PAPER • OPEN ACCESS

A comprehensive machine learning-based investigation for the index-value prediction of 2G HTS coated conductor tapes

To cite this article: Shahin Alipour Bonab *et al* 2024 *Mach. Learn.: Sci. Technol.* **5** 025040

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
16 November 2023

REVISED
5 April 2024

ACCEPTED FOR PUBLICATION
30 April 2024

PUBLISHED
13 May 2024

PAPER

A comprehensive machine learning-based investigation for the index-value prediction of 2G HTS coated conductor tapes

Shahin Alipour Bonab[1], Giacomo Russo[2], Antonio Morandi[2] and Mohammad Yazdani-Asrami[1,*]

[1] Propulsion, Electrification & Superconductivity group, James Watt School of Engineering, University of Glasgow, Glasgow G12 8QQ, United Kingdom
[2] Department of Electrical, Electronic, and Information Engineering, University of Bologna, Viale del Risorgimento 2, 40136 Bologna, Italy
[*] Author to whom any correspondence should be addressed.

E-mail: mohammad.yazdani-asrami@glasgow.ac.uk

Keywords: artificial intelligence, neural network, *n*-value, superconductors, modelling

Abstract

Index-value, or so-called *n*-value prediction is of paramount importance for understanding the superconductors' behaviour specially when modeling of superconductors is needed. This parameter is dependent on several physical quantities including temperature, the magnetic field's density and orientation, and affects the behaviour of high-temperature superconducting devices made out of coated conductors in terms of losses and quench propagation. In this paper, a comprehensive analysis of many machine learning (ML) methods for estimating the *n*-value has been carried out. The results demonstrated that cascade forward neural network (CFNN) excels in this scope. Despite needing considerably higher training time when compared to the other attempted models, it performs at the highest accuracy, with 0.48 root mean squared error (RMSE) and 99.72% Pearson coefficient for goodness of fit (*R*-squared). In contrast, the rigid regression method had the worst predictions with 4.92 RMSE and 37.29% *R*-squared. Also, random forest, boosting methods, and simple feed forward neural network can be considered as a middle accuracy model with faster training time than CFNN. The findings of this study not only advance modeling of superconductors but also pave the way for applications and further research on ML plug-and-play codes for superconducting studies including modeling of superconducting devices.

1. Introduction

Superconductors are a type of materials that can carry high currents with no or much reduced (depending on the DC or AC operating conditions) ohmic losses compared to conventional conductors. They also exhibit distinctive magnetic properties like flux trapping and levitation capability. Due to these exceptional properties, superconducting technology is an enabler for a variety of applications in the fields of energy generation and transmission, transportation, biomedicine, among others. Today, it is acknowledged that the macroscopic behavior of superconductors (such as the magnetic properties and the losses) can be reproduced by numerical models that employ a nonlinear constitutive law relating the local electric field E to the current density J [1–3] that is the well-known power law, written in equation (1),

$$E = E_0 \left| \frac{J}{J_C} \right|^n \qquad (1)$$

where E_0 is a conventional parameter used as a criterion for the definition of the conventional critical current density J_c (a value of 1 μVcm^{-1} is assumed), and the n (n-value, also known as index-value and power exponent) is fundamental for reproducing the nonlinearity of superconductors' behavior. The n-value is not merely a convenient mathematical approximation, the power law can be regarded as the macroscopic expression of the flux creep phenomena occurring in type II superconductors [4]. In particular, at a given

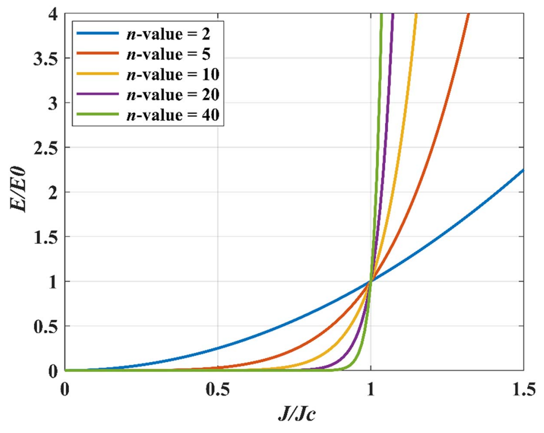


Figure 1. Scaled E–J relations for different n-values.

temperature and applied magnetic field, the n-value is proportional to the activation energy for depinning of the flux vortices; high n-values correspond to high depinning energy and therefore to weak creep. As a consequence, the n-value closely relates to the vortex creep and the micro-structure of the film [5]. In fact, the n-value is a dimensionless number that offers insights into the non-linear electric field response of superconductors under an applied current. This parameter provides insights into the superconductor material quality and characteristics, its micro-structure, and critical current surfaces. For instance, the n-value depends on grain boundaries, defects, inhomogeneities, vortex creep/dynamics under Lorentz forces, and the distribution of pinning centers. Higher n-values indicate sharp transitions from the superconducting to the normal state, which is desirable for applications requiring high current densities [5]. Usually, both the critical current J_c and the n-value are deduced from the fitting of the experimentally measured E–J curve. The impact of the n-value on the E–J curve shape is shown in figure 1. Although for most superconductors the n-value ranges from 20 to 40 (especially high temperature superconductors), curves for n-values of 2, 5, and 10 are also included in figure 1 to exacerbate its impact on the steepness of the E-J transition.

The n-value importance for practical aspects of superconductors' behavior was recently investigated, showing its role in stability by affecting the quench velocity propagation and temperature [6–9]. Moreover, lower n-values indicate a slow transition to the normal state, carrying significant implications for practical applications. For instance, high-temperature superconducting (HTS) materials typically have n-values around 20, whereas their low-temperature superconducting (LTS) counterparts boast values closer to 100. While HTS offers enhanced magnet protection, operating it near the critical current threshold—around 90%—can lead to resistive voltages, resulting in energy losses and potentially harmful heating effects [10, 11].

In terms of industrial applications of superconducting materials and the importance of n-value, the use of superconducting magnets is promising in the industries where strong, energy efficient, and reliable magnetic fields are required. As an example, the designers of fusion devices are implementing both LTS and HTS based magnets in their research and development projects to improve the design of these devices and provide strong magnetic fields around 20 T. In line with definition of n-value which was mentioned earlier, a higher n-value indicates a sharper transition from the superconducting state to the normal state. This means that as the current or magnetic field strength approaches the critical value, the superconducting material undergoes a rapid transition to the normal state. This can change the magnet protection implications in terms of internal fault or quench occurrences. Depending on the application of these materials as of magnets how rapid the magnet is needed to transit to normal state, the n-value should critically take into account by the designers. To extract the n-value, the E–J curve is measured in characterization experiments at different magnetic field conditions (magnitude and angle) and temperature, and then, value of n allowing the best fitting of the measured data by means of equation (1) is found. As an example, Rimikis *et al* briefly cover voltage-current characteristics of composite superconductors, essential for high-field magnet development and n-values were investigated via four-point measurement [12]. However, in order to limit the number of experiments needed for the precise characterization of the material an interpolation method for n-value over a wide range of operating conditions in terms of magnetic field and angle is needed. Oh *et al* built upon the Kramer model and introduced empirical equations for the critical current and the n-value, using the principle of proportionality. Furthermore, they explored the relationship between these two terms in details in [13, 14]. An alternative approach—based on the use of machine learning (ML) techniques—to the fitting

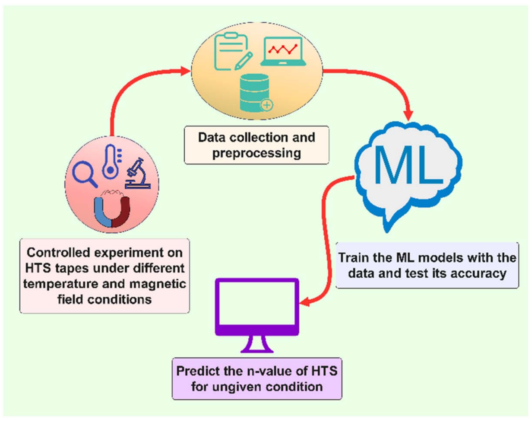

Data collection and preprocessing
ML
Controlled experiment on HTS tapes under different temperature and magnetic field conditions
Train the ML models with the data and test its accuracy
Predict the n-value of HTS for ungiven condition

Figure 2. Overview of the ML modeling for n-value estimation.

of the experimental data has been proposed in very recent years which would drastically reduce computational cost and other technical resources. The summary of the ML-based prediction process has been illustrated in figure 2.

In the literature, researchers have rarely used intelligent techniques for modeling or the prediction of parameters of superconductors. Russo *et al* developed artificial intelligence-based models for reconstructing the critical current and index-value surfaces of HTS tapes. Their study demonstrated that artificial neural network (ANN) has better accuracy than extra gradient boosting (XGBoost) and Kernel Ridge Regression (KRR) models for critical current and n-value estimation [15]. Zhu *et al* address the prediction of critical current and n-value in second-generation HTS conductors using a backpropagation neural network. The network efficiently estimates $J_c(B, \theta, T)$ and $n(B, \theta, T)$, benefiting from shared hidden layers [16]. The approach accommodates variations between manufacturers and batches, enhancing adaptability. The method indicates a high accuracy accompanied by a fast prediction.

Although some efforts related to the use of ML techniques for the estimation of different parameters of superconductors have been done in literature, there is a need for a comprehensive study on n-value prediction of the superconductors using ML. In other words, there is a margin for improvement in the n-value prediction accuracy, which is the founding reason for carrying out this study in the first place. In this paper, a comprehensive investigation of different ML methods for n-value prediction has been carried out to assess which method results in higher accuracy and faster response.

The main reason behind predicting the n-value for a sample using ML model is that the experiments only cover a limited amount of test conditions like temperature and magnetic field. As these tests are done under severely low temperatures, it is so expensive to do the tests for a desired condition. Therefore, the ML methods are implemented to make models based on the experimental data, which can predict the n-value on the conditions that no experiments have been done (unseen conditions by model). In this approach, only conducting a preliminary set of tests on any superconducting sample is enough for generating the data that is necessary for developing the ML model. Once the ML models are trained with those limited data, they can predict n-value very fast without having access to the whole dataset or look-up tables. In other words, the ML model does not need a training process after being created, and it can be integrated into any modeling or experimental systems. This is what we call it, plug-and-play function. Subsequently, designers and researchers can depend on the predictive capabilities of ML model for assessing the desired physical conditions, which can save money and time for many companies that are using that type of superconductor.

Also, there are some pros of using AI modeling instead of traditional mathematical fitting. As the pattern of n-value is extensively non-linear, the mathematical fitting methods that mostly are based on linear (or polynomial regressions) are not able to capture this level of non-linearity. This is while the ML models by using complex and innovative approaches can learn these complex patterns and predict the unseen datapoints through extrapolation with significantly high accuracy. Another advantage of ML models is their ability to handle vast amounts of data and input variables. In fact, more datapoints often lead to more robust models which estimate with more accuracy. This is while in mathematical fittings, the increased complexity that comes with larger datasets may result in lower accuracy as they may be limited by assumptions about the data distribution. Another major benefit of the ML model is their ability to get retrained. This means that the model can be trained with more data of even other superconductors to predict n-value of vast numbers of superconductors (not only one sample.)

In the following, first in section 2, the data collection and the process for making it suitable for ML models is described. In section 3, all the models that have been used in this study are briefly introduced and explained. In section 4, a detailed explanation hyperparameter sensitivity analysis for the superior model and the summary of the best identified values for other models are presented. Next, the results of tuned models are compared in section 5. Then, in section 6, the detailed results of the superior model are presented for different temperatures, field density, and field angle. Finally, a conclusion based on the results of the third section is presented in section 7.

2. Data collection and preprocessing

For training and testing the aforementioned ML models, the open-access database of 'HTS critical current data' provided by the Robinson Research Institute (Victoria University of Wellington, New Zealand) [17–19] was used that can be found using this link (https://htsdb.wimbush.eu/dataset/3759315). This dataset includes multiple data points for the n-value of various commercial HTS tapes, recorded under different combinations of temperature, magnetic field amplitude, and orientation (i.e. the angle between the c-axis and the field vector). The HTS tape dataset that was used in this study is the one of SuperOx GdBCO 2G HTS for temperatures between 15 K and 85 K and the magnetic field between 0.01 T and 7 T. The n-value also varies between 1.01 and 42.69. The reason for using the data related to one HTS sample exclusively is that it is reasonable to assume that the n-value properties, as well as all other properties of any HTS tape, are sample specific. However, it is specified that the ML models developed in this study can be updated with the experimental dataset corresponding to another HTS tape if its n-valued prediction depending on the operating conditions needs to be addressed. The effective application of such an approach that produces ML models specific for unique HTS tapes was recently demonstrated with regard to FEM models' validation [20]. The raw dataset needs some techniques for preprocessing. First of all, the data points that have missing information caused by sensor malfunction during experiments have to be removed. Then, the conditions (either temperature or magnetic field conditions) that have a lot of missing data points should be removed as they suffer from the lack of sufficient training data points. So, a dataset including the temperature, magnetic field magnitude, and magnetic field orientation as the effective input parameters and the n-value as the target value with 14 022 observations has been made for being used to develop the ML models. For carrying out this step, the data needs to be split into distinct datasets for training, validating, and testing processes. The splitting process ensures that the used ML model generalizes well to unseen data and provides reliable performance metrics. This process can be done by randomly choosing the data points from the input dataset. For this study, according to the common practice among data scientists, 20% of the input dataset was randomly considered for the testing and validation process and 80% for training of the model. These rates have been chosen as the increase in the training rate increases the chance of overfitting the model. Also, lower training rates will result in lower accuracy of the model as it has not been well-trained with sufficient data points to learn the way that the n-value is changing.

To assess the accuracy and reliability of the proposed ML models, their performance should be evaluated by numeric metrics. These metrics/indexes provide a quantitative measure of a model's performance, which enables scientists to judge how well their model is functioning. Also, the effectiveness of models can be compared, so that better solutions can be chosen through them. In this paper, some indexes have been considered for the performance of models based on their robustness to clarify the model's overall quality and being well-known among ML researchers [21–24]:

$$\mathrm{MAE} = \sum_{k=1}^{n_s} \frac{|p_k - y_k|}{n_s} \tag{2}$$

$$\mathrm{MARE} = \sum_{k=1}^{n_s} \frac{\left|\frac{(p_k - y_k)}{y_k}\right|}{n_s} \tag{3}$$

$$\mathrm{RMSE} = \sqrt{\sum_{k=1}^{n_s} \frac{(p_k - y_k)^2}{n_s}} \tag{4}$$

$$R^2 = \frac{\sum_{k=1}^{n_s} (p_k - \bar{p})^2 - \sum_{k=1}^{n_s} (p_k - y_k)^2}{\sum_{k=1}^{n_s} (p_k - \bar{p})^2} \tag{5}$$

In these equations, y is the predicted value, $\bar{y}$ is the mean value of y, n_s is the number of samples of the training dataset, p_k are the actual values, and $\overline{p_k}$ is the mean value of p_k. Equation (2) represents the mean absolute error (MAE) which quantifies the average error of the model to predict each data point. Therefore, it has the same unit as the predicted parameter.

Equation (3) reports the mean absolute relative error (MARE) between the predicted value and the actual (real experimental) value of each data point in percent. This is a very useful tool to evaluate the accuracy of a model as it does not have a unit.

Equation (4) represents the root mean squared error (RMSE) which has been widely adopted by data scientists in recent decades. This index has been developed based on the logic to get the average value of squared errors instead of the error value itself. This approach ensures that data points with substantial errors exert a greater influence on the overall accuracy of metrics and consequently, on the model.

Equation (5) shows the R-squared, also known as the coefficient of determination and the goodness of fit, which is a statistical metric employed in regression analysis for evaluating the degree of fit between the predicted quantities by the regression model and the actual observed data. It operates on a scale from 0 to 1, where higher values—closer to 1—signify a stronger fit. A higher R-squared value suggests a better fit of the model to the data, indicating that a larger proportion of the variability in the dependent variable is accounted for by the independent variables.

3. Modelling of different ML techniques

In this section, all the models and the indices that are used in this study are explained in detail. It should be noted that the cascade-forward and feed-forward neural networks have been developed in the MATLAB software package as the working frame for programming of the developed neural networks by our groups; however, for the remaining ML models, although the programming of all of the models have been carried out and administrated by our groups, the functions in the existing and open access libraries of the Python 3 (like SciKitLearn, XGBoost, etc) have been used.

3.1. Multilayer perceptron neural network (MLPNN)

Neural networks are a class of models that have been inspired by the structure and function of the human brain's neural system. The MLPNN, also known as the feed-forward neural network, is one of the most used neural network methods by researchers due to its flexibility to both low- and high-complexity datasets resulting in very high accuracy [25]. A MLPNN contains at least three layers, which are referred as input, hidden, and output layers and within each layer, there are some computational units, which are known as neurons. Each neuron consists of one weight factor for weighting the incoming data from up-hand layers to lower the error and one bias factor to trim the result independent to the input data. In MLPNN, which follows a forward approach (without feedback of next layers), the network updates itself at consecutive cycles, which are known as epochs [26]. During each epoch, the data is provided from the input dataset and goes through the network layers while the model tries to optimize the weight and bias factors of its neurons using that data to reach a better learning of the pattern of the data [27]. Once the epoch finished, the network tests itself by evaluating the error between the predicted and actual value and according to the amount of this error, it changes the weight and bias factors through a feedback loop to reduce the error for next epoch. This backward feeding of the network is known as backpropagation process. This process proceeds till the network reaches to a saturate condition of error or stop at the pre-set error [28]. The fundamental mathematical equation of this methodology is as follows:

$$y_p = f^0\left(\sum_{j=1}^{n}\omega_i^0 x^i f_j^H\left(\sum_{i=1}^{n}\omega_{ji}^H x_i\right)\right) \tag{6}$$

where f^0 and f_j^H are representatives of the output layer and the hidden layer activation function, respectively. By adding a bias factor to both the input and hidden layers, equation (2) turns into:

$$y_p = f^0\left(\omega^b + \sum_{j=1}^{n}\omega_i^0 x^i f_j^H\left(\omega_i^0 + \sum_{i=1}^{n}\omega_{ji}^H x_i\right)\right) \tag{7}$$

where ω_j^H and ω^b demonstrate the respective weight from bias to the hidden and output layers [23].

The number of the hidden layers, setup of the neurons, training function and activation functions needs to be optimized to reach the highest accuracy. Since there are a great number of possible combinations, it is

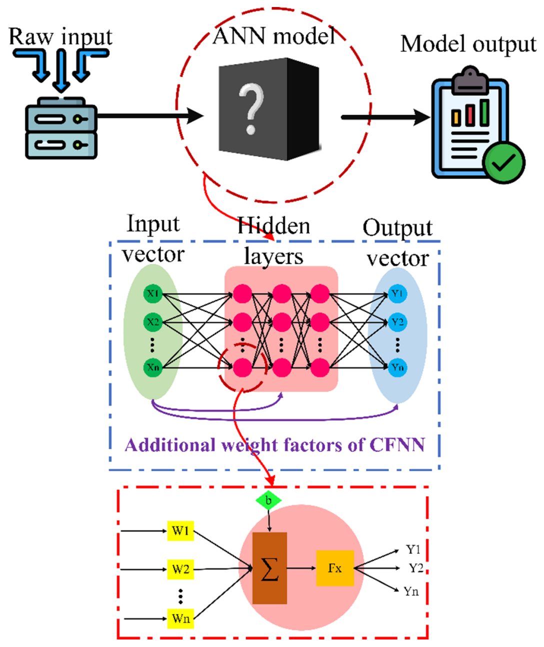


Figure 3. Schematic of FFNN and CFNN.

not possible to test all of them and by using a grid search approach for the hyperparameters in a reasonable and common range in the literature, it has been tried to increase the overall performance and quality of the model.

3.2. Cascade forward neural network (CFNN)

The CFNN is basically like MLPNN with the main difference being that the number of weight factors of each neuron in the lower hand layers (the layers after input layer including all hidden layers and output layer) increases in a cascade form. Therefore, unlike MLPNN, all the previous layers' outputs are involved in the parameters' adjustment process [29, 30]. This is while in a MLPNN model, each layer receives information only from the neurons in the preceding layer. For better understanding, a simple three-layer CFNN and MLPNN models are taken as examples which contains an input, a hidden layer, and an output layer. In both of these models, the neurons are interconnected and feed their next layer. So, the input layer only feeds the hidden layer. This is while, in CFNN, the neurons of input layer directly involve in the results of neurons of output layer by feeding the information to them. This more complex architecture results in more training time. That said, the training process of the network becomes deeper than MLPNN which will result in higher accuracy in some cases, especially for high non-linear datasets [31]. Helping for understanding of this procedure, figure 3 demonstrates how both CFNN and FFNN methods are working.

With all the details provided, it is obvious that the fundamental equations of CFNN are very likely to MLPNN and only needs some adjustments to add cascade weights. The related equation to this method can be written as follows:

$$y_p = \sum_{i=1}^{n} f^i \omega_i^0 x^i + f^0 \left(\sum_{j=1}^{n} \omega_i^0 x^i f_j^H \left(\sum_{i=1}^{n} \omega_{jh}^H x_i \right) \right) \tag{8}$$

where, f^i represents the activation function of the output layer, and f_j^H signifies the activation function of the hidden layer. When introducing bias to both the input layer and the hidden layers, equation (4) will undergo some modifications:

$$y_p = \sum_{i=1}^{n} f^i \omega_i^0 x^i + f^0 \left(\omega^b + \sum_{j=1}^{n} \omega_i^0 x^i f_j^H \left(\omega_i^0 + \sum_{i=1}^{n} \omega_{jh}^H x_i \right) \right) \tag{9}$$

where, ω_j^H and ω^b represent the weights associated with the connection from bias to the hidden layer and from bias to the output layer, respectively [23].

For CFNN, the detailed range of hyperparameters and the process is provided in the hyperparameter optimization section.

3.3. Decision tree regression (DTR)

Decision Tree (DT) is a type of supervised learning model that can be used for both classification and regression tasks. This approach is based on a binary tree structure where nodes are split to form the decision tree [32]. The algorithm of the decision tree involves dividing the dataset into smaller segments or classes and presenting the outcome in a leaf node [33]. Essentially, the decision tree processes the dataset to create a tree-shaped structure (branches) that facilitates prediction. This is why it is sometimes referred to as tree structure regression [34]. DT comprises three distinct types of nodes: root nodes, interior nodes, and leaf nodes. The root node, being the first node, branches into more nodes known as interior nodes. These interior nodes represent the model's data characteristics and decision criteria, while the leaf nodes indicate the final prediction from the decision-making process [35]. Therefore, the hyperparameters that should be optimized for this method are mostly for controlling the number and the conditions that each node or leaf is created. For example, minimum sample of a leaf is a hyperparameter that defines how many samples are needed at least to reach a leaf node (where the model does not go deeper).

3.4. Random forest regression (RFR)

Random forest is an ensemble learning technique that is capable of both classification and regression processes. The method of learning in a random forest is built around the idea of combining multiple decision trees that divide the input data using specific parameters in a tree-like arrangement. Every tree is formed using a bootstrapped subset of the data, and at each node, the best subset and predictors are picked randomly for splitting [34]. The final prediction is made by tallying the votes from the decision trees, and the output is determined accordingly. One of the important privileges of this method is that the overfitting is unlikely [35, 36]. Same for decision tree, the hyperparameters of this method are for controlling of the subtrees such as maximum depth, etc. A unique and important hyperparameter for this method is the number of trees that the random forest creates to reach a decision. More trees generally mean more accuracy of prediction, but after a certain point, the performance will not get affected by changing of this parameter.

3.5. Gradient boosting regression (GBR)

The GBR algorithm involves a sequence of regressions that are trained step by step, continually refining the prediction of target values to rectify errors. Through iterative optimization of the regression's output, GBR is employed to solve the minimization problems. This leads to a gradual reduction of errors from prior regressions [37]. The gradient boosting tree (GBR) F_n can be expressed as the summation of n number of regressions:

$$F_n\left(x_t\right) = \sum_{i=1}^{n} f_i\left(x_t\right) \tag{10}$$

with each f_i being a decision tree. This optimization is tackled using the steepest descent technique [38]. In this research, the number of estimators varied between 10^3 and 10^6 to ensure the stability of results. As this algorithm usually uses decision trees, the hyperparameters are for controlling the size of the trees. Additionally, learning rate as a very important adjusts the speed of learning. Generally, lower amounts of the learning rate result in higher accuracies but it can also increase the chance of overfitting. It is worth noting that the other methods that are using the same approach of GBR have almost same hyperparameters.

3.6. Light gradient boosting regression (LightGBM)

LightGBM is a widely used boosting method that has fixed the drawback of GBR when handling large datasets. The process incorporated gradient-based one-side sampling (GOSS) and exclusive feature bundling (EFB) techniques to aid in training without compromising model accuracy and performance, as outlined by [39]. The GOSS algorithm selectively excludes a significant portion of data instances with small gradients, focusing solely on those that contribute significantly to information gain. Specifically, it starts by assigning the top $a \times 100\%$ instances with the largest gradients to a subset A. Then, for the remaining set A^c, which constitutes $(1-a) \times 100\%$ instances with smaller gradients, a new subset B is formed through random sampling with a size of $b \times |A^c|$. Finally, the variance gain $\tilde{V}_j(d)$ is employed to split data instances which can be expressed as follows [39, 40]:

$$\tilde{V}_j\left(d\right) = \frac{1}{n}\left(\frac{\left(\sum_{x_i\in A_l} g_i + \frac{1-a}{b}\sum_{x_i\in B_l} g_i\right)^2}{n_l^j\left(d\right)} + \frac{\left(\sum_{x_i\in A_r} g_i + \frac{1-a}{b}\sum_{x_i\in B_r} g_i\right)^2}{n_r^j\left(d\right)}\right). \tag{11}$$

In equation (11), $n_l^j(d)$ and $n_r^j(d)$ represent the left and right nodes, A_l and A_r represent subsets of A, and B_i and B_r are the subsets of B. EFB, or EFB, serves to decrease the feature count by grouping features that do not overlap within a sparse feature space. This results in enhanced model performance and computational efficiency. Moreover, a leafwise tree growth approach is employed to lift the model performance and prevent overfitting [40]. To more help for preventing overfitting condition in the model, a hyperparameter which is called as minimum child samples, controls the least required number of datapoints for each of leaf nodes. Another key hyperparameter is the maximum number of leaf nodes in each tree. Generally, more leaf nodes mean that the model has deeper decision trees that makes it more robust. That said, it can potentially increase the risk of overfitting. Also, if it is set to low value, the model will not demonstrate good performance.

3.7. Hist GBR (HGBR)

HGBR is an ML method that is used for regression problems and has been developed based on GBR. HGBR is designed to expedite training on large datasets with lots of data points/observations. It employs histogram-based strategies to speed up the creation of decision trees [41, 42]. Instead of dealing with individual data points, HGBR operates with histograms that represent the values of features resulting in enhanced computational efficiency [38, 43]. Same as for GBR, the number of estimators changed between 10^3 and 10^6. Since the model learns the pattern in an iterative manner, the number of maximum iterations needs to be set to a reasonable value to let the model learn the pattern well. Rest of the hyperparameters are same of other boosting methods.

3.8. Extreme gradient boosting regression (XGBoost)

XGBoost is an ML algorithm that can be used for regression problems. In terms of its architecture, it evaluates the data by assessing the results of multiple subtrees to make a final evaluation. What makes the XGBoost a n important ML technique compared with other traditional methods is its scalability, speed, and robustness [44].

XGBoost incorporates a distinctive regularization term into its objective function, a crucial component for mitigating overfitting and enhancing the model's capacity to generalize. Additionally, it employs a parallelized and optimized gradient boosting framework, a key reason for its remarkable speed compared to conventional gradient boosting techniques. This acceleration is made possible through techniques such as tree pruning and optimization, allowing XGBoost to efficiently handle extensive datasets [45].

The mathematical ensemble model of this method is as follows [46]:

$$\hat{y}_i = \sum_{k=1}^{K} f_k\left(x_i\right), f_k \in R \tag{12}$$

where $\hat{y}_i$ is the predicted n-value, k is the number of subtrees, f is the correlation mapping relationship between the structure and weights of a tree in the subtree set, x_i is the input vector, and R represents the subtree set.

The objective function of XGBoost is composed of two main components: the first part quantifies the error between the predicted values and the actual true (real) values of the model, while the second part is a regularization term that manages the complexity of the model which can be mathematically represented as follows [45]:

$$Y = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i\right) + \sum_{k=1}^{K} \Omega\left(f_k\right). \tag{13}$$

In this equation, the term $\sum_{i=1}^{n} l\left(y_i, \hat{y}_i\right)$ quantifies the summation of differences (i.e. errors) between the predicted values of the model $\left(\hat{y}_i\right)$ and the true (real) values $\left(y_i\right)$. The second term, $\sum_{k=1}^{K} \Omega\left(f_k\right)$ quantifies the complexity of the tree model. This complexity term, also known as the regularization term, is vital for preventing the model from overfitting and helps control the overall complexity of the model. This balance between error minimization and model complexity regulation is a key feature of XGBoost, contributing to its effectiveness in predictive modeling [15]. To facilitate a clearer comprehension of the XGBoost method's process, figure 4 has been provided. As this method is a member of the gradient boosting method family, its hyperparameters are like other described methods of this family like GBR, LightGBM, etc.

3.9. CatBoost regressor

One of the accurate ML methods, which is a member of boosting methods, is CatBoost regression. Catboost was developed based on [47] and can be used as both a classification and regression method. It can handle various datasets, either small or large ones [48].

When it comes to comparing CatBoost and other GBR methods, the major distinction is that CatBoost employs a random permutation approach. It computes and assigns an average classification value to samples with similar category values and substitutes them with the specified permutation. As it has been exemplified in [49] if a given permutation is $\left[\sigma_1, \ldots, \sigma_n\right]_n^T$, CatBoost replaces that with equation (14) [50, 51]:

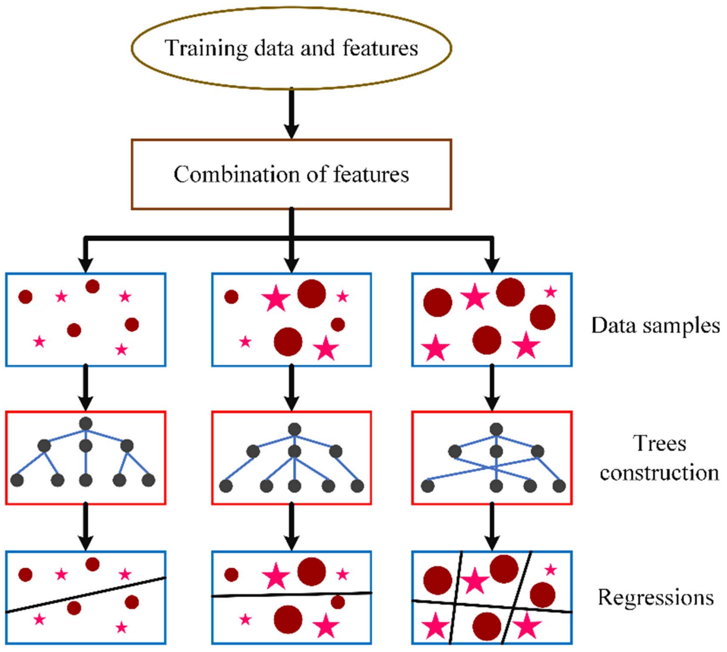


Figure 4. A schematic of XGBoost and CatBoost regression methods.

$$\hat{X}_k^i = \frac{\sum_{j=1}^{n} I\left[X_j^i = X_k^i\right].Y_{\sigma_j} + \beta P}{\sum_{j=1}^{n} I\left[X_j^i = X_k^i\right]\beta} \tag{14}$$

where I is the indicator function, X_k^i is the ith subtype feature of the kth training sample, and P and β are the foregoing value and weight of the foregoing parameter, respectively. Also, figure 4 demonstrates how this technique works.

Regarding the hyperparameters of this method, it generally uses the same parameters as other boosting methods. However, an important parameter that does not exist for other boosting methods is *L2* leaf regularization factor. This parameter applies *L2* regularization to leaf scores to penalize large coefficients and prevent more complex trees. Through this process, it prevents the overfitting condition of the model. More information regarding *L1* and *L2* regularization will be provided in section 3.10.

3.10. Linear, Huber, Lasso, and Elastic Net regressions (ENR)

Linear regression is a statistical ML technique that is employed for analyzing and predicting numerical variables through correlations. It is focused on determining how effectively one variable can be used to predict another variable in a linear manner [52]. For this purpose, several predictors are utilized to forecast a single dependent variable trying to find a simple linear equation that has the lowest error in predicting the target value. Equation (11) represents the formula for the linear regression model,

$$y = b_0 + \Sigma_{i=1}^{p} b_i x_i + \in \tag{15}$$

where b_0 is the intercept, p is the number of independent variables, b_i are the coefficients of the independent variables, x_i is the independent variable, and $\in$ is the random regression error [53].

There are other variations of linear regression including Huber regression (HR), Lasso regression, and ENR, each incorporating unique regularization methods.

HR helps with the problem of outliers in regular linear regression by using a mix of squared and absolute error in its calculations [54].

Lasso regression (LR), on the other hand, encourages simpler models by adding up the absolute values of coefficients in the cost function. This also means it can automatically decide which features are more important [52]. The lasso problem can be defined as:

$$\text{minimize } \beta, \beta_0 \left\{ \frac{1}{2N_j} \sum_{i=1}^{N_j} \left(y_i - \Sigma_{k=1}^{p} x_{ik}\beta_k\right)^2 + \lambda \Sigma_{k=1}^{p} |\beta_k| \right\} \tag{16}$$

where $\lambda\Sigma_{k=1}^{p}|\beta_k|$ is considered an *L1* regularization term. Should we consider the squared value of β_k in the formula instead of $|\beta_k|$, the term is known as the *L2* regularization term [55].

ENR, though, balances things out by using both *L1* (Lasso) and *L2* (Ridge) regularizations. *L1* makes coefficients sparse, while *L2* prevents them from getting too large. This makes Elastic Net better at handling situations where features are related, unlike just using Lasso. So, these methods make linear regression better in different ways: Huber deals with strange values, Lasso picks features, and Elastic Net gets the best of both *L1* and *L2*, especially when features are connected [56].

Ridge regression introduces *L2* regularization as a penalty term in the objective function of multiple linear regression. This transforms the problem of finding the optimal coefficients into a constrained optimization problem. By incorporating this *L2* regularization, ridge regression encourages simpler coefficient values, which, in turn, enhances the algorithm's ability to generalize to new data. The objective function for ridge regression is represented as follows [57, 58]:

$$J(\theta) = \frac{1}{2}\sum_{j=1}^{m}\left(\theta X_j - y_j\right)^2 + \lambda||\theta||^2 \tag{17}$$

where λ is the ridge parameter, y_j is the actual value, and $\theta^T X_j$ is the predicted value.

There are few hyperparameters for this family of algorithms. Most of these parameters are for directly controlling the way model trains itself to reach the final regression. Also, some parameters such as alpha, *L1*, and *L2* ration are for controlling of the regularization factor of the model.

3.11. KRR

KRR is a technique used for handling complex data that cannot be effectively addressed through a simple linear relationship. The primary difference between Ridge regression and KRR lies in the type of data they are designed for. Ridge regression is primarily used for linear regression tasks, where the goal is to model relationships between input features and a target variable using linear combinations. It uses the kernel trick to transform the dataset into a higher-dimensional space, where it conducts a form of ridge regression [59]. What sets ridge regression apart from ordinary linear regression is its use of *L2* regularization, which involves adding a regularization term consisting of the squares of the model parameters. The objective function that KRR tries to minimize is as follows [15]:

$$\text{Obj}_{\text{KRR}} = \frac{\sum_{i=1}^{n_s}(y_i - x_i)^2}{n_s} + \sum_{i=1}^{m}\beta_i^2. \tag{18}$$

In this equation, y_i is the predicted value, x_i is actual value, n_s is the number of data/observations, m is the number of parameters, and β is the model parameter.

One of the most important parameters that should be optimized for this model is kernel. There are some common kernels in the literature including radial basis function, polynomial, sigmoid, and linear. Alpha is also another important parameter that controls the regularization factor of the KRR.

3.12. Non-linear regression (NLR)

Most of the datasets are very complex in a way that cannot result in good accuracy for simple regression problems. Therefore, various non-linear methods, like polynomial regression, have been developed.

Polynomial regression involves fitting a polynomial function of a selected degree to the dataset. The degree of the polynomial dictates the model's complexity and its ability to capture the underlying patterns in the data and should be tested to find the optimum number of degrees to result highest accuracy [58].

3.13. Support vector regression (SVR)

SVR is a supervised algorithm used for regression tasks. It works by focusing on a subset of the training data and ignoring data that are close to the model's predictions within a specific threshold ε [60, 61]. To solve regression problems, SVR relies on choosing the right kernel and relevant parameters [62]. A notable strength of SVR is its ability to handle high-dimensional spaces without relying heavily on the input space's dimensionality [63]. SVR employs a linear function, referred to as the SVR equation, to nonlinearly map the input data into a higher-dimensional space. In equation (19), ϕ is the function that transfers data into higher dimensional space and ω and b are weight and bias factors, respectively,

$$g(x) = \omega\phi(x) + b. \tag{19}$$

The main goal is to identify a hyperplane that has the widest possible margin on both sides (figure 5) while keeping the error or deviations of data points from the hyperplane as small as possible [35, 64]. In this

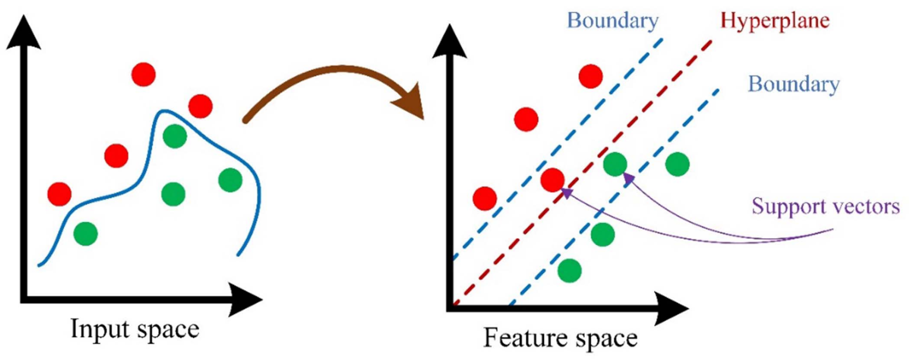


Figure 5. A schematic of SVR technique.

study, the same set of functions that was mentioned for KRR are used for choosing the best function for our model. Another key hyperparameter is the epsilon that sets the amount of error that the model can ignore during the process of finding of hyperplane and the support vectors. C is also the other crucial parameter that highly affect the performance of the model as it changes the regularization term's strength in the model.

3.14. K-nearest neighbor (KNN) regression

KNN is an ML technique that operates by determining the value of a new data point based on its proximity to the K nearest training data points [65]. The KNN regression model operates by determining the distance between a new observation and all the observations present in the training data [66]. The most common distance metric used is the Euclidean distance, which is demonstrated in equation (20) [67]:

$$d\left(x_i, x_j\right) = \sqrt{\sum_{k=1}^{p} \left(x_{ik} - x_{jk}\right)^2}. \tag{20}$$

In this equation, p is the number of input features, x_{ik} is the value of kth input feature for the ith observation. However, in some cases, Minkowski's approach for evaluation of distance results more accuracy for the final model. The related equation for this is as follows:

$$d\left(x_i, x_j\right) = \left(x_i, x_j\right) = \left(\Sigma_{i=1}^{p} \left|x_{ik} - x_{jk}\right|^{n}\right)^{\frac{1}{n}}. \tag{21}$$

After the calculation process of all distances, the algorithm chooses the K neighbors that have the lowest distances. Then, the mean value of those points will be considered as the prediction value [68]. Therefore, the most important hyperparameters of KNN are related to the number of neighbors that the model considers to make an estimation for a new datapoint along the method for distance calculation.

3.15. Adaptive boosting (AdaBoost)

AdaBoost is one of the most common ensemble ML methods that is used by researchers in the field of ML. It is renowned for its capability to enhance the performance of weak learners and its resilience against overfitting. Fundamentally, it trains a sequence of base learners on a given dataset while iteratively adjusting the sample weights. During each iteration, it trains the weak sample on the updated data, and this process repeats until a set number of iterations is completed or a desired level of accuracy is attained, which is shown in figure 6. Then, it combines these base learners to form a powerful model. Also, to elevate the accuracy of the final model, it weighs the samples with higher error [67, 69]. In terms of fundamental equations, considering that the technique is using T sub-classifiers, the final prediction will be as follows [70]:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \tag{22}$$

where in equation (22), $H(x)$ is the final prediction, $h_t(x)$ is the predicted value of sub-classifier t, and α_t is the weight factor that has been considered for that sub-classifier.

This model is also benefited from using loss function to update the weights of individual weak learners that creates. Therefore, an important hyperparameter for this method is this function which is commonly considered to be linear or exponential among the researchers.

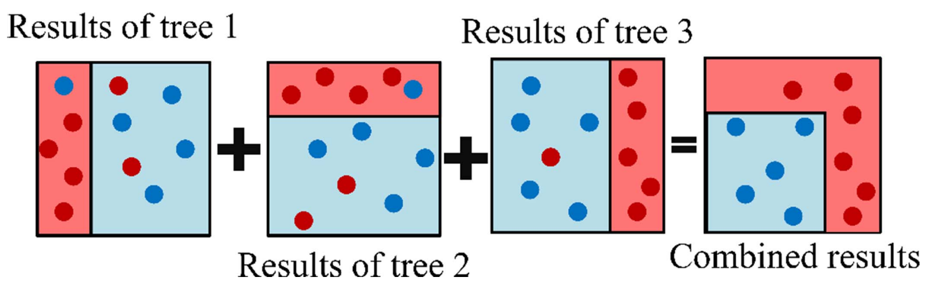


Figure 6. A schematic of prediction process of AdaBoost.

4. Sensitivity analysis of hyperparameters

One of the most important steps of every ML modeling is the sensitivity analysis of the effective controlling parameters, so-called hyperparameters [71]. These parameters are unique for each method; however, some of them are common between the models. The reason that the sensitivity analysis of the hyperparameters is an essential part of ML modeling is that the value of these parameters has direct impact on the accuracy of the model while there is no definite rule to set them in a particular value. Even further, the range that these parameters have influence on the model's performance are highly dependent on the input dataset.

It is worth noting that the training process of an AI model includes considering initial values for the model's parameters, updating these parameters using the training dataset, and testing of the model with testing dataset. The second and third stages should be repeated until the error of model against the training set size reaches to a saturated condition like what has been illustrated in figure 7.

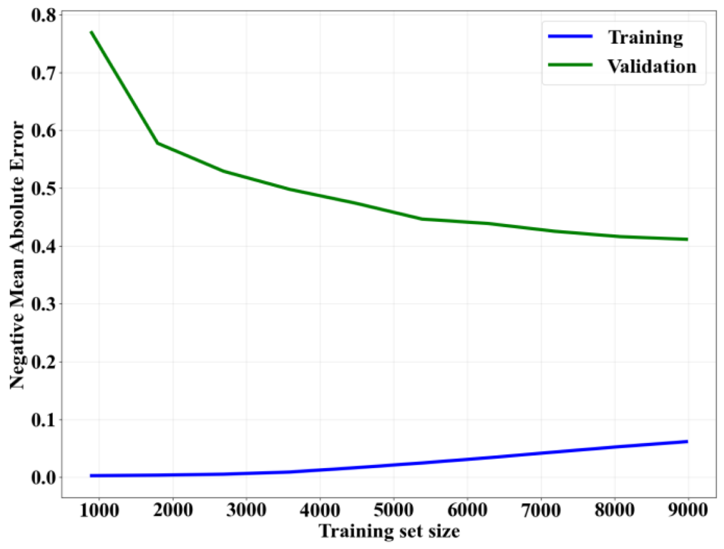
To do the sensitivity analysis on the hyperparameters of each implemented algorithm, a one-at-a-time approach have been used, which is a very common method for sensitivity analysis of different ML hyperparameters. This means that at each step, merely one parameter is variable, and the remaining parameters do not change. First, an initial value for each of the hyperparameters is considered. Next, an effective range for each hyperparameter, in which the performance of the model significantly changes, has to be identified. Then, to further enhance the performance of the model, the value that resulted in the highest accuracy of model is used for the next steps of analysis and remains unchanged. Referring to the details of table 1, the best identified hyperparameters within the testing range that resulted in the highest accuracy are reported. Also, at the front of each of the hyperparameters, the range in which they have been tested is presented. It is worth noting that to make this table, all these parameters should be changed and tested to check their effectiveness on the models' performance, demanding a great deal of attention to reach the best setup. This is because there are no generalized values for these parameters to implement for all cases of studies and they are problem (dataset) dependent. To make it easier to understanding, the process of this analysis has been summarized in figure 8.

Also, as an example, to reach the best setup for CFNN model, which is the most accurate model in this paper, an exhaustive testing process on the five different number of hidden layers, the number of neurons in each layer varying between 5 to 30, six different training functions, and 16 pair of activation functions has been done. The training and testing of each selection of these hyperparameters have been repeated 50 times, ensuring that the results are reproducible, and the model is performed in a stable manner. This resulted in that a total number of 9800 different cases have been investigated merely for CFNN hyperparameter sensitivity analysis process.

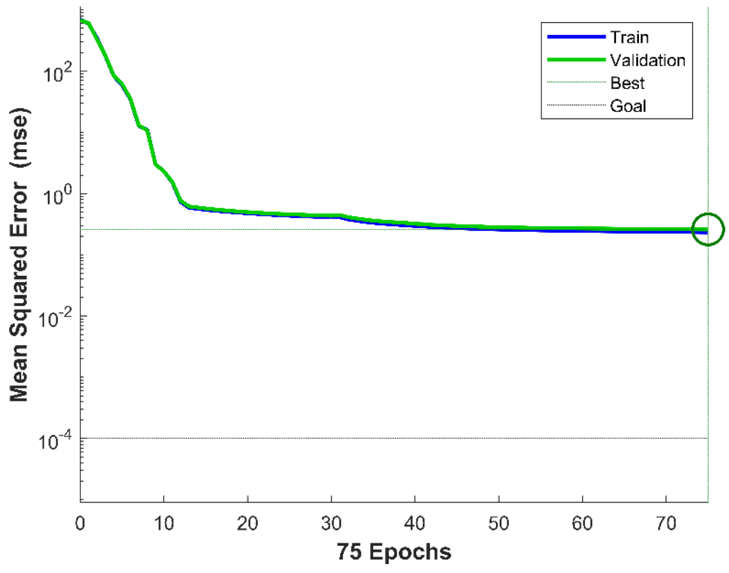
The details of the tuning of hyperparameters for all methods including their testing range and the best value for each of parameters within the testing range are provided in table 1. The best values have been provided to help the researchers have an insight into the values of these parameters and also enable them to reproduce all developed models.

As CFNN model resulted in the highest accuracy amongst all other ML methods and is actually the superior model of this paper, its sensitivity analysis process is explained in the following paragraph. Before starting of the analysis, all of the hypermeters need an initial (default) value, so the analysis can be done step by step.

First of all, the number of hidden layers, which most of the researchers consider to be equal to one due to simplicity, undergoes the sensitivity analysis process. At this stage, the Levenberg–Marquardt (LM) and tansig-purelin have been considered as the training function and the pair of activation function of CFNN model, respectively. Then, the number of hidden layers varied between 1 and 5. The findings of this step of



(a)



(b)

Figure 7. Learning curve of (a) XGBoost and (b) CFNN model.

study demonstrated that the triple hidden-layer model is the best selection as it has the highest accuracy than the others while has lower testing time than the models with 4 and 5 hidden layers. Also, in an entwine approach for neurons analysis, the number of the neurons in each of the hidden layers varied between 1 to 40, and the results of best setup of the neurons considered as representative of the performance of that number of layers. For the purpose of simplicity of the analysis, the number of neurons considered to be equal in all hidden layers. For instance, [15 15 15] resulted in the best performance for triple layer model among all possible conditions, and its results are nominated for triple layer model. According to table 2, it is evident that the double layer model has better accuracy than rest of tested number of hidden layers. Also, the results of neurons sensitivity analysis indicated that considering 30 neurons for each of the hidden layers in the model is the best choice for this model.

Next, the training function of the model needs to be analyzed. To reach this matter, four popular algorithms were selected, which are LM, Fletcher–Powell conjugate gradient (FPCG), resilient backpropagation (RB), variable learning rate backpropagation (VLRB), scaled conjugate gradient (SCG), and Polak–Ribiére conjugate gradient (PRCG). In terms of time, VLRB and FPCG have the shortest response times, indicating they are the fastest among those listed, though VLRBs accuracy is significantly lower.

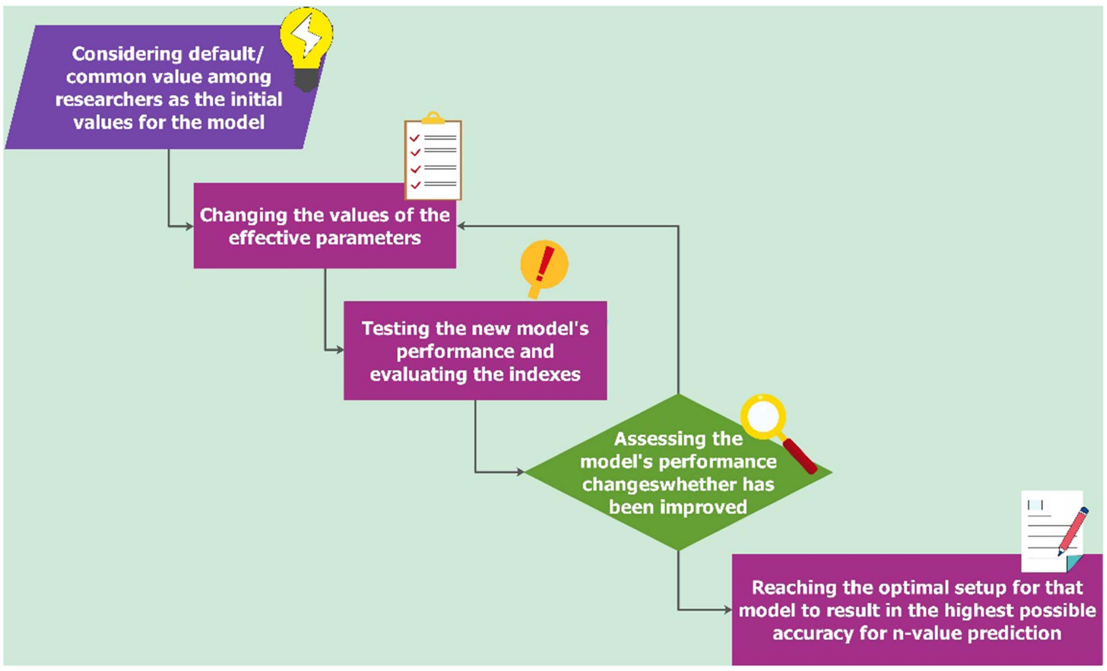

Considering default/ common value among researchers as the initial values for the model
Changing the values of the effective parameters
Testing the new model's performance and evaluating the indexes
Assessing the model's performance changeswhether has been improved
Reaching the optimal setup for that model to result in the highest possible accuracy for n-value prediction

Figure 8. A schematic of the process of sensitivity analysis.

Although LM with 20.3 ms response time is not the fastest algorithm, it might be a good choice for applications (like this work) where prediction accuracy is paramount and slightly longer response times are acceptable.

Table 3 highlights that Levenberg-Marquardt algorithm is the superior option as it has only 0.517 RMSE indicating the highest accuracy in predictions among the compared functions. A lower RMSE value means the model's predictions are closer to the actual values. In contrary, GDX with 3.717 has the lowest RMSE amongst all tested training functions, suggesting that it performs significantly worse than the others in accurately predicting outcomes. In terms of R-squared, LM with 0.9949 has the highest R-squared, which shows that the model has perfectly fitted to the data. By contrast, VLRB has the lowest R-squared value with 0.8134, demonstrating that it is less effective at predicting variance in the data compared to others. In terms of time, VLRB and FPCG have the shortest response times, indicating they are the fastest among those listed, though VLRB's accuracy is significantly lower. Although LM with 20.3 ms response time is not the fastest algorithm, it might be a good choice for applications (like this work) where prediction accuracy is paramount and slightly longer response times are acceptable.

Finally, for activation function, it was considered that the activation function between all layers except between the last hidden layer and output layer to be same. This is a common way for activation function sensitivity analysis as the total possible combinations are extensively great. The finding of this step of study that has been summarized in table 4 demonstrated that logsig-purelin is the best pair of activation functions (logsig places between input and hidden layer and the hidden layers themselves while purelin is between hidden layer and output layer.)

5. Comparison between results of different methods

The models that are explained in section 3 of this paper are trained and tested using the respective datasets for each process.

To enhance the accuracy of each proposed method, the best value of each hyperparameter of the effective parameters is considered for the final modeling with each method. After obtaining the best hyperparameter for each method, n-value prediction has been done for each ML technique separately, and then, prediction results using different performance indexes R^2, RMSE, MARE, and MAE of the models have been listed in table 5. Furthermore, figure 9 visualizes the difference in the model's accuracies in terms of all indexes which was explained in section 2.

As can be seen in table 5, which has been sorted with respect to R^2 value in descending order, the CFNN model has the highest fitting capability to the real n-value of the superconductor with a 99.68% R-squared

Table 1. The best hyperparameters of all ML models and the searched range for sensitivity analysis.

Method	Hyper parameter	Method	Hyper parameter

CFNN	n_layers = 3 (from 1 to 5)
	n_neurons = 15 (from 5 to 30)
	training_function = Levenburg–Marquardt (LM) (other tested algorithms are (SCG), (RB), (VLRB))
	activation_function = Purelin—Logsig (a pair consisting tansig, purelin, logsig, and satlin)
		SVR	kernel = 'rbf' ('rbf', 'linear', 'poly', 'sigmoid')
			C = 100 (logarithmic from 0.001 to 1000)
			epsilon = 0.01 (logarithmic from 0.001 to 1)
			degree = 3 (for 'poly' kernel)
			shrinking = True (and False)
			tol = 0.001 (logarithmic from 0.001 to 1000)
			cache_size = 200
			verbose = False (and True)

MLPNN	n_layers = 3 (from 1 to 5)
	n_neurons = 30 (from 5 to 30)
	training_function = Levenburg–Marquardt (other tested algorithms are (SCG), (RB), (VLRB))
	activation_function = Tansig—Purelin (a pair consisting tansig, purelin, logsig, and satlin)
		KNNR	n_neighbors = 10 (3–15)
			weights = 'uniform'
			algorithm = 'auto'
			leaf_size = 30 (from 10 to 50)
			metric = 'minkowski'
			metric_params = None

CatBoost	iterations = 100 000 (logarithmic from 1000 to 10 000 000)
	learning_rate = 0.001419 (autocorrection by code itself)
	depth = 10 (from 2 to 15)
	l2_leaf_reg = 3.0 (1–5)
	border_count = 254
		KRR	alpha = 0.1 (logarithmic from 0.001 to 1000)
			kernel = 'poly' ('rbf', 'linear', 'poly', 'sigmoid')
			gamma = None
			degree = 3 (for 'poly' kernel)
			coef0 = 1
			kernel_params = None

XGBoost	n_estimators = 100 000 (logarithmic from 1000 to 1000 000)
	learning_rate = 0.1 (from 0.05 to 1)
	max_depth = 3 (from 2 to 15)
	min_child_weight = 1
	subsample = 1.0 (and 0)
	colsample_bytree = 1.0 (and 0)
	gamma = 0 (and 1)
	reg_alpha = 0 (and 1)
	reg_lambda = 1 (and 0)
	scale_pos_weight = 1 (and 0)
		AdaBoost	base_estimator = None
			n_estimators = 1000 000 (logarithmic from 1000 to 10 000 000)
			learning_rate = 1.0 (from 0.5 to 1)
			loss = 'linear' (and exponential)
			random_state = None

HGBR	learning_rate = 0.1 (from 0.05 to 1)
	epsilon = 1.35
	max_iter = 100 (from 100 to 10 000)
	max_depth = None (to make it unlimited)
	min_samples_leaf = 20 (from 10 to 50)
		HR	epsilon = 1.35 (0.5 to 1.5)
			max_iter = 100 (from 100 to 10 000)
			alpha = 0.0001 (logarithmic from 0.0001 to 1000)
			warm_start = False (and True)

RFR	n_estimators = 1000 000 (logarithmic from 1000 to 1000 000)
	criterion = 'mse'
	max_depth = None (to make it unlimited)
	min_samples_split = 2
	min_samples_leaf = 1
	max_features = 'auto'
	bootstrap = True (and False)
		LR	alpha = 0.01 (logarithmic from 0.0001 to 1000)
			fit_intercept = True (and False)
			normalize = False (and True)
			precompute = False (and True)
			positive = False (and True)
			`selection = 'cyclic'
			max_iter = 1000 (logarithmic from 100 to 100 000)
			tol = 1×10^{-4}
			warm_start = False (and True)
			random_state = None

(Continued.)

Table 1. (Continued.)

Method	Hyper parameter	Method	Hyper parameter

LightGBM	n_round = 10 000 (logarithmic from 1000 to 1000 000)
	learning_rate = 0.1 (from 0.05 to 1)
	max_depth= None (to make it unlimited)
	num_leaves = 31 (from 10 to 50)
	min_child_samples = 20 (from 10 to 50)
	subsample = 1.0 (and 0)
	colsample_bytree = 1.0 (and 0)
	reg_alpha = 0.0 (and 1)
	reg_lambda = 0.0 (and 1)
		ENR	alpha = 0.001 (logarithmic from 0.0001 to 1000)
			l1_ratio = 0.5
			fit_intercept = True (and False)
			normalize = False (and True)
			precompute = False (and True)
			max_iter = 1000
			tol = 0.0001
			warm_start = False (and True)
			positive = False (and True)
			selection = 'cyclic'
			random_state = None

GBR	n_estimators = 1000 (logarithmic from 1000 to 1000 000)
	learning_rate = 0.1 (from 0.05 to 1)
	max_depth = 3 (from 2 to 15)
	min_samples_split = 2
	min_samples_leaf = 1
	subsample = 1.0 (and 0)
	max_features = None (to make it unlimited)
	alpha = 0.9
	max_leaf_nodes = None (to make it unlimited)
		Linear	fit_intercept = True (and False)
			normalize = False (and True)
			copy_X = True (and False)
			n_jobs = None

DTR	criterion = 'mse'
	splitter = 'best'
	max_depth = None (to make it unlimited)
	min_samples_split = 2 (from 1 to 15)
	min_samples_leaf = 1 (from 1 to 15)
	min_weight_fraction_leaf = 0.0
	max_features = None (to make it unlimited)
	random_state = None
		RRB	n_learners = 100 (logarithmic from 0.001 to 1000)
			learning_rate = 0.1 (from 0.05 to 1)
			loss = 'ls'
			max_depth = 4 (from 2 to 15)
			alpha = 0.001
			max_leaf_nodes = None (to make it unlimited)
			warm_start = False (and True)
			tol = 0.0001 (logarithmic from 0.0001 to 1000)

NLR	Degree = 7 (from 2 to 10)		

Table 2. Performance of CFNN model with different number of hidden layers considering the best setup of neurons for each condition.

Number of hidden layers	Setup of neurons	RMSE	*R*-Squared	Response time (ms)
1	[25]	0.919 023	0.9864	14.3
2	**[30 30]**	**0.517 361**	**0.9949**	**20.3**
3	[15 15 15]	0.524 021	0.9947	21.7
4	[30 30 30 30]	0.597 127	0.9940	218.1
5	[30 30 30 30 30]	0.577 445	0.9943	469.4

Table 3. Performance of CFNN model using different training functions.

Training function	RMSE	*R*-squared	Response time (ms)
LM	**0.517**	**0.9949**	**20.3**
SCG	1.317	0.9779	63.2
RB	1.505	0.9695	15.6
VLRB	3.717	0.8134	14.9
FPCG	1.168	0.9820	15.3
PRCG	1.126	0.9834	17.3

Table 4. Performance of CFNN model with different pairs of activation function.

Pair of activation function	RMSE	R-squared	Response time (ms)
logsig—logsig	6.425	0.0859	29.6
logsig—purelin	**0.496**	**0.9967**	**9.1**
logsig—satlin	5.157	0.7982	23.6
logsig—tansig	0.501	0.9965	28.1
purelin—logsig	5.569	0.6471	13.4
purelin—purelin	4.972	0.6118	14
purelin—satlin	5.591	0.6435	16.6
purelin—tansig	4.958	0.6118	17.8
satlin—logsig	5.187	0.7827	13
satlin—purelin	0.74	0.9937	16
satlin—satlin	5.161	0.7981	16.3
satlin—tansig	0.778	0.9924	13.9
tansig—logsig	5.173	0.7916	26.6
tansig—purelin	0.517	0.9949	20.3
tansig—satlin	5.161	0.7961	17.1
tansig—tansig	0.56	0.9959	25.2

Note: The row that has been bolded represents the best results for each step of analysis for each table with bold contents.

Table 5. The performance comparison of different ML models in terms of RMSE, R-squared, MAE, and MARE.

Method	RMSE	R-Squared	MAE	MARE (%)	Test time (ms)
CFNN	0.495 800	0.996 751	0.315 800	1.432 800	9
MLPNN	0.645 409	0.996 045	0.360 700	1.604 500	11
CatBoost	0.545 725	0.992 218	0.319 717	1.754 850	52
XGBoost	0.563 300	0.991 709	0.392 526	2.034 177	28
HGBR	0.581 071	0.991 178	0.374 948	2.154 406	378
RFR	0.601 371	0.990 550	0.366 798	1.906 172	1600
LightGBM	0.604 361	0.990 456	0.365 310	2.496 930	2800
GBR	0.703 901	0.987 736	0.440 453	2.768 740	87
DTR	0.816 340	0.982 587	0.503 291	2.581 918	14
NLR	0.864 451	0.980 401	0.583 627	3.520 835	18
SVR	0.968 318	0.976 837	0.599 249	4.069 813	1400
KNNR	1.672 581	0.926 903	0.980 620	6.376 660	69
KRR	2.364 010	0.852 786	1.810 173	11.823 296	5300
AdaBoost	2.684 893	0.811 673	2.201 210	11.891 801	228
HR	4.862 069	0.379 983	3.622 299	18.917 729	15
LR	4.863 364	0.379 653	3.655 027	18.877 112	17
ENR	4.863 437	0.379 634	3.655 438	18.882 267	14
Linear	4.863 448	0.379 631	3.655 493	18.882 941	11
RRB	4.923 443	0.372 946	3.696 345	18.791 929	184

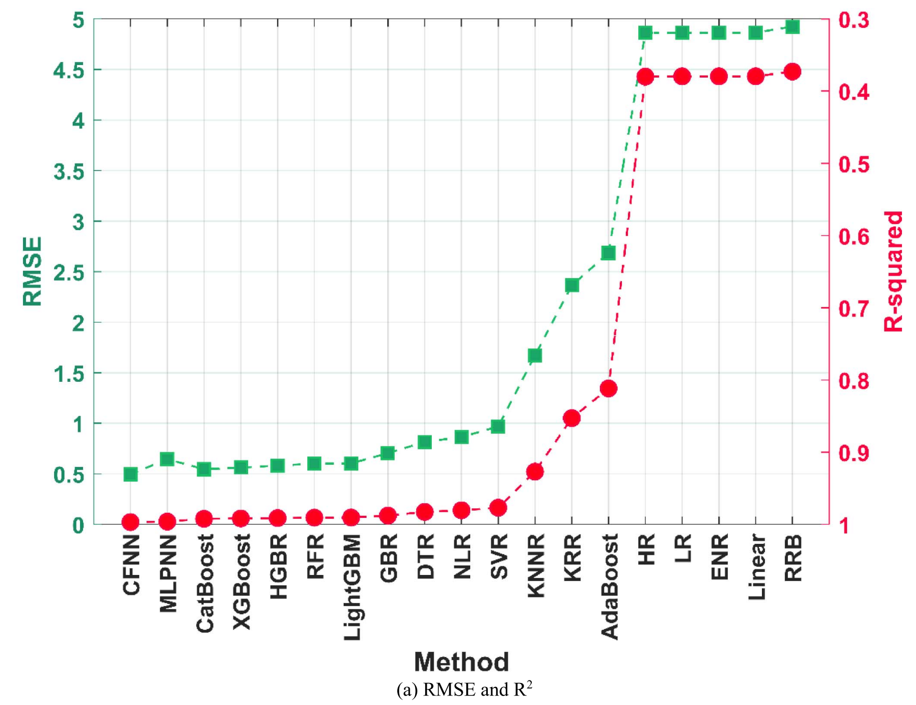
value. Moreover, it also has the lowest RMSE which proves that the prediction of the model based on CFNN has low error.

Using the same logic, the second-best option for the n-value estimation for this type of superconductor is MLPNN, which is a simpler version of ANN than CFNN. It is a very slight difference in terms of R^2 value, but the RMSE is about 30% higher than CFNN which means that it is not as very accurate as CFNN due to its higher error. This is because there are quite a few points that have very large error in FFNN model. Therefore, while the R-squared has not changed very much, the RMSE value soars. The figure 10 shows the results of the three best ML models for n-value prediction. As a visual example, in figure 10, the FFNN model is not fitted to the actual points, caused errors near 110%, whilst for CFNN model it never exceeds 12% for the same temperature.

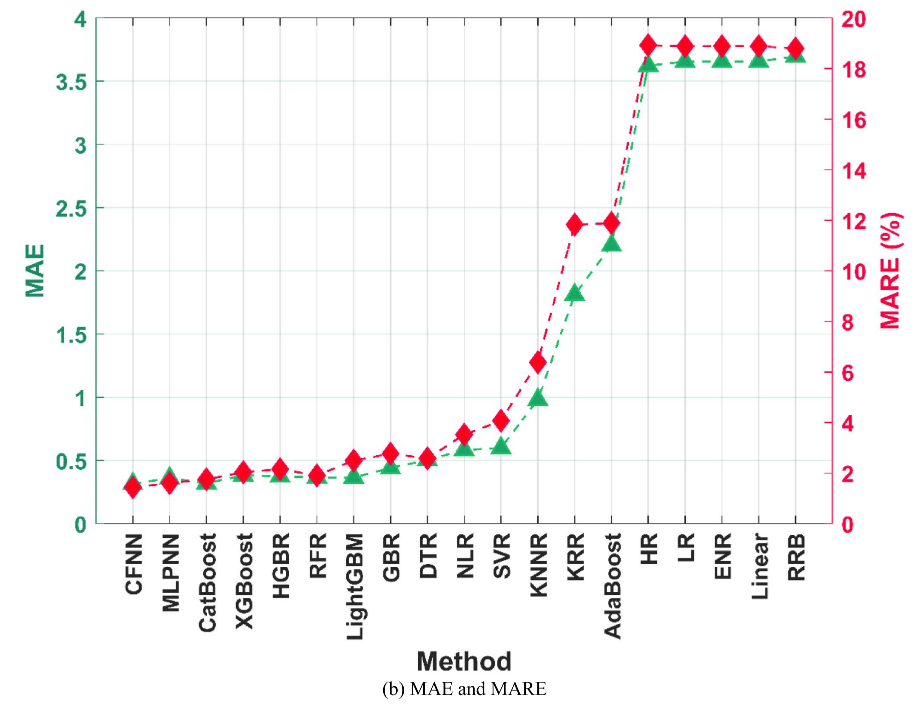
The XGBoost, HGBR, RFR, and LightGBM in the next places can be considered the alternative options instead of ANN models as all of them still have very high fitting accuracy.

In contrast, the RRB and all the linear regression including Huber, Lasso, Elastic Net, and traditional linear regression have the worst accuracies, making them a weak option for n-value estimation. These models have almost more than 10 times RMSE than the CFNN model.

After checking the accuracy of all models, in the following, the predictability of the models for specific temperatures will be proposed and then compared. Since there are 16 different temperatures that have been



(a) RMSE and R^2



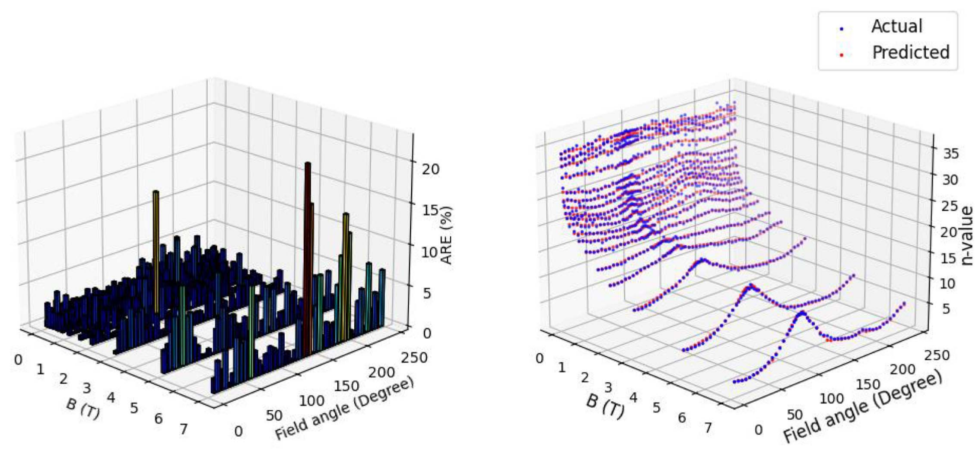
(b) MAE and MARE

Figure 9. The performance comparison of different ML models in terms of (a) RMSE and R-squared (b) MAE and MARE.

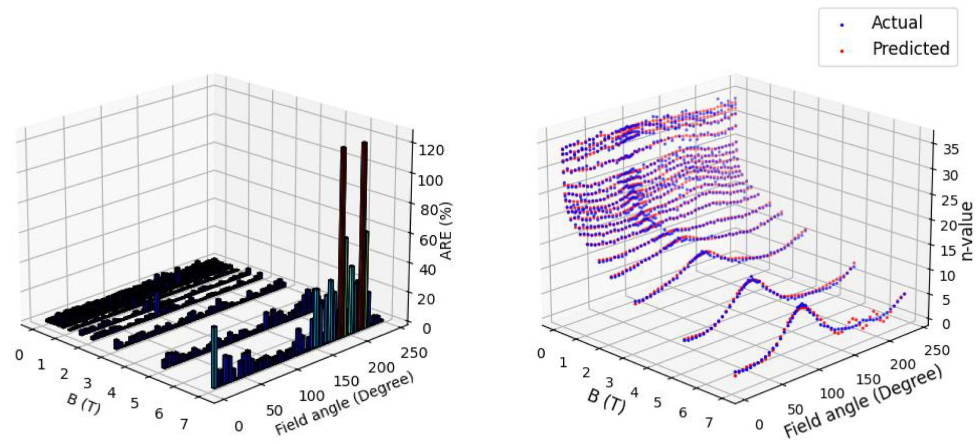
investigated in this study, so, for the purposes of making comparisons between models, a common temperature should be chosen for all methods. Also, due to the importance of 77.5 K temperature in the context of superconductors, which is the boil-off temperature of liquid Nitrogen, it is selected for reporting the performance of the model in figure 10. As can be seen, ARE of most of these models is increased in higher magnetic fields [11]. The results of the other 16 ML models for prediction on *n*-value of superconductors are shown in the figure A1 in the appendix.

6. Detailed results of the best model: CFNN

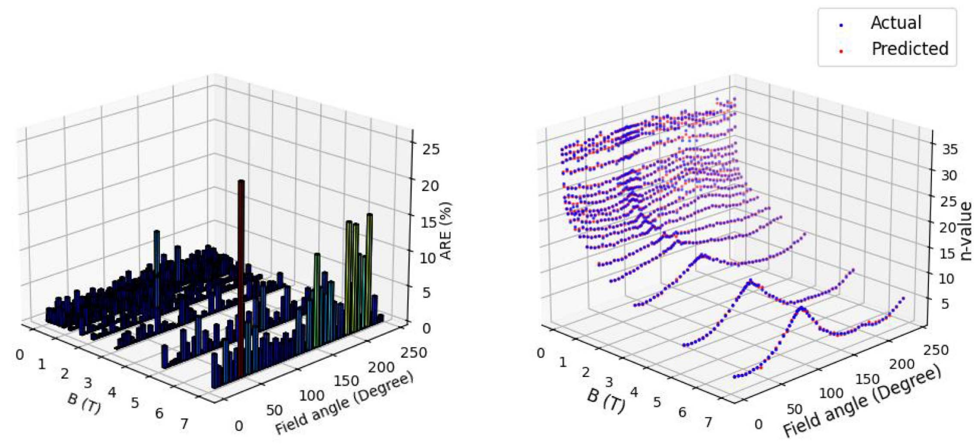
In accordance with the discourse in section 5, a highly accurate model for predicting the *n*-value of this superconductor has been established using the CFNN method. Consequently, this subsection is dedicated to



Cascade Forward Neural Network



Multi-Layer Perceptron Neural Network



CatBoost

Figure 10. Actual and predicted n-values (right figures) and ARE (left figures) of 3 best ML models for T = 77.5 K.

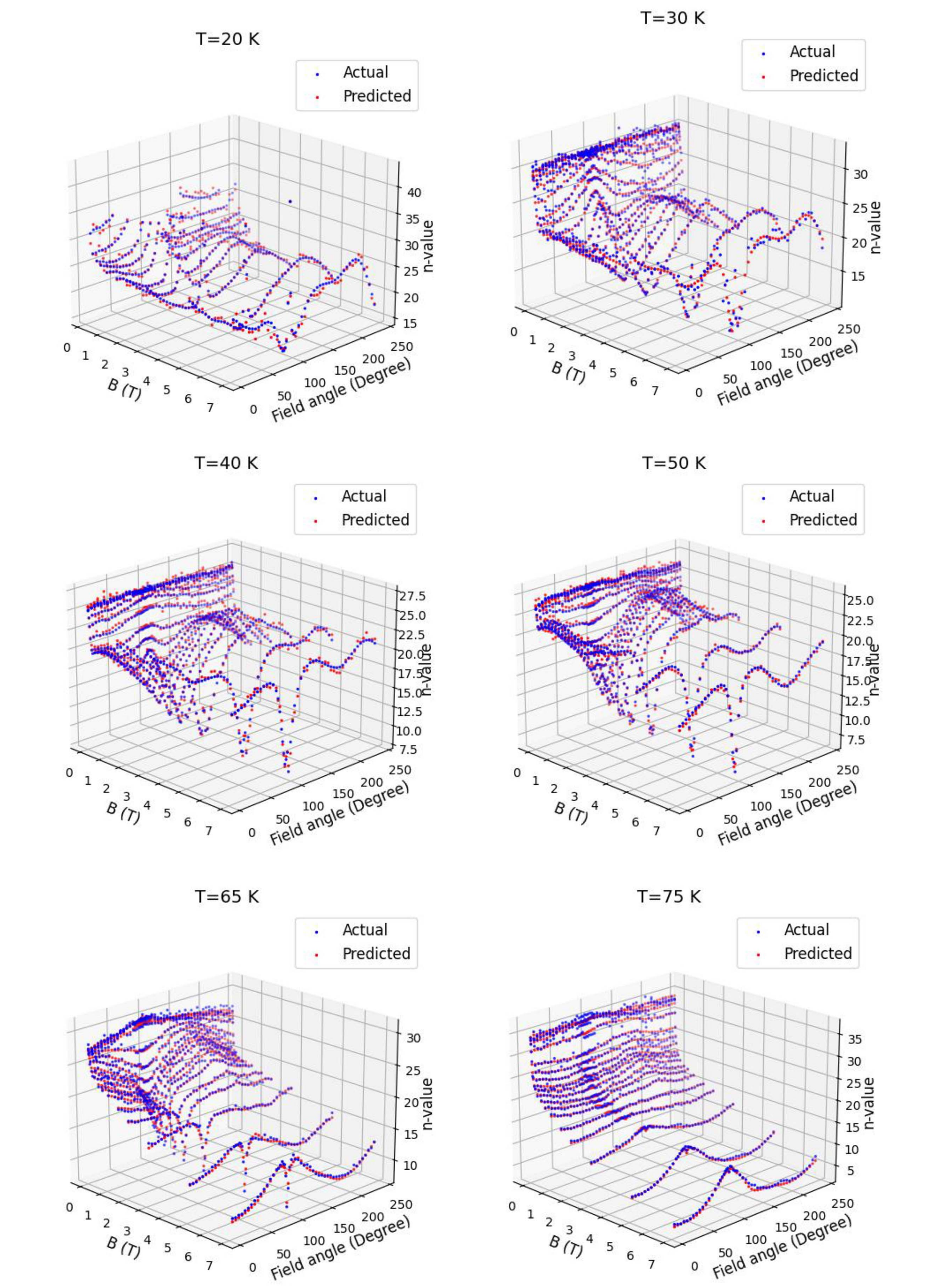


Figure 11. Actual and predicted *n*-value for CFNN model in temperatures 20, 30, 40, 50, 65, and 75 K.

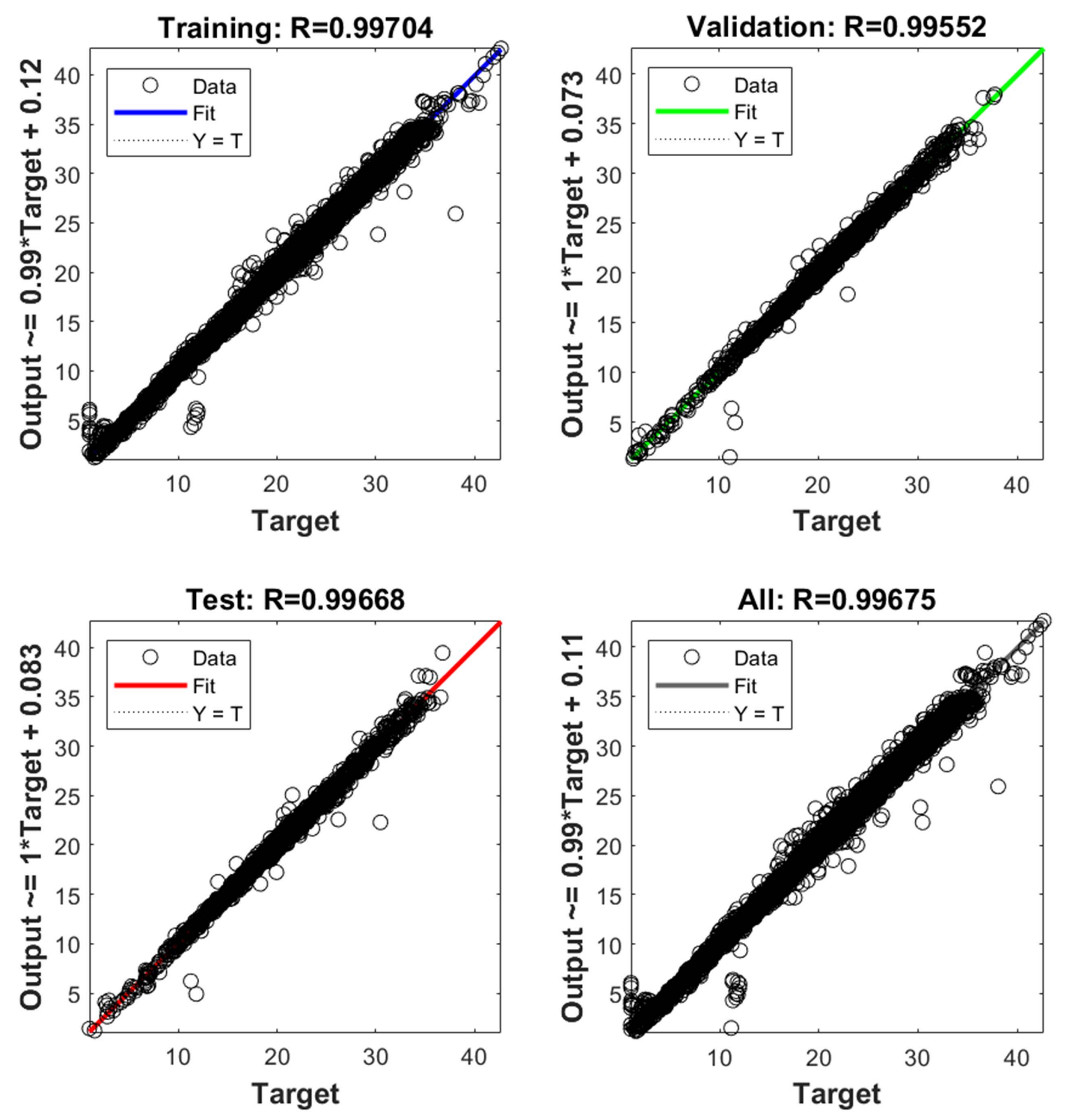


Figure 12. Regression plot for proposed CFNN model for training, validation, and testing stages.

providing a comprehensive analysis of CFNNs results. As illustrated in figure 11, it becomes evident that the model excels in n-value prediction under conditions of lower magnetic fields. Furthermore, the model exhibits heightened accuracy in scenarios characterized by elevated temperatures.

Also, figure 12 demonstrates how the CFNN model is well-fitted to the training, validation, and testing datasets. In this type of diagram, which is very common among ML researchers, data points are typically scattered across a graph, and a straight line is fitted to these points in such a way that it minimizes the overall distance between the line and the data points. The slope of the line indicates the change in the dependent variable for a one-unit change in the independent variable. The vertical axis intercept represents the value of the dependent variable when the independent variable is zero. The closer this line is to the $x = y$ line, the more accurately the model predicts. According to the line equations in figure 12, the lines are very close to the ideal condition.

7. Conclusion

The *n*-value, an important parameter in superconductors characterization subject to significant variation under different magnetic fields (including magnitude and angle of exposure to the wire) and temperature conditions in superconductor materials, has been the focus of an investigation into *n*-value prediction utilizing a range of ML methods. Within the findings of our paper, it became evident that the CFNN model emerged as the most favored choice for this task, highlighting its achievement of the lowest RMSE at 0.4958 when compared to the alternative models. Additionally, the CFNN model demonstrated the highest R-squared value among all the models, an impressive 99.6751%, indicating its robust training and excellent adaptation to the training dataset. This high *R*-squared value serves as evidence that the model is adept at predicting the *n*-value with a commendable level of accuracy. Consequently, the CFNN model offers a promising avenue for precise *n*-value forecasting within the context of superconducting materials under varying magnetic fields and temperature conditions.

Data availability statement

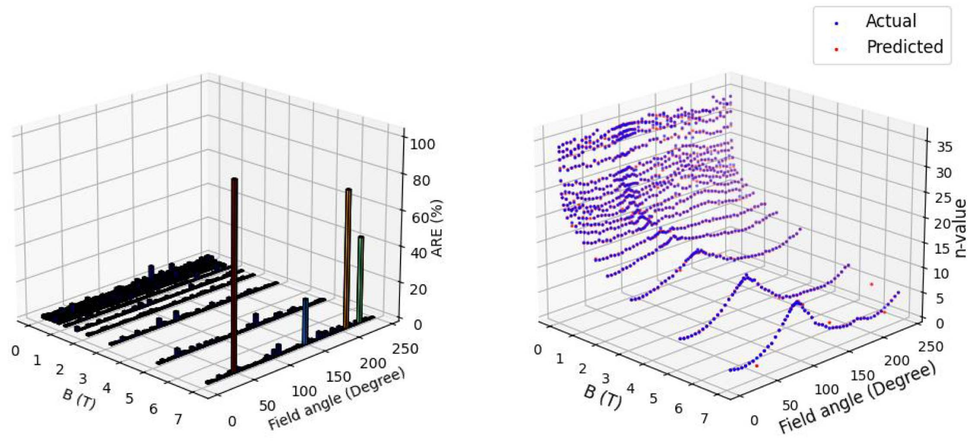
All data that support the findings of this study are included within the article (and any supplementary files).

Acknowledgments

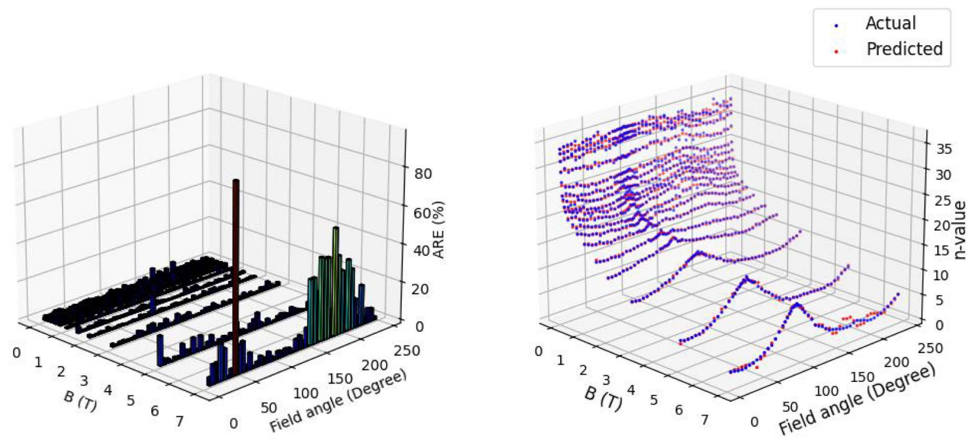
The authors would like to thank Dr. S C Wimbush, Dr. N M Strickland of Robinson Research Institutes for providing openly accessible experimental data for index-value characteristics of HTS tapes on the public website. Indeed, this work was not accomplishable without having access to such a comprehensive data set.

For the purpose of open access, the author(s) has applied a Creative Commons Attribution (CC BY) license to any Author Accepted Manuscript version arising from this submission.

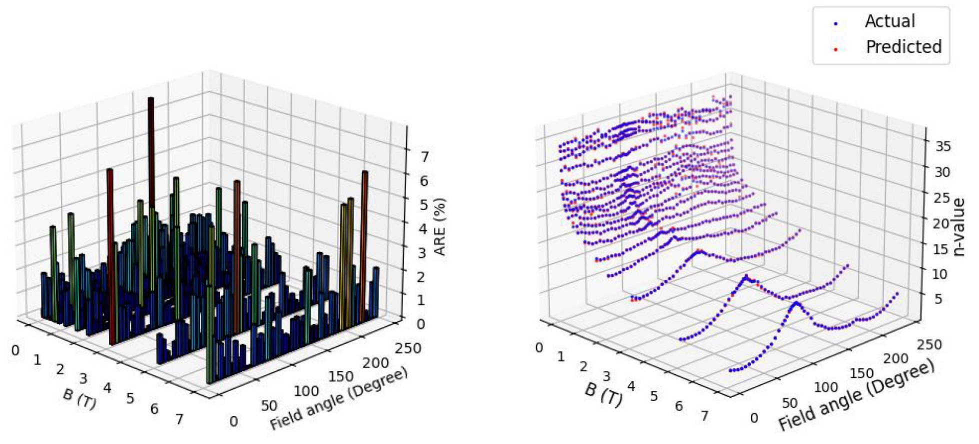
Appendix



XGBoost

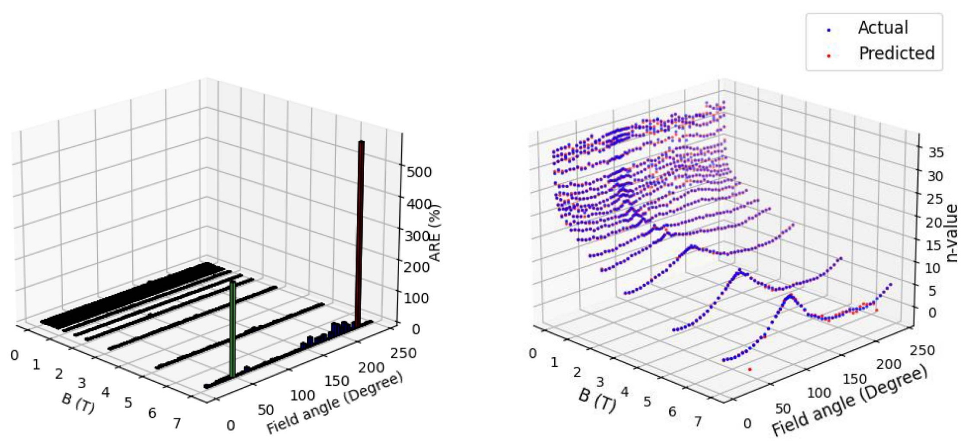


Hist Gradient Boosting Regression

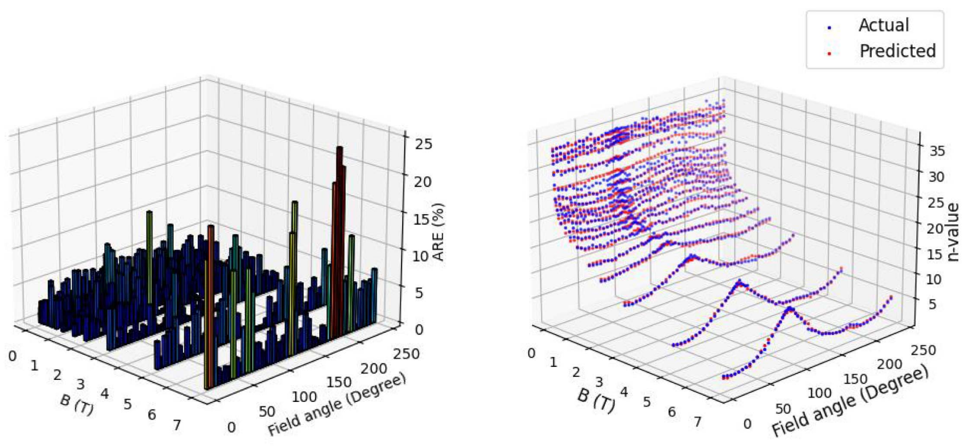


Random Forest Regression

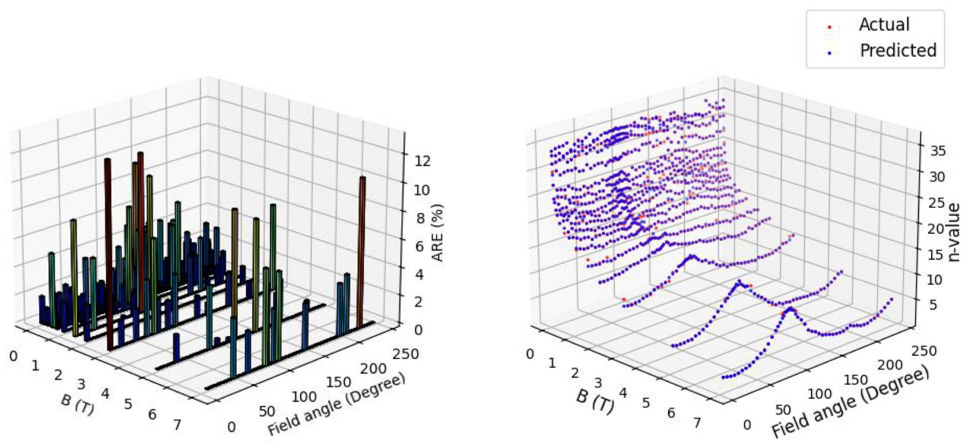
Figure A1. Actual and predicted n-values (right figures) and ARE (left figures) of different ML models for $T = 77.5$ K.



LightGBM

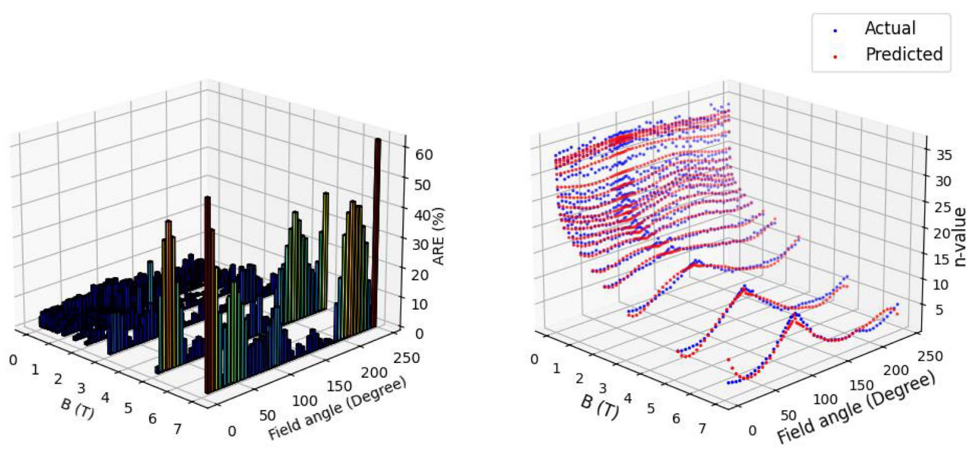


Gradient Boost Regression

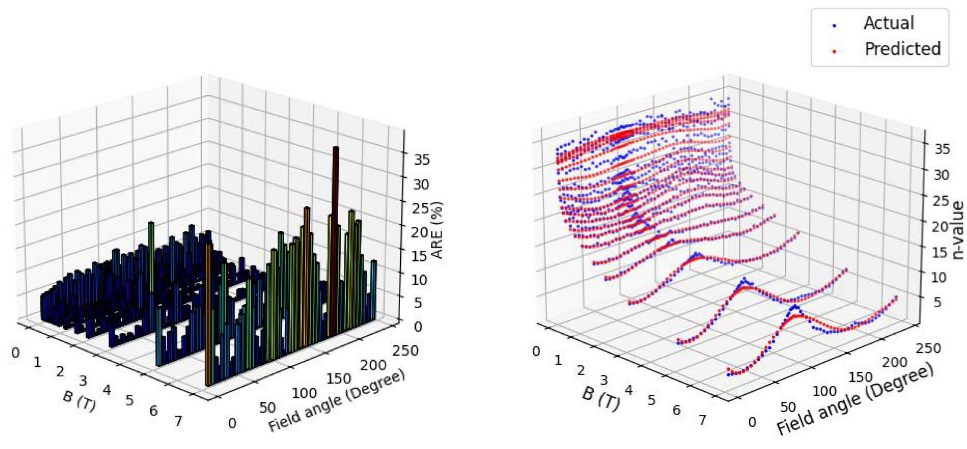


Decision Tree Regression

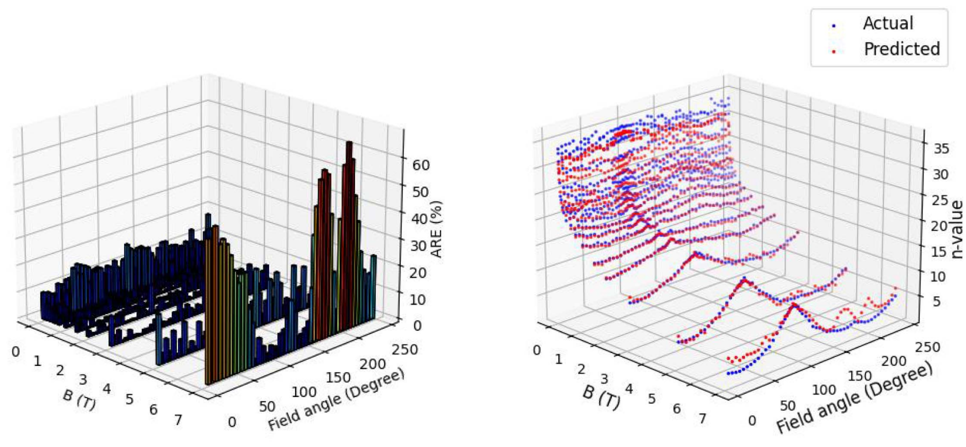
Figure A1. (Continued.)



Non-linear regression

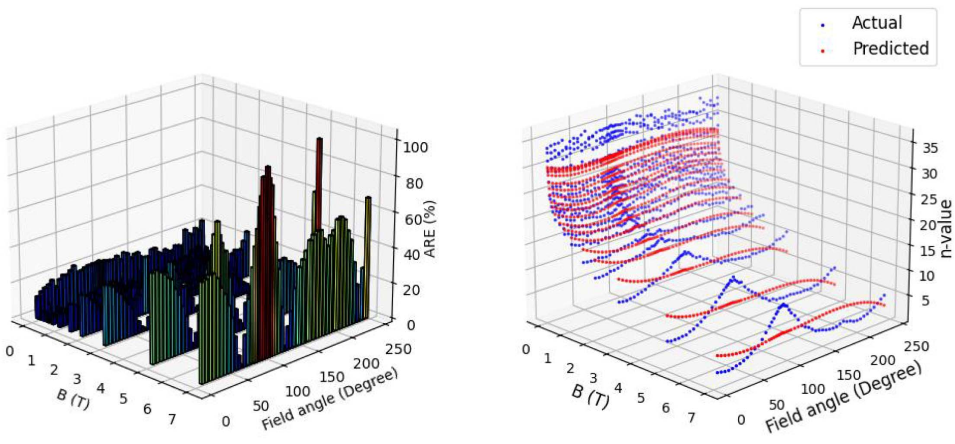


Support Vector Regression

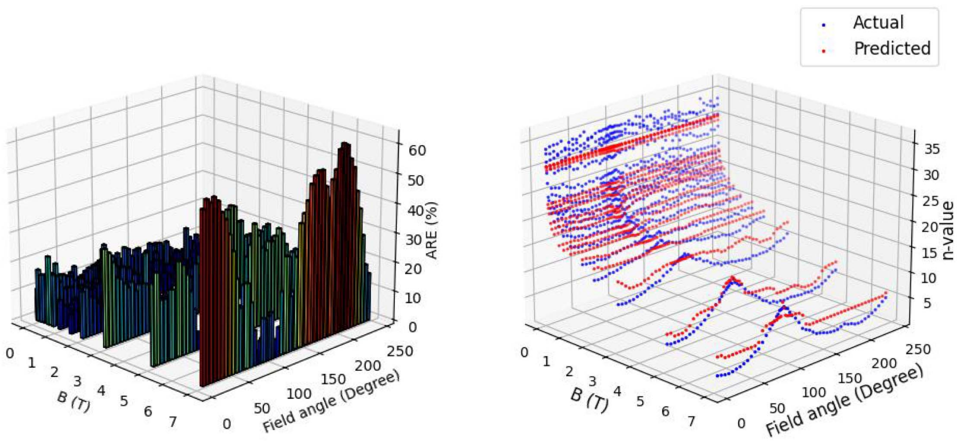


K-Nearest Neighbours Regression

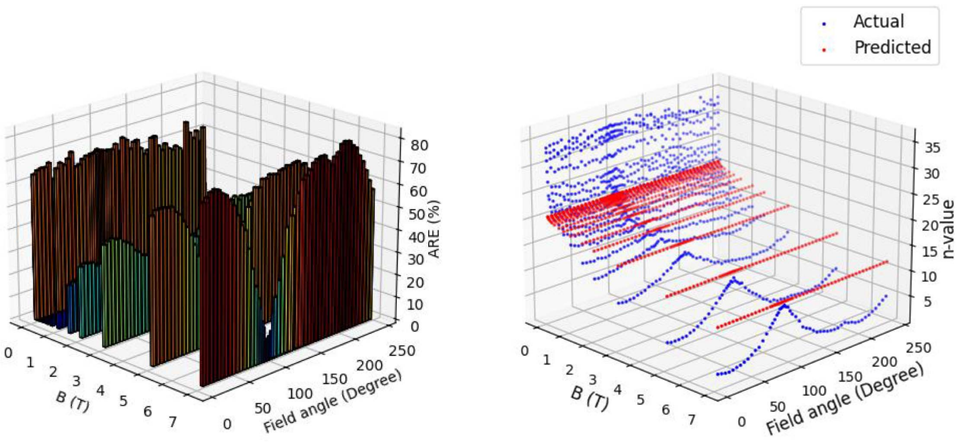
Figure A1. (Continued.)



Kernel Rigid Regression

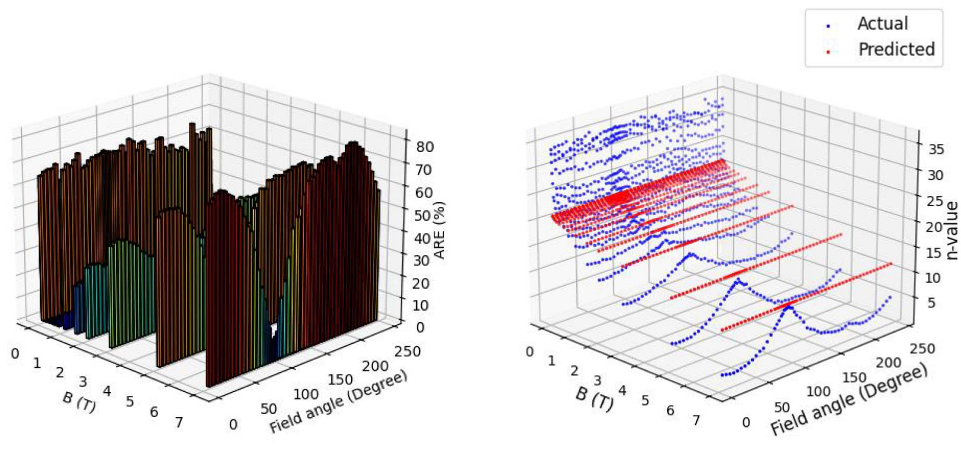


AdaBoost

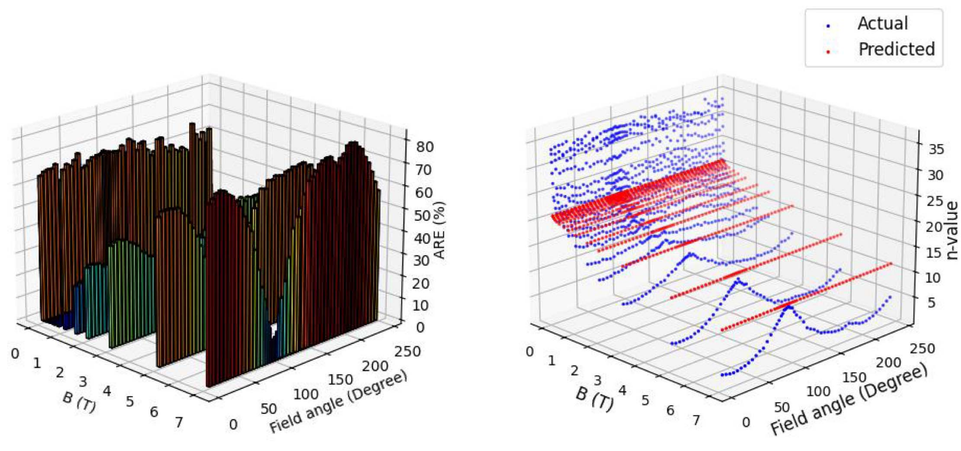


Huber Regression

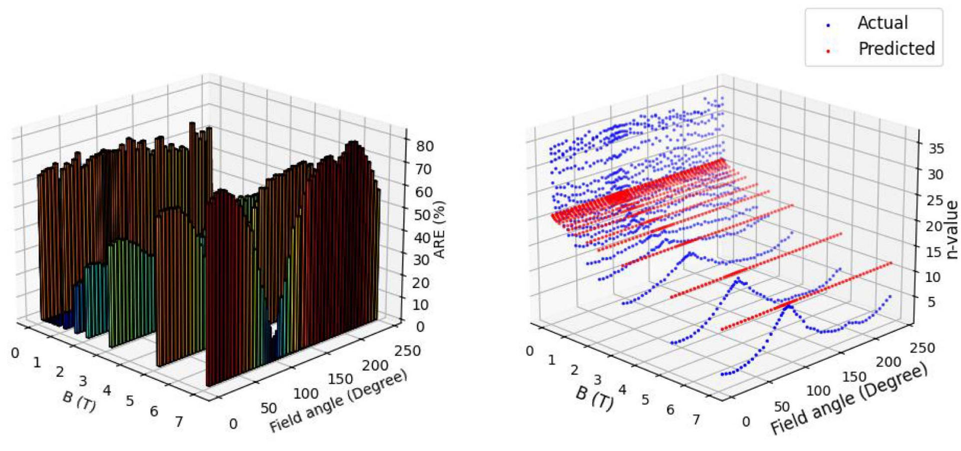
Figure A1. (Continued.)



Lasso Regression

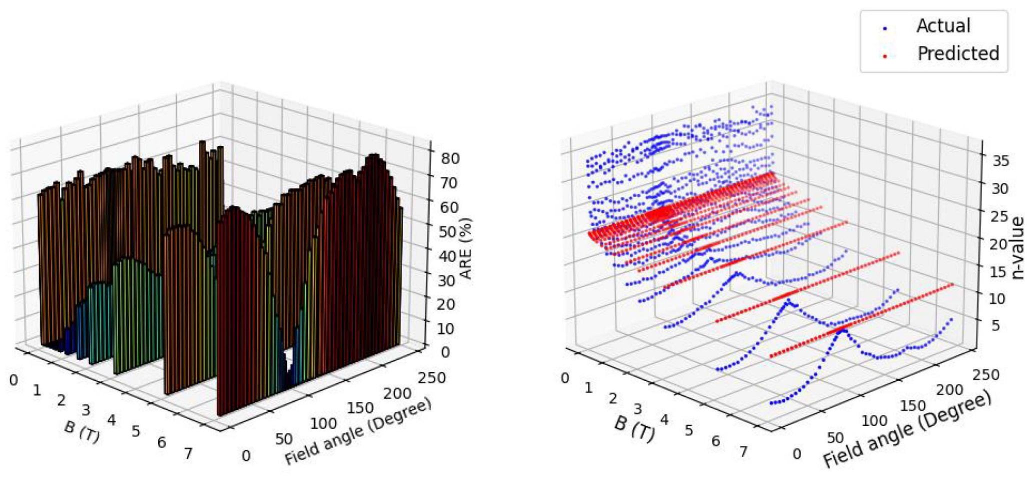


Elastic Net Regression



Linear Regression

Figure A1. (Continued.)



Ridge Regression Booster

Figure A1. (Continued.)

ORCID iDs

Shahin Alipour Bonab https://orcid.org/0009-0002-8316-4336
Giacomo Russo https://orcid.org/0000-0002-2011-8423
Antonio Morandi https://orcid.org/0000-0002-1845-4006
Mohammad Yazdani-Asrami https://orcid.org/0000-0002-7691-3485

References

[1] Brandt E H 1998 Superconductor disks and cylinders in an axial magnetic field. I. Flux penetration and magnetization curves *Phys. Rev.* B **58** 6506–22
[2] Brandt E H 1996 Superconductors of finite thickness in a perpendicular magnetic field: strips and slabs *Phys. Rev.* B **54** 4246
[3] Morandi A 2012 2D electromagnetic modelling of superconductors *Supercond. Sci. Technol.* **25** 104003
[4] Yeshurun Y, Malozemoff A P and Shaulov A 1996 Magnetic relaxation in high-temperature superconductors *Rev. Mod. Phys.* **68** 911–49
[5] Matsushita T, Matsuda A and Yanagi K 1993 Irreversibility line and flux pinning properties in high-temperature superconductors," *Physica* C **213** 477–82
[6] Kim J H, Dou S X, Matsumoto A, Choi S, Kiyoshi T and Kumakura H 2010 Correlation between critical current density and n-value in MgB_2/Nb/Monel superconductor wires *Physica* C **470** 1207–10
[7] Romanovskii V, Watanabe K and Ozhogina V 2009 Thermal peculiarities of the electric mode formation of high temperature superconductors with the temperature-decreasing n-value *Cryogenics* **49** 360–5
[8] Marchevsky M 2021 Quench detection and protection for high-temperature superconductor accelerator magnets *Instruments* **5** 27
[9] Bykovskiy N, Bajas H, Dicuonzo O, Bruzzone P and Sedlak K 2023 Experimental study of stability, quench propagation and detection methods on 15 kA sub-scale HTS fusion conductors in SULTAN *Supercond. Sci. Technol.* **36** 034002
[10] Godeke A 2023 High temperature superconductors for commercial magnets *Supercond. Sci. Technol.* **36** 113001
[11] Bruzzone P, Fietz W H, Minervini J V, Novikov M, Yanagi N, Zhai Y and Zheng J 2018 High temperature superconductors for fusion magnets *Nucl. Fusion* **58** 103001
[12] Rimikis A, Kimmich R and Schneider T 2000 Investigation of n-values of composite superconductors *IEEE Trans. Appl. Supercond.* **10** 1239–42
[13] Oh S, Choi H, Lee C, Lee S, Yoo J, Youm D, Yamada H and Yamasaki H 2007 Relation between the critical current and the n value of ReBCO thin films: a scaling law for flux pinning of ReBCO thin films *J. Appl. Phys.* **102** 043904
[14] Oh S and Kim K 2006 A scaling law for the critical current of Nb3Sn stands based on strong-coupling theory of superconductivity *J. Appl. Phys.* **99** 033909
[15] Russo G, Yazdani-Asrami M, Scheda R, Morandi A and Diciotti S 2022 Artificial intelligence-based models for reconstructing the critical current and index-value surfaces of HTS tapes *Supercond. Sci. Technol.* **35** 124002
[16] Zhu L, Wang Y, Meng Z and Wang T 2022 Critical current and n-value prediction of second-generation high temperature superconducting conductors considering the temperature-field dependence based on the back propagation neural network with encoder *Supercond. Sci. Technol.* **35** 104002
[17] Wimbush S C and Strickland N M 2017 A public database of high-temperature superconductor critical current data *IEEE Trans. Appl. Supercond.* **27** 1–5
[18] High-temperature superconducting wire critical current database (available at: https://htsdb.wimbush.eu/) (Accessed 27 October 2023)
[19] Strickland N M, Hoffmann C and Wimbush S C 2014 A 1 kA-class cryogen-free critical current characterization system for superconducting coated conductors *Rev. Sci. Instrum.* **85** 113907

[20] Russo G and Morandi A 2023 Evaluation of the performance of commercial high temperature superconducting tapes for dynamo flux pump applications *Energies* **16** 7244
[21] Yazdani-Asrami M, Sadeghi A, Seyyedbarzegar S and Song W 2022 DC electro-magneto-mechanical characterization of 2G HTS tapes for superconducting cable in magnet system using artificial neural networks *IEEE Trans. Appl. Supercond.* **32** 1–10
[22] Yazdani-Asrami M, Sadeghi A and Song W 2023 Ultra-fast surrogate model for magnetic field computation of a superconducting magnet using multi-layer artificial neural networks *J. Supercond. Nov. Magn.* **36** 575–86
[23] Ituabhor O, Isabona J, Zhimwang J T and Risi I 2022 Cascade forward neural networks-based adaptive model for real-time adaptive learning of stochastic signal power datasets *Int. J. Comput. Netw. Inf. Secur.* **14** 63–74
[24] Yazdani-Asrami M, Sadeghi A, Seyyedbarzegar S M and Saadat A 2022 Advanced experimental-based data-driven model for the electromechanical behavior of twisted YBCO tapes considering thermomagnetic constraints *Supercond. Sci. Technol.* **35** 054004
[25] Lee W, Park D, Bascuñán J and Iwasa Y 2022 Artificial intelligence methods for applied superconductivity: material, design, manufacturing, testing, operation, and condition monitoring *Supercond. Sci. Technol.* **35** 105007
[26] Ekonomou L, Fotis G P, Maris T I and Liatsis P 2007 Estimation of the electromagnetic field radiating by electrostatic discharges using artificial neural networks *Simul. Modelling Pract. Theory* **15** 1089–102
[27] Yazdani-Asrami M, Fang L, Pei X and Song W 2023 Smart fault detection of HTS coils using artificial intelligence techniques for large-scale superconducting electric transport applications *Supercond. Sci. Technol.* **36** 085021
[28] Yazdani-Asrami M, Taghipour-Gorjikolaie M, Song W, Zhang M, Chakraborty S and Yuan W 2021 Artificial intelligence for superconducting transformers *Transformers Mag.* **8** 22–30 (available at: www.transformers-magazine.com)
[29] Alzayed M, Chaoui H and Farajpour Y 2021 Maximum power tracking for a wind energy conversion system using cascade-forward neural networks *IEEE Trans. Sustain. Energy* **12** 2367–77
[30] Mohammadi M-R, Hemmati-Sarapardeh A, Schaffie M, Husein M M and Ranjbar M 2021 Application of cascade forward neural network and group method of data handling to modeling crude oil pyrolysis during thermal enhanced oil recovery *J. Pet. Sci. Eng.* **205** 108836
[31] Alipour Bonab S, Song W and Yazdani-Asrami M 2023 A new intelligent estimation method based on the cascade-forward neural network for the electric and magnetic fields in the vicinity of the high voltage overhead transmission lines *Appl. Sci.* **13** 11180
[32] Sadeghi A, Alipour Bonab S, Song W and Yazdani-Asrami M 2024 Intelligent estimation of critical current degradation in HTS tapes under repetitive overcurrent cycling for cryo-electric transportation applications *Mater. Today Phys.* **42** 101365
[33] Hastie T, Tibshirani R and Friedman J 2009 The elements of statistical learning (https://doi.org/10.1007/978-0-387-84858-7)
[34] Breiman L, Friedman J H, Olshen R A and Stone C J 1984 *Classification And Regression Trees* (Routledge) (https://doi.org/10.1201/9781315139470)
[35] Balogun A-L and Tella A 2022 Modelling and investigating the impacts of climatic variables on ozone concentration in Malaysia using correlation analysis with random forest, decision tree regression, linear regression, and support vector regression *Chemosphere* **299** 134250
[36] Breiman L 2001 Random forests *Mach. Learn.* **45** 5–32
[37] Friedman J, Hastie T and Tibshirani R 2000 Additive logistic regression: a statistical view of boosting *Ann. Statist.* **28** 337–407
[38] Zhang G, Shin S and Jung J 2023 Cascade forest regression algorithm for non-invasive blood pressure estimation using PPG signals *Appl. Soft Comput.* **144** 110520
[39] Guo X, Gui X, Xiong H, Hu X, Li Y, Cui H, Qiu Y and Ma C 2023 Critical role of climate factors for groundwater potential mapping in arid regions: insights from random forest, XGBoost, and LightGBM algorithms *J. Hydrol.* **621** 129599
[40] Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, Ye Q and Liu T Y LightGBM: a highly efficient gradient boosting decision tree (available at: https://github.com/Microsoft/LightGBM)
[41] Ong Y J, Zhou Y, Baracaldo N and Ludwig H 2020 Adaptive histogram-based gradient boosted trees for federated learning (arXiv:2012.06670)
[42] Nhat-Duc H and Van-Duc T 2023 Comparison of histogram-based gradient boosting classification machine, random forest, and deep convolutional neural network for pavement raveling severity classification *Autom. Constr.* **148** 104767
[43] Guryanov A 2019 Histogram-based algorithm for building gradient boosting ensembles of piecewise linear decision trees *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Springer) pp 39–50
[44] Li L, Qiao J, Yu G, Wang L, Li H-Y, Liao C and Zhu Z 2022 Interpretable tree-based ensemble model for predicting beach water quality *Water Res.* **211** 118078
[45] Chen T and Guestrin C 2016 XGBoost: a scalable tree boosting system (https://doi.org/10.1145/2939672.2939785)
[46] Cao W, Liu Y, Mei H, Shang H and Yu Y 2023 Short-term district power load self-prediction based on improved XGBoost model *Eng. Appl. Artif. Intell.* **126** 106826
[47] Dorogush A V, Ershov V and Yandex A G 2023 CatBoost: gradient boosting with categorical features support (arXiv:1810.11363)
[48] Prokhorenkova L, Gusev G, Vorobev A, V. Dorogush A and Gulin A 2017 CatBoost: unbiased boosting with categorical features (arXiv:1706.09516)
[49] Rastgoo A and Khajavi H 2023 A novel study on forecasting the airfoil self-noise, using a hybrid model based on the combination of CatBoost and arithmetic optimization algorithm *Expert Syst. Appl.* **229** 120576
[50] Huang G, Wu L, Ma X, Zhang W, Fan J, Yu X, Zeng W and Zhou H 2019 Evaluation of CatBoost method for prediction of reference evapotranspiration in humid regions *J. Hydrol.* **574** 1029–41
[51] Lu C, Zhang S, Xue D, Xiao F and Liu C 2022 Improved estimation of coalbed methane content using the revised estimate of depth and CatBoost algorithm: a case study from southern Sichuan Basin, China *Comput. Geosci.* **158** 104973
[52] Bates D M and Watts D G 1988 *Nonlinear Regression Analysis and Its Applications* (Wiley) (https://doi.org/10.1002/9780470316757)
[53] Ezimand K and Kakroodi A A 2019 Prediction and spatio—temporal analysis of ozone concentration in a metropolitan area *Ecol. Indic.* **103** 589–98
[54] Feng Y and Wu Q 2022 A statistical learning assessment of Huber regression *J. Approx. Theory* **273** 105660
[55] Czajkowski M, Jurczuk K and Kretowski M 2023 Steering the interpretability of decision trees using lasso regression—an evolutionary perspective *Inf. Sci.* **638** 118944
[56] Minaravesh B and Aydin O 2023 Environmental and demographic factors affecting childhood academic performance in Los Angeles county: a generalized linear elastic net regression model *Remote Sens. Appl.* **30** 100942
[57] Zheng Y, Ge Y, Muhsen S, Wang S, Elkamchouchi D H, Ali E and Ali H E 2023 New ridge regression, artificial neural networks and support vector machine for wind speed prediction *Adv. Eng. Softw.* **179** 103426

[58] Wang X, Wang X, Ma B, Li Q, Wang C and Shi Y 2023 High-performance reversible data hiding based on ridge regression prediction algorithm *Signal Process.* **204** 108818
[59] Welling M (available at: https://web2.qatar.cmu.edu/~gdicaro/10315-Fall19/additional/welling-notes-on-kernel-ridge.pdf) (Accessed 25 October 2023)
[60] Cortes C, Vapnik V and Saitta L 1995 *Support-Vector Networks Editor* (Kluwer Academic Publishers)
[61] Drucker H, Burges C, Kaufman L, Smola A and Vapoik V 1996 Support vector regression machines *Adv. Neural Inf. Process. Syst.* **28** 779–84
[62] Hsu C-W, Chang C-C and Lin C-J 2003 A practical guide to support vector classification (available at: www.csie.ntu.edu.tw/∼cjlin)
[63] Li Y, Huang X, Tang J, Li S and Ding P 2023 A steps-ahead tool wear prediction method based on support vector regression and particle filtering *Measurement* **218** 113237
[64] Chen K, Liao Q, Liu K, Yang Y, Gao G and Wu G 2023 Capacity degradation prediction of lithium-ion battery based on artificial bee colony and multi-kernel support vector regression *J. Energy Storage* **72** 108160
[65] Keramat-Jahromi M, Mohtasebi S S, Mousazadeh H, Ghasemi-Varnamkhasti M and Rahimi-Movassagh M 2021 Real-time moisture ratio study of drying date fruit chips based on on-line image attributes using kNN and random forest regression methods *Measurement* **172** 108899
[66] Gou J, Xiong T and Kuang Y 2011 A novel weighted voting for K-nearest neighbor rule *J. Comput.* **6** 833–40
[67] Sumayli A 2023 Development of advanced machine learning models for optimization of methyl ester biofuel production from papaya oil: gaussian process regression (GPR), multilayer perceptron (MLP), and K-nearest neighbor (KNN) regression models *Arab. J. Chem.* **16** 104833
[68] Fix E and Hodges J L 1989 Discriminatory analysis. Nonparametric discrimination: consistency properties *Int. Stat. Rev.* **57** 238
[69] Wen L, Li Y, Zhao W, Cao W and Zhang H 2023 Predicting the deformation behaviour of concrete face rockfill dams by combining support vector machine and AdaBoost ensemble algorithm *Comput. Geotech.* **161** 105611
[70] Freund Y and Schapire R E 1996 Experiments with a new boosting algorithm (available at: www.research.att.com/)
[71] Taylor R, Ojha V, Martino I and Nicosia G 2021 Sensitivity analysis for deep learning: ranking hyper-parameter influence *2021 IEEE 33rd Int. Conf. on Tools with Artificial Intelligence (ICTAI)* (*November 2021*) (IEEE) pp 512–6