation Matrix:

The correlation matrix highlights the relationship between the features and SOC. Features like $\mathbf{I_{avg}}$ and $\mathbf{V_{avg}}$ show stronger correlations with the SOC.

Model Performance Comparison:



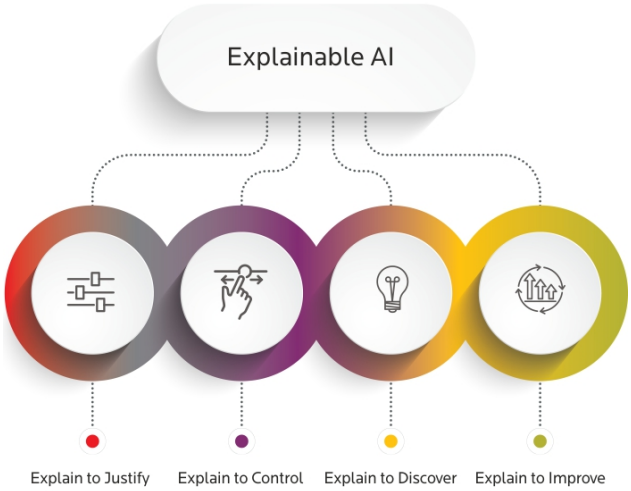


Feature Importance

Explainable Artificial Intelligence (XAI)

Considering ***MLP-FNN*** as the best model, we proceed to ***interpret the trained model***. ***Explainable Artificial Intelligence (XAI)*** focuses on developing techniques that make ***machine decisions understandable and transparent*** to humans. The goal is to ***improve the confidence and reliability*** of AI models by providing clear explanations of how and why certain predictions or decisions were made.

Visualization:



SHAP (SHapley Additive exPlanations):

-***SHAP*** is a technique within XAI, based on ***Shapley values*** from ***game theory***, which quantifies the contribution of each feature to the model's prediction.
- It assigns a ***SHAP value*** to each feature, representing its impact on the prediction.

SHAP Formula:

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} \big(f(S \cup \{i\}) - f(S)\big)$$

Where:

- $\phi_i \rightarrow$ SHAP value for feature i
- $F \rightarrow$ Set of all features
- $S \rightarrow$ Subset of features excluding i
- $f(S) \rightarrow$ Model prediction for the subset S

SHAP Value Calculation Breakdown

1. Sampling the Test Data:

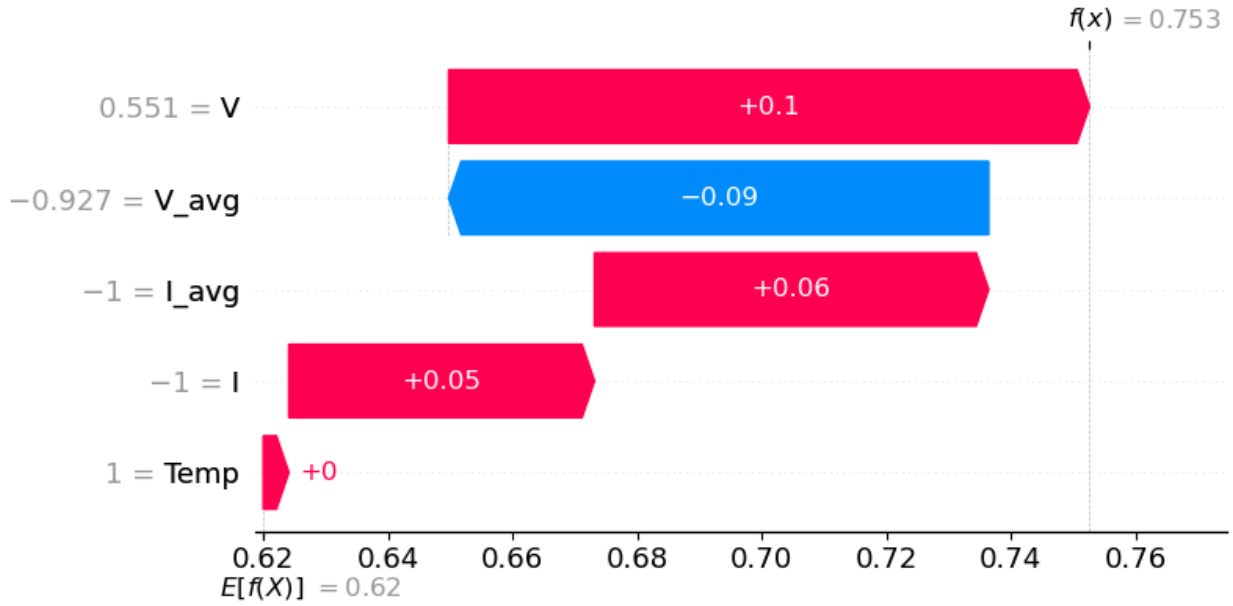
- `shap.sample(X_test, 100)` $\rightarrow$ Samples a ***subset of 100 data points*** from `X_test` to speed up calculations.
- This ***sampled data*** is used by the ***KernelExplainer*** to calculate SHAP values efficiently.

2. KernelExplainer Technique:

- The ***KernelExplainer*** uses a ***kernel-based technique*** to determine the effect of features on the model's predictions.
- It evaluates ***all possible feature combinations*** in complete subsets to determine how much each feature contributes.
- In practice: SHAP value = Effect of adding/removing a feature from the model output

3. Full Subset Calculation:

- `num_full_subsets` $= 2 \rightarrow$ Specifies that ***2 complete subsets*** were used for SHAP value calculation.
- This means the SHAP calculation considered ***two complete feature combinations*** when estimating the feature contributions.



SHAP Waterfall Plot Interpretation

In the waterfall plot above:

- The ***x-axis*** represents the values of the target (dependent) variable, which is the ***State of Charge (SOC)***.
- x is the ***chosen observation***,
- $f(x)$ is the ***predicted value*** of the model for the given input x,
- $E[f(x)]$ is the ***expected value*** of the target variable, i.e., the ***mean of all predictions***: $\text{mean}\big(\text{model.predict}(X)\big)$

SHAP Values:

- Each bar length represents the ***SHAP value*** of a feature for this observation.
- The SHAP values indicate how much each feature contributes to pushing the model's prediction higher or lower than the expected value.

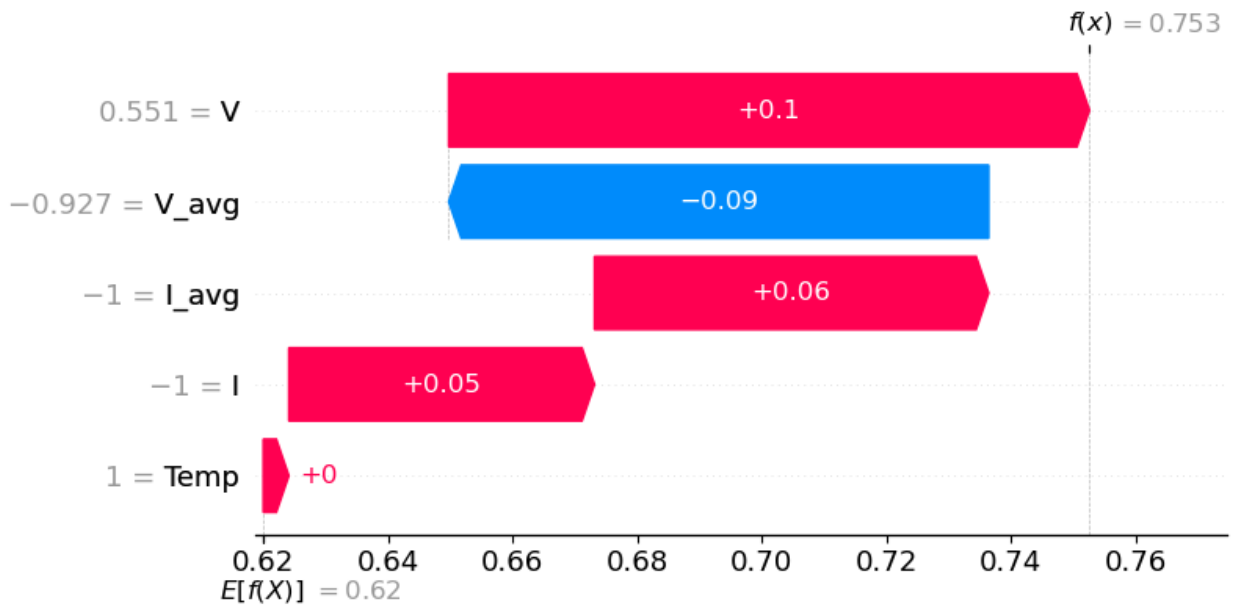
- $I_{avg} \rightarrow +0.09$
- $V_{avg} \rightarrow -0.08$
- $V \rightarrow +0.05$
- $I \rightarrow -0.03$
- $Temp \rightarrow +0.01$

Key Insights:

- The ***sum of all SHAP values*** equals $E[f(x)] - f(x)$.
- The ***absolute SHAP values*** reveal the magnitude of each feature's influence:
  - I_{avg} contributed the most,
  - V_{avg} the second most,
  - V the third,
  - I the fourth,
  - $Temp$ had the ***lowest contribution*** to the prediction.

Visualization:

The following plots displays the feature importance, indicating how much each feature contributes to SOC estimation.



SHAP Values Interpretation

- The ***SHAP values*** shown are *valid for this observation only.*
- With other data points, the SHAP values will ***change***.
- To understand the ***global importance*** or contribution of features across the entire dataset, you can use:
  - A ***bee swarm plot***
  - `summary_plot()` from the SHAP library

Example Insight:

- High values of I_{avg} have a ***high negative contribution*** to the prediction,
- Low values of I_{avg} have a ***high positive contribution***.
- The ***Temp*** feature has ***minimal contribution***, regardless of whether its values are high or low.

Global Feature Importance:

- All variables are displayed in order of ***global feature importance***:
  - The ***first feature*** is the most important,
  - The ***last feature*** is the least important.