Journal of **N**onlinear **A**nalysis and **O**ptimization
Vol. 15, Issue. 1, No.11: 2024
ISSN : **1906-9685**

HARNESSING NATURE INSPIRED OPTIMIZATION TECHNIQUE USING HYPERPARAMETER TUNING IN MACHINE LEARNING ALGORITHM

T.CHANDRIKA (20NM1A05H7), P.AVANTHIKA *(20NM1A05D5)* , *S.ANUSHA (20NM1A05G7) Bachelorof Technology* in *Computer Science & Engineering*
Dr. D. Karun Kumar Reddy Associate Professor *Vignan's Institute of Engineering for Women*

ABSTRACT
Hyper parameter tuning plays a critical role in the success of machine learning models by optimizing the configuration settings that govern model performance. Traditional methods such as grid search and random search are effective but computationally expensive, especially for complex models with numerous hyper parameters. To address this challenge, nature-inspired optimization techniques have emerged as promising alternatives due to their ability to efficiently explore large search spaces and find near-optimal solutions. Specifically, algorithms inspired by natural phenomena such as swarm behavior, evolutionary processes, and foraging strategies are explored. Each optimization algorithm leverages unique principles derived from nature to guide the search for optimal hyper parameter configurations. Experimental results demonstrate the effectiveness of the nature-inspired optimization techniques in improving the performance of machine learning models through hyper parameter tuning. By efficiently navigating the hyper parameter space, these algorithms enable the discovery of configurations that lead to enhanced predictive accuracy, faster convergence, and improved generalization. In conclusion, integrating nature-inspired optimization techniques into hyper parameter tuning processes offers a powerful approach to enhance the performance and efficiency of machine learning models. Future research directions may explore hybrid approaches that combine multiple optimization algorithms or adapt these techniques to emerging paradigms such as deep learning and reinforcement learning. KEYWORDS: Hyper parameter tuning, Nature-inspired optimization, Differential Evolution, Optimization algorithms, Model optimization, Search space exploration, Computational efficiency, Performance improve.

1. INTRODUCTION

In the realm of health management, maintaining optimal blood glucose levels stands as a cornerstone of general well-being, especially for those who are active or prone to diseases such as diabetes. The delicate balance of blood sugar levels impacts various functions of the body, and deviation from the norm can lead to a whole series of health complications. Traditionally, monitoring blood glucose levels has relied heavily on invasive techniques such as blood tests, which can be uncomfortable, and sometimes impractical for continuous monitoring. However, recent strides in medical technology and research have shown a promising path: the exploration of superficial body features as potential indicators of blood glucose levels.

Superficial body features encompass a spectrum of observable characteristics, ranging from skin texture and color to subtle facial expressions and even the composition of breath. These outward manifestations of internal physiology offer attractive prospects for non-invasive and continuous monitoring of blood glucose levels. By harnessing the body and its subtle cues and signals, researchers and clinicians aim to develop innovative approaches that can provide real-time insight into glucose dynamics, enabling more proactive and personalized management strategies.

Blood glucose level refers to the concentration of glucose (a type of sugar) present in the bloodstream. Glucose is an important source of energy for cells, and its levels are strictly regulated by the body to

ensure proper functioning. Maintaining proper blood glucose level is essential for overall health, as excess or insufficient levels can lead to various health issues. The hormone insulin, produced by the pancreas, plays a key role in regulating blood glucose. When you eat, especially carbohydrates, the digestive system breaks down the food into glucose, which enters the bloodstream. Insulin facilitates the uptake of glucose by cells, where it can be used as energy or stored for later use. In people without diabetes, the body effectively balances insulin production and glucose utilization to maintain blood glucose levels in a relatively narrow range, usually around 70-100 mg/dL (3.9-5.6 mmol/L) when fasting and below 140 mg/dL (7.8 mmol/L) two hours after a meal. For people living with diabetes, access to affordable treatment, including insulin, is critical to their survival. There is a globally agreed target to halt the rise in diabetes and obesity by 2025. About 422 million people worldwide have diabetes, the majority living in low-and middle-income countries, and 1.5 million deaths are directly attributed to diabetes each year. Both the number of cases and the prevalence of diabetes have been

steadily increasing over the past few decades.
Blood glucose levels play a crucial role in health, particularly for those managing diabetes or other metabolic conditions. There are two main types of imbalances: Type 1 Hyperglycemia, characterized by elevated blood glucose levels, typically stemming from factors like insufficient insulin, stress, poor diet, or medication issues. Symptoms include increased thirst, frequent urination, fatigue, blurred vision, headaches, and slow wound healing. Long-term hyperglycemia can lead to complications such as cardiovascular disease, kidney d image, nerve damage, and vision problems. On the other hand, Type 2 Hypoglycemia occurs when blood glucose levels drop below normal, often due to excessive insulin or medication, missed meals, or increased physical activity. Symptoms of hypoglycemia include shivering, sweating, paleness, rapid heartbeat, confusion, or irritability. Severe hypoglycemia can lead to unconsciousness or seizures, and repeated episodes can weaken the body's ability to detect low blood sugar, raising the risk of severe seizures over time.

1.1 Motivation

This project is driven by the urgent necessity for efficient blood glucose monitoring, especially crucial for individuals navigating conditions such as diabetes. Conventional methods are often invasive and intermittent, posing challenges for continuous management. Seeking non-invasive, continuous monitoring solutions, we aim to relieve the burden on patients and healthcare providers, facilitating proactive interventions and personalized management. Leveraging Machine learning (ML) algorithms holds promise in revolutionizing diagnostics and treatment optimization. By harnessing ML's predictive power, we seek to develop robust models for real-time blood glucose prediction, empowering individuals with actionable insights. Additionally, exploring nature- inspired optimization techniques for ML Hyperparameter Tuning(HT) presents an exciting opportunity to enhance model performance. Drawing inspiration from biological processes, these techniques offer efficient strategies for navigating complex parameter spaces and optimizing model performance. Ultimately, our project aims to intersect healthcare innovation, ML, and nature- inspired optimization to advance personalized healthcare and improve patient outcomes in metabolic health management, particularly for individuals with diabetes.

1.2 Problem Definition

The central challenge tackled in this project revolves around the critical necessity for precise, non-invasive, and ongoing monitoring of blood glucose levels, especially pertinent for individuals managing diabetes or other metabolic conditions. Traditional methods like finger stick testing and continuous glucose monitoring (CGM) devices are often invasive, cumbersome, and lacking in real-time insights into glucose dynamics, potentially inaccessible due to cost constraints. Addressing this, the project aims to develop a ML framework capable of accurately and continuously predicting blood glucose levels based on superficial body features, requiring diverse datasets encompassing physiological data like skin texture, color, facial expressions, and breath composition. Additionally, it

seeks to overcome the challenge of HT in ML algorithms by exploring nature-inspired optimization techniques such as genetic algorithms, particle swarm optimization, enhancing model performance and generalizability efficiently. By achieving these objectives, the project aims to advance metabolic health management, enable proactive interventions, and enhance the quality of life for individuals with diabetes and related conditions.

1.3 Objective of Project

The project has a dual objective: firstly, to develop a ML framework for accurately predicting real-time blood glucose levels based on superficial body features, utilizing diverse physiological datasets to train models to discern correlations between these features and glucose levels, thereby enabling non-invasive and continuous monitoring. Secondly, it aims to tackle the challenge of HT in ML algorithms by exploring nature- inspired optimization techniques like genetic algorithms and particle swarm optimization, aiming to efficiently enhance model performance and generalizability. These endeavors ultimately aim to improve the efficacy of continuous monitoring and management strategies for individuals managing diabetes and related conditions.

1.4 Limitations of Project

While ambitious, this project faces limitations. Firstly, the efficacy of the proposed ML framework for predicting blood glucose levels based on superficial body features hinges on dataset quality and diversity, potentially hindering model generalizability across different populations. Secondly, the success of nature-inspired optimization techniques for HT depends on various factors such as algorithm choice, parameter settings, and computational resources, which may not be universally available. Additionally, predicting blood glucose levels is intricate due to factors like diet, exercise, stress, and medication, potentially leading to discrepancies between predicted and actual levels. Lastly, integrating the ML framework into healthcare systems poses challenges including user interface design, data privacy, regulatory compliance, and scalability, necessitating collaboration across disciplines for effective implementation and long-term sustainability.

CHAPTER-2 LITERATURE SURVEY

2.1 Introduction

This chapter provides an overview of previous research on knowledge sharing. It is important to set the context of the literature review work by first providing an explanation of its specific purpose for this particular project. The main purpose of the literature review work was to survey previous studies. This was in order to scope out the key data collection requirements for the primary research to be conducted.

2.2 Existing System

The advancement of medical technology, coupled with the exponential growth of data, presents an opportune moment to leverage ML algorithms for blood glucose prediction and monitoring. ML techniques have demonstrated remarkable potential in various healthcare applications, promising to revolutionize diagnostics, treatment optimization, and patient care. Rastogi et al. (R. Rastogi and M. Bansal, “Diabetes prediction model using data mining techniques”, Measurement: Journal of the International Measurement Confederation (IMEKO), Measurement: Sensors Volume 25, February 2023), by using the Kaggle data set, the authors found that the LR gives more accurate results i.e., 82.46% as compared to other ML methods. Bhat et al. (S. S. Bhat, V. Selvam, G. A. Ansari, M. D. Ansari, and M H. Rahman, “Prevalence and Early Prediction of Diabetes Using Machine Learning in North Kashmir: A Case Study of District Bandipora”, Computational Intelligence and Neuroscience, Volume 2022, Article ID 2789760) have observed that a RF works best with the highest accuracy of 98% among the others. The dataset used in their research was a clinical dataset collected from clinical diabetic professionals. Patel et al. (K. Patel,

M. Nair and S. Phansekar, "Diabetes Prediction using Machine Learning", International Journal of Scientific & Engineering Research Volume 12, Issue 3, March-2021) have proved that LR gives the highest accuracy of 78% in comparison to other models.
ML involves the development of algorithms and statistical models enabling

computers to perform tasks without explicit instructions, relying on patterns and inference from data. These algorithms work by first collecting relevant data, pre- processing it to clean noise and handle missing values, then selecting or extracting meaningful features. Models are chosen based on the problem at hand, and during training, they learn from labeled or unlabeled data by adjusting internal parameters to minimize prediction errors. Evaluation using separate test data assesses model performance, followed by potential fine-tuning. Once trained, the model can make predictions or decisions on new data. Ultimately, ML algorithms aim to generalize patterns from training data to make accurate predictions or decisions on unseen data, with algorithm selection influenced by factors like data characteristics, dataset size, and the problem domain.

Table 2.1: Analysis of Existing System for past few years

S. No.	**Author(year)**	**Method**	**Techniques**	**Accuracy(in %)**
1.	Rastogi et al. (2023)	SVM, NB Classifier, LR and RF	Oversampling	LR – 82.46%
2.	Febrian et al. (2023)	KNN and NB	ML Algorithm	KNN - 69.37% NB - 71.37%
3.	Bhat et al. (2022)	LR, DT, GB, SVM, RF and MP	Sampling	RF - 98%
4.	Patel et al. (2021)	LR, RF, KNN, DT	ML Algorithm	LR with 78%
5.	Xue et al. (2020)	SVM, NB, Light GBM	ML Algorithm	SVM - 96.54%
6.	Soni et al. (2020)	SVM, RF, KNN, LR, DT, GB	ML Algorithm	RF - 77%
7.	Muhammad et al. (2020)	LR, SVM, KNN, RF, NB, GB	ML Algorithm	RF - 88.76%
8.	Dutta et al. (2019)	KNN, LR, XGB, SVM, RF	Feature Selection, K-fold cross-validation	LR -96%
9.	Sisodia et al. (2018)	SVM, NB, DT	ML Algorithm	NB – 95.2%
10.	Zou et al. (2018)	RF	ML Algorithm	RF - 80.84%

2.3 Disadvantages Of Existing System

The major drawbacks of ML algorithms are oversampling and Feature Selection:

Feature Selection:
A popular method for reducing the dimensionality of datasets is feature selection, which involves choosing a subset of pertinent features. The possible loss of information is a major negative. Eliminating features entails getting rid of information that can be valuable and enhance the functionality of the model. The accuracy or predictive power of the model may decline if significant features are eliminated during the feature selection phase. Furthermore, the curse of dimensionality makes feature selection more difficult in high dimensional datasets. The volume of the feature space

expands exponentially with the amount of features, making it more difficult to identify the ideal subset of features that really improve the performance of the model. Furthermore, if feature selection techniques are not used correctly, bias may be introduced, producing models that are biased towards particular features or miss significant patterns in the data. Moreover, feature selection may fail to take into account interdependencies between features, hence neglecting the collective impact of features on the model's performance and perhaps producing less-than-ideal model results.

Sampling:
Though it's an essential method for handling big datasets and dealing with processing limitations, sampling in ML has a number of disadvantages of its own. Because sampling has intrinsic representativeness difficulties, one significant constraint is the possibility of adding bias or inaccuracies into the model. The diversity and richness of the full dataset may not be fully captured when a subset of data is chosen for training or evaluation, which could result in a skewed or partial representation of the underlying distribution. The outcome of this could be models that are unduly impacted by specific traits or patterns seen in the sampled subset, which could result in poor generalization abilities and inaccurate predictions on fresh data. Furthermore, sampling may result in

the loss of critical information, especially if the sampled group does not fully capture significant patterns or uncommon events. Moreover, scaling issues may arise due to the computational difficulty of sampling strategies, particularly in situations involving big datasets or environments with limited resources. Despite these drawbacks, careful consideration should be paid to validation procedures and sampling strategies in order to lower risks and ensure the development of reliable and accurate ML models. While using traditional ML methods, we may or may not achieve accurate results when comparing different ML algorithms. If we do obtain some level of accuracy, it can vary across different datasets, leading to inconsistent results. Therefore, we employ HT with nature- inspired optimization techniques to identify the ideal parameters for achieving optimal solutions.

2.4 Proposed System

In ML, Hyperparameter Tuning (HT) is a crucial procedure that optimizes the configuration settings, or hyperparameters, that control a model's learning process. Hyperparameters, in contrast to parameters, are fixed values that control the algorithm's behavior and performance. Parameters are learned from the data during training. The aim of HT is to identify the ideal hyperparameter combination that maximizes the model's performance on a given task or dataset. In this procedure, a predetermined set of hyperparameter values or ranges are often searched through methodically, an evaluation metric is selected to assess the model's performance, and the hyperparameters are repeatedly refined based on the performance that is seen. For HT, a variety of optimization methods can be used, such as grid search, random search, Bayesian optimization, and more sophisticated algorithms like genetic algorithms. In order to guarantee that ML models function as best they can in practical situations, HT is crucial for enhancing model robustness, accuracy, and generalization capacity.
HT is closely linked to Nature Inspired Optimization (NIO) techniques in the context of ML. In ML, hyperparameters are configuration settings that are not learned from the data but rather are specified before the training process begins. Examples of hyperparameters include learning rate, regularization strength, number of hidden layers in a neural network, or the depth of a Decision Tree(DT). The link between HT and NIO lies in their shared goal of finding the best configuration within a search space. HT aims

to identify the optimal set of hyperparameters for a ML model to achieve the best performance on a given task. This involves exploring a multidimensional space of hyperparameters and evaluating the model's performance using a chosen metric, such as accuracy or loss. NIO techniques provide powerful tools for efficiently searching through the hyperparameter space. These algorithms leverage principles

from natural systems, such as natural selection, genetic variation, or collective behavior, to guide the search towards promising regions of the solution space. Among the array of nature-inspired optimization techniques, the Firefly algorithm (FA) stands out as one method in which we harness the principles of natural phenomena. This algorithm, like others of its kind, draws inspiration from the behaviors observed in nature to address optimization challenges. By mimicking the behavior of fireflies, it navigates through solution spaces seeking optimal outcomes. The FA, alongside its counterparts, contributes to solving complex problems efficiently and effectively across diverse domains such as engineering, computer science, finance, and biology.

2.5 Conclusion

While using traditional ML methods, we may or may not achieve accurate results when comparing different ML algorithms. If we do obtain some level of accuracy, it can vary across different datasets, leading to inconsistent results. Therefore, we employ HT with nature-inspired optimization techniques to identify the ideal parameters for achieving optimal solutions.

CHAPTER-3 SYSTEM ANALYSIS AND DESIGN

3.1 Introduction

System analysis is an important activity that takes place when we are building a new information system or changing existing ones. The Analysis is used to gain an understanding of an existing and what is required for it. At the conclusion of the analysis, there is a system description and asset of requirements for a new system. If there is no existing system, the analysis defines only the requirements.

This phase is a detailed appraisal of the system. It also includes the system's problems what the end-users required of any new or changed system. After this phase, analyst should have complied with both the detailed operation of the system what is required for the new system. The appraisal includes fining how the system works.

Thus, a rule, system analysis is a difficult but rewarding job. There are many constraints to be complied with in this work and people to be complied with in this work and people to satisfy. But there is the reward of seeing a new system does its job perfectly.

3.2 Software Requirement Specification

Requirements specification is the starting point of the software development activity. The Requirements specification states the goals and objectives of the software, describing it in the context of the computer-based system. The requirements specification includes an information description, functional description, non-functional description. Further classified into three types. Those are:

3.2.1 Software Requirement

The functional requirements or the overall description documents include the product perspective and features, operating system and operating environment, graphics requirements, design constraints, and user documentation.

The appropriation of requirements and implementation constraints gives the general

overview of the project in regards to what the areas of strength and deficit are and how to tackle them.

- Python idle 3.8 version (or)
- Anaconda 3.8 (or)
- Jupiter (or)
- Google collab
- Operating System (windows or any higher version of windows)

http://doi.org/10.36893/JNAO.2024.V15I1N01.283-310

3.2.2 Hardware Requirement

Minimum hardware requirements depend very much on the particular software a given enthought Python / Canopy / VS Code user develops. Applications that need to store large arrays/objects in memory will require more RAM, whereas applications that need to perform numerous calculations or tasks more quickly will require a faster processor.

- PROCESSOR: Intel core i3
- RAM: 4GB
- HARD DISK: 2TB
- Input device: Standard Keyboard and Mouse
- Output device: High-resolution monitor

3.2.3 User Requirement

The user requirement document can be used as a guideline to planning cost timetable, milestone, testing etc., the explicit nature of user requirements document allows customer to show it to various stakeholders to make sure all necessary described.

3.3 Algorithms

3.3.1 Decision Tree

Decision Tree (DT) algorithm is a popular supervised learning algorithm used for both classification and regression tasks in ML. It's a versatile algorithm known for its simplicity, interpretability, and ability to handle both numerical and categorical data. The algorithm works by recursively partitioning the feature space into smaller regions,

eventually forming a tree-like structure composed of decision nodes and leaf nodes.

The following algorithm simplifies the working of a DT:

- Step I: Start the decision tree with a root node, X. Here, X contains the complete dataset.
- Step II: Determine the best attribute in dataset X to split it using the ‘attribute selection measure (ASM).
- Step III: Divide X into subsets containing possible values for the best attributes.
- Step IV: Generate a tree node that contains the best attribute.
- Step V: Make new decision trees recursively by using the subsets of the dataset X created in step III. Continue the process until you reach a point where you cannot further classify the nodes. Call the final node a leaf node.

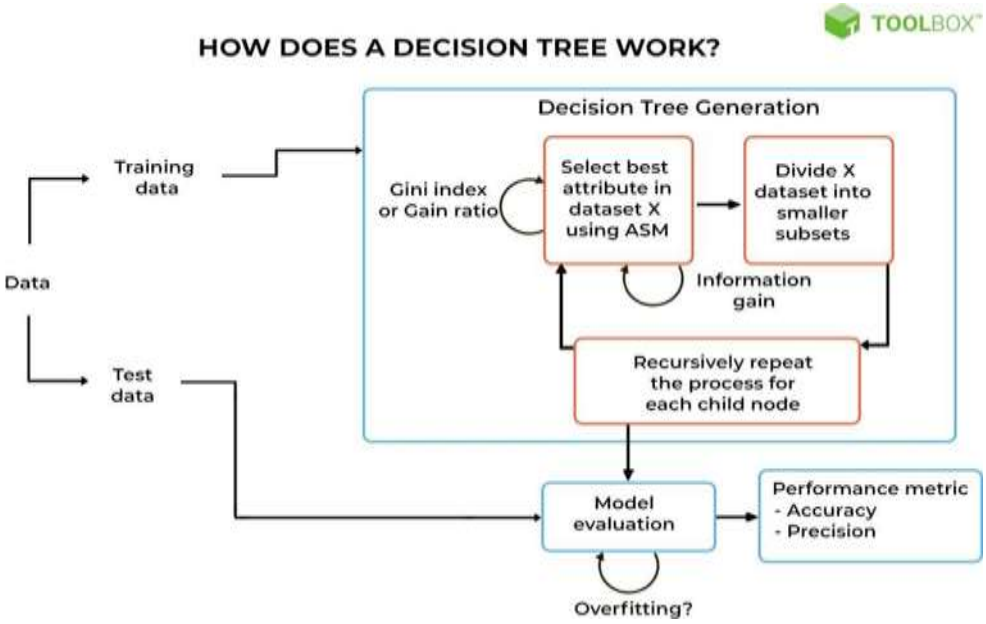

Figure 3.1: Working flow of the Decision Tree Algorithm

http://doi.org/10.36893/JNAO.2024.V15I1N01.283-310

In the above algorithm, the attribute selection measure refers to a type of heuristic used for selecting the splitting criterion in a way that best separates a given dataset (X) into individual subsets. In other words, it determines how the datasets or subsets at a given node are to be split.
DT algorithm is intuitive and easy to interpret, making it a valuable tool for understanding and explaining the decision-making process in ML models. However, it's susceptible to overfitting, especially with complex datasets, which can be mitigated using techniques like pruning or ensemble methods like Random Forest.

3.3.2 Random Forest

Random Forest (RF) is a popular ensemble learning algorithm used in ML for both classification and regression tasks. It works by constructing multiple decision trees during training and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Below is a simplified explanation of the RF algorithm and its step-by-step process:
Random Forest Algorithm

- Random Sampling: Randomly select 'n' samples from the dataset with replacement (bootstrap samples).
- Feature Selection: Randomly select 'm' features from the dataset.
- Decision Tree Construction: Construct a decision tree based on the selected samples and features. At each node:

Choose the best split among the 'm' features.
Split the node into child nodes based on the selected split.

- Repeat Steps 1-3: Repeat steps 1-3 'k' times to create 'k' decision trees.
- Voting: For classification tasks, each tree "votes" for the class of the input sample. For regression tasks, each tree provides a prediction.
- Aggregate Results: Aggregate the votes or predictions from all trees to make a final decision:

For classification, use majority voting to select the class with the most votes.

For regression, take the average of all predictions.

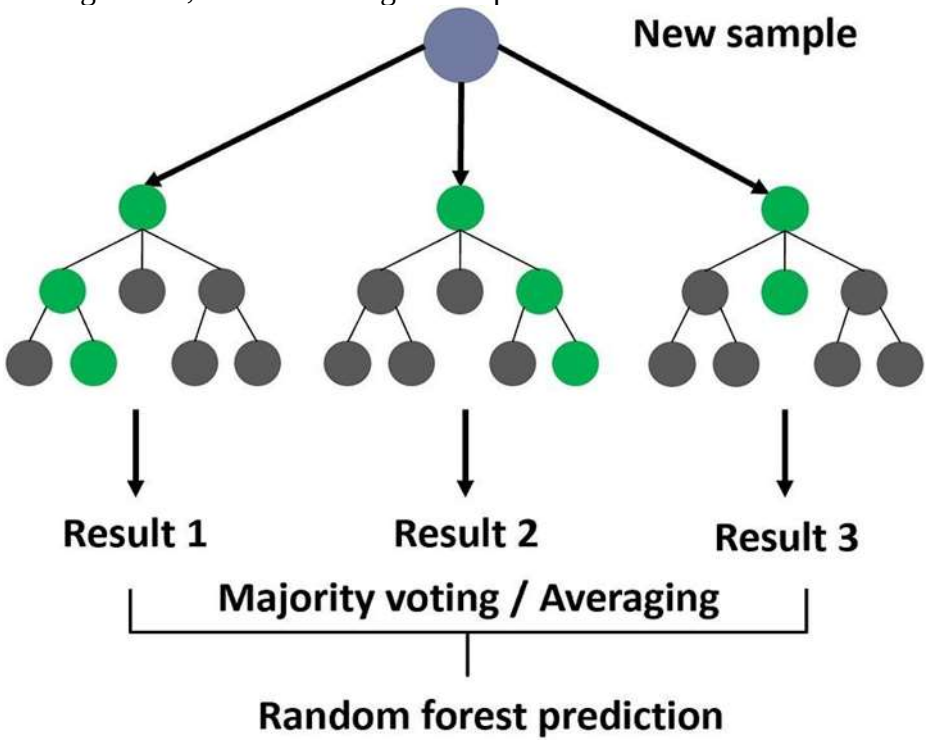

Figure 3.2: Working flow of the Random Forest Algorithm

The key advantages of RF include its ability to handle high-dimensional data, feature importance estimation, and resistance to overfitting. Additionally, it is less sensitive to noisy data compared to individual DTs. RF is a versatile algorithm suitable for a wide range of applications in ML.

3.3.3 Logistic Regression

Logistic Regression (LR) is a statistical method used for binary classification tasks, where the outcome variable is categorical with two possible values. It's a fundamental algorithm in ML, particularly in scenarios where the relationship between the independent variables and the probability of a particular outcome needs to be modeled. Logistic Regression algorithm:

- Initialize the Parameters: Start by initializing the weights and the bias. These are the parameters that the algorithm will adjust during training to fit the data.

Calculate the Linear Combination: For each data point, calculate the linear combination of the input features with the weights. Add the bias term to this result.

- Apply the Sigmoid Function: Pass the linear combination through the sigmoid function. This function transforms the output into a value between 0 and 1, which can be interpreted as the probability of the input belonging to the positive class.

- Calculate the Loss: Compare the predicted probabilities with the actual labels to calculate the loss. One common loss function for LR is the binary cross-entropy loss.

- Update the Parameters: Use gradient descent (or another optimization algorithm) to update the weights and bias in the direction that reduces the loss. This step involves calculating the gradient of the loss function with respect to the parameters.

- Repeat: Repeat steps 2-5 for a fixed number of iterations or until the loss converges to a satisfactory level.

- Prediction: Once the model is trained, use it to predict the probability of new data points belonging to the positive class. A threshold can be applied to these probabilities to make binary predictions.

- Evaluation: Evaluate the performance of the model using metrics such as accuracy, precision, recall, or F1 score on a separate validation or test set.

- Iterate and Improve: Iterate on the model by adjusting hyperparameters, feature engineering, or trying different optimization algorithms to improve performance.

http://doi.org/10.36893/JNAO.2024.V15I1N01.283-310

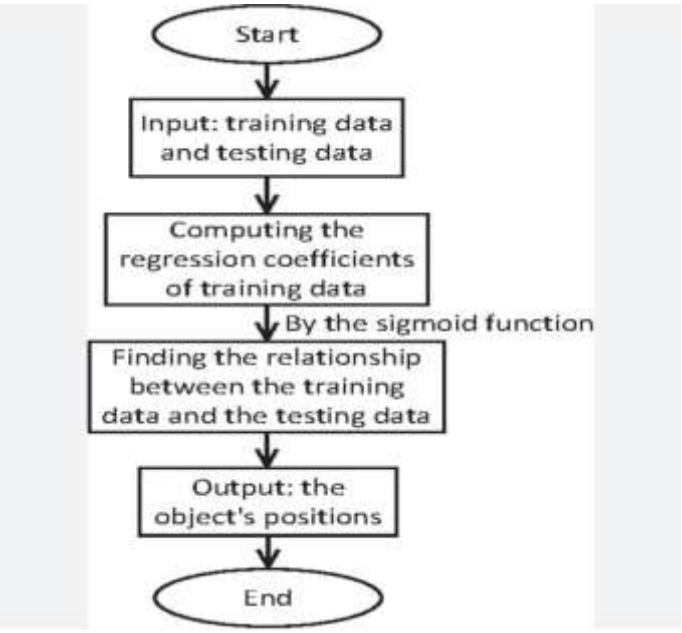

Figure 3.3: Working flow of the Logistic Regression Algorithm

Nonetheless, LR serves as a valuable tool in predictive modeling, often serving as a benchmark against which more sophisticated algorithms are compared. As the field of ML continues to evolve, LR remains a cornerstone, offering a solid foundation upon which more advanced techniques are built.

3.3.4 K-Nearest Neighbors (KNN) Algorithm

The k-Nearest Neighbors (KNN) algorithm is a simple yet effective method used for classification and regression tasks in ML. It operates by storing all available cases and classifying new cases based on their similarity to existing cases. In classification, the algorithm assigns the most common class among the k nearest neighbors of a data point, while in regression, it calculates the average of the k nearest neighbors' values. KNN's performance heavily relies on the choice of the distance metric and the number of neighbors (k).

K-Nearest Neighbors (KNN) algorithm:

- Choose the Number of Neighbours (K): Decide on the number of neighbours to consider when making predictions. This is a hyperparameter that needs to be specified upfront.
- Select a Distance Metric: Choose a distance metric (e.g., Euclidean distance, Manhattan distance) to measure the distance between data points. This metric determines how "close" two points are in the feature space.
- Prepare the Training Data: Store the feature values and corresponding class labels of the training dataset.
- Input New Data: Receive a new data point for which you want to make a prediction.
- Calculate Distances: Calculate the distance between the new data point and all points in the training dataset using the chosen distance metric.
- Find Nearest Neighbours: Select the K data points from the training dataset that are closest to the new data point based on the calculated distances.
- Majority Vote: For classification, determine the class label by taking a majority vote among the K nearest neighbours. The class with the highest count among the

neighbours is assigned to the new data point.

http://doi.org/10.36893/JNAO.2024.V15I1N01.283-310

- Weighted Vote (Optional): Optionally, you can assign weights to the neighbours based on their distance to the new data point. Closer neighbours can be given higher weights, so their influence on the prediction is greater.
- Output Prediction: For regression, the predicted value can be the average (or weighted average) of the target values of the K nearest neighbours. For classification, the predicted class label is the one determined by the majority vote.
- Repeat: Repeat steps 4-9 for each new data point you want to classify or predict.
- Evaluation: Evaluate the performance of the model using metrics such as accuracy, precision, recall, or F1 score on a separate validation or test set to assess its generalization ability.
- Iterate and Improve: Iterate on the model by adjusting hyperparameters (e.g., K, distance metric), feature engineering, or trying different strategies to handle ties or weights to improve performances.

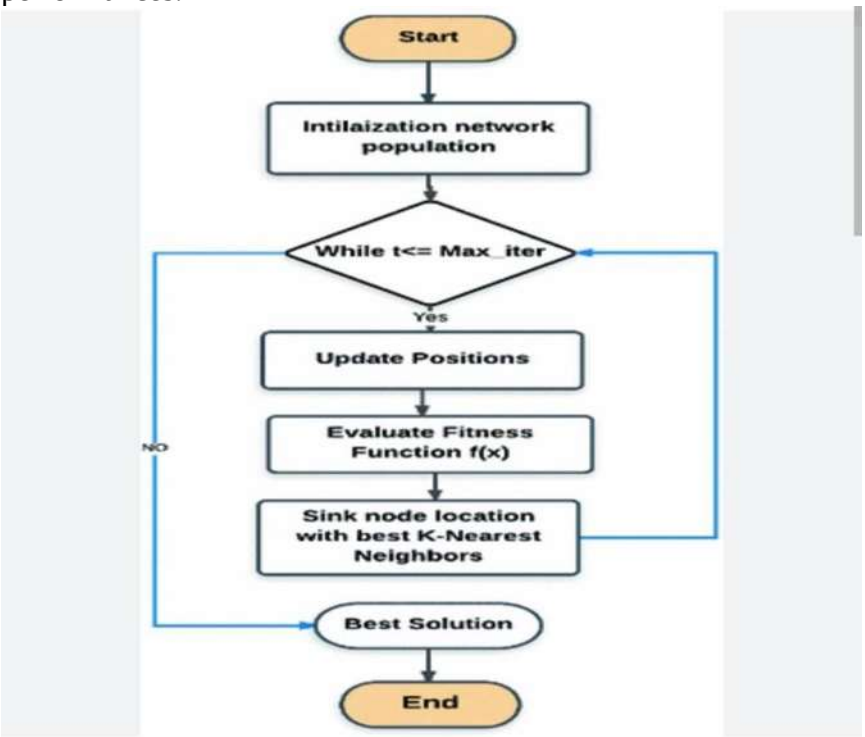

Figure 3.4: Working flow of the Logistic Regression Algorithm

Despite its simplicity, KNN can be computationally expensive for large datasets,

especially in high-dimensional spaces, and it does not learn explicit models from the data. However, its ease of implementation and interpretability make it a popular choice, particularly for small to medium- sized datasets or as a baseline model for comparison in more complex tasks.

3.4 Conclusion

We presented an analysis and detailed explanation of the algorithm. We have analyzed the working of each algorithm in and efficient manner. Now we are moving to the methodology part.

http://doi.org/10.36893/JNAO.2024.V15I1N01.283-310

CHAPTER-4 METHODOLOGY

4.1 Hyperparameter Tuning

Hyperparameter tuning (HT) is the process of optimizing the settings of a ML model that are not learned from the data but are predefined by the practitioner. These settings, known as hyperparameters, profoundly influence the model's performance and generalization ability. HT involves systematically exploring different combinations of hyperparameters to find the configuration that maximizes the model's performance on a validation dataset. Techniques such as grid search, random search, or Bayesian optimization are commonly used to efficiently search the hyperparameter space. By fine-tuning these settings, practitioners aim to improve the model's accuracy, robustness, and ability to generalize to unseen data.

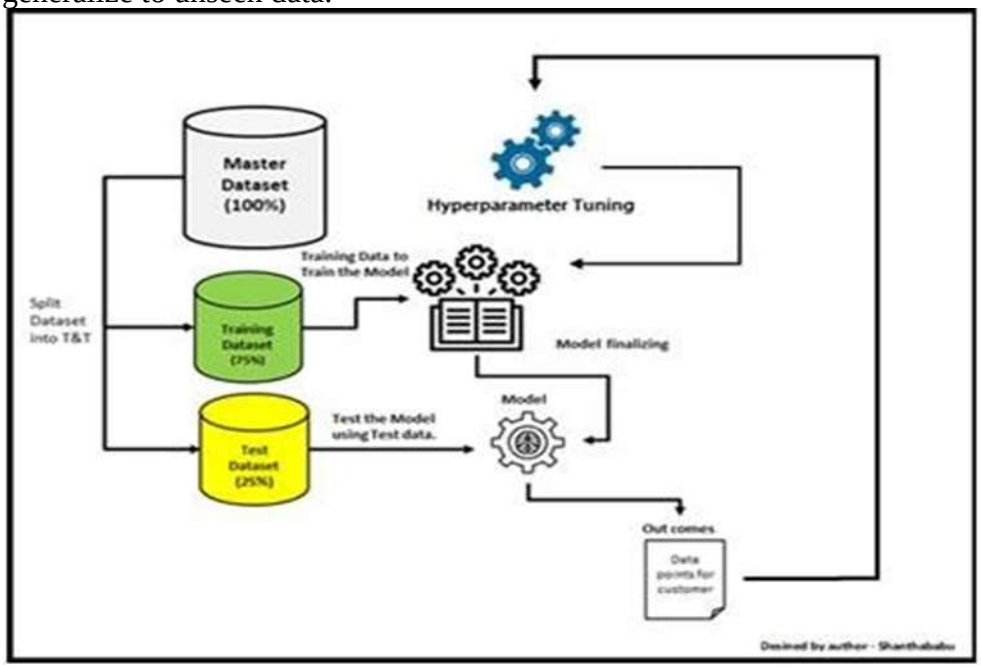

Figure 4.1: Working procedure of Hyperparameter Tuning

Let’s break down the hyperparameter tuning working procedure step by step:

1. Master Dataset

Initial Dataset: Begin with a master dataset containing both features (inputs) and the target variable (output) for your machine learning task.

2. Splitting into Training and Testing datasets

Data Splitting: Divide the master dataset into two subsets – a training dataset and a testing dataset.

Typical Split: Common splits include 80% for training and 20% for testing, but the exact ratio can vary.

3. Model Training on Training Datasets

Select Model: Choose a machine learning model for your task (e.g., DT, RF, LR, and KNN).Hyperparameter Initialization: Set initial values for hyperparameters (e.g., learning rate, depth of the tree).

Training: Train the model on the training dataset using the chosen hyperparameters.

4. Model Evaluation on Testing Dataset

Testing Dataset: Use the testing dataset, which the model has not seen during training, to evaluate its performance.

Metric Evaluation: Assess the model's performance using relevant metrics (accuracy, precision, recall, F1 score).

5. Initial Model Outcomes

Outcome Analysis: Analyze the initial outcomes and performance of the model on the testing dataset.

http://doi.org/10.36893/JNAO.2024.V15I1N01.283-310

Identify Issues: Identify issues such as overfitting or underfitting based on the model's performance.

6. Hyperparameter Tuning

Define Search Space: Define a search space for hyperparameters (e.g., ranges for learning rates, tree depths).

Choose Optimization Method: Select an optimization method (e.g., grid search, random search, nature-inspired optimization like Firefly Algorithm).

Iterative Process: Iteratively adjust hyperparameters, train the model on the training dataset, and evaluate on the testing dataset.

7. Finalizing Model

Optimal Hyperparameters: Identify the set of hyperparameters that result in the best performance on the testing dataset.

Train Final Model: Train the final model using the optimal hyperparameters on the entire training dataset.

8. Model Evaluation with Optimal Hyperparameters:

Testing Dataset Evaluation: Evaluate the model with optimal hyperparameters on the testing dataset to ensure to ensure generalization.

9. Final Outcomes and Model Deployment:

Performance Analysis: Analyze the final performance metricsto ensure improvement.

Deployment: If satisfied with the model's performance, deploy it for predictions on new, unseen data.

10. Documentation and Reporting:

Record Hyperparameters: Document the final set of hyperparameters used in the optimized model.

Report Outcomes: Summarize the performance improvements achieved through hyperparameter tuning.

4.2 Hyperparameter Tuning With Nature-Inspired Optimization

HT is closely linked to NIO techniques in the context of ML. In ML, hyperparameters are configuration settings that are not learned from the data but rather are specified before the training process begins. Examples of hyperparameters include learning rate, regularization strength, number of hidden layers in a neural network, or the depth of a DT. The link between HT and NIO lies in their shared goal of finding the best configuration within a search space. HT aims to identify the optimal set of hyperparameters for a ML model to achieve the best performance on a given task. This involves exploring a multidimensional space of hyperparameters and evaluating the model's performance using a chosen metric, such as accuracy or loss.

NIO techniques provide powerful tools for efficiently searching through the hyperparameter space. These algorithms leverage principles from natural systems, such as natural selection, genetic variation, or collective behavior, to guide the search towards promising regions of the solution space.

4.3 Nature-Inspired Optimization Techniques

Nature-inspired optimization techniques are a class of algorithms that draw inspiration from natural phenomena or processes to solve optimization problems. These

algorithms are designed to mimic the behavior of biological systems, physical processes, or ecological interactions in nature. Examples include genetic algorithms, particle swarm optimization, simulated annealing, ant colony optimization, and evolutionary strategies. These techniques often leverage concepts such as natural selection, mutation, reproduction, swarm intelligence, or the dynamics of physical systems to iteratively search for optimal solutions to complex problems. Nature-inspired optimization algorithms are widely used in various fields, including engineering, computer science,

finance, and biology, due to their ability to efficiently explore large solution spaces and find high-quality solutions.
Among the array of nature-inspired optimization techniques, the FA stands out as one method in which we harness the principles of natural phenomena. This algorithm, like others of its kind, draws inspiration from the behaviors observed in nature to address optimization challenges. By mimicking the behavior of fireflies, it navigates through solution spaces seeking optimal outcomes. The FA, alongside its counterparts, contributes to solving complex problems efficiently and effectively across diverse domains such as engineering, computer science, finance, and biology. Now let's talk about the FA, one of the techniques for NIO.

4.4 Firefly Optimization Algorithm

In the domain of ML, the efficacy of models hinges on selecting optimal hyperparameters, a task often challenging due to the intricate and high-dimensional search space involved. Firefly Optimization Algorithm (FA), inspired by the flashing behavior of fireflies in attracting mates, has emerged as a promising optimization technique. It was proposed by Xin-She Yang in 2008 and is particularly useful for solving continuous optimization problems. Drawing on the principles of attraction based on brightness and distance among fireflies, FA iteratively adjusts solutions' positions in the search space, akin to optimizing hyperparameters. Notably, FA offers robustness in navigating complex spaces, inherent parallelizability for scalability, and ease of implementation without requiring intricate mathematical formulations. Integrating FA into HT pipelines holds the promise of revolutionizing model optimization, potentially enhancing efficiency, reducing computational overhead, and expediting development cycles. Through experimentation, we aim to compare FA's performance with traditional

techniques, contributing to advancing optimization methods in ML and fostering the adoption of nature-inspired approaches in practical settings. Here's a step-by-step overview of the FA:

1. Initialization:

Initialize the population of fireflies with random solutions. Let $\boldsymbol{N}$ denote the number of fireflies in the population, and $\boldsymbol{D}$ denotes the dimensionality of the search space. Each firefly $\boldsymbol{i}$ is represented by its position $\boldsymbol{Xi}$ in the D-dimensional search space.

2. Evaluation:

Evaluate the brightness Ii of each firefly based on its fitness value, which is determined by the objective function ($\boldsymbol{Xi}$). Higher fitness values yield brighter fireflies.
Mathematically, the brightness $\boldsymbol{Ii}$ of firefly $\boldsymbol{i}$ can be computed as:
$\boldsymbol{Ii} = \boldsymbol{f}(\boldsymbol{Xi})$

3. Attraction:

Calculate the attractiveness **Aij** of each firefly **i** towards every other firefly j based on their brightness and distance.
The attractiveness $\boldsymbol{Aij}$ is inversely proportional to the square of the distance between fireflies i and j and directly proportional to the brightness **Ij** of firefly j.
The attractiveness of **Aij** is given by:
$\boldsymbol{Aij} = \mathrm{Q}\boldsymbol{e} - \boldsymbol{r2}$

Where:
Q is the attractiveness scaling coefficient
is the absorption coefficient.
$\boldsymbol{rij}$ is the Euclidean distance between fireflies **i** and **j**.

http://doi.org/10.36893/JNAO.2024.V15I1N01.283-310

4. Movement:

Update the position of each firefly based on the attractiveness of other fireflies. Firefly i moves towards the brighter firefly j based on the attractiveness $\boldsymbol{Aij}$ and a randomization factor α.

The updated positio$\boldsymbol{i}$n $\boldsymbol{X'}$ of Firefly i is given by:

$\mathbf{X'i} = \mathbf{X\,i} + \beta\mathbf{e} - \mathbf{yr2}\,.\,(\mathbf{Xj} - \mathbf{Xi}) + \alpha.\,\mathrm{e}$

5. Termination Criterion:

Define a termination criterion, such as reaching a maximum number of iterations or achieving a desired level of convergence.

The termination criterion in the FA typically involves defining a stopping condition based on a certain number of iterations or reaching a desired level of convergence. One commonly used termination criterion is to halt the algorithm after a specified maximum number of iterations **Tmax** has been reached.

Mathematically, the termination criterion can be defined as follows:

$\mathbf{t} > \mathbf{Tmax}$

where:

- **t** is the current iteration number.
- **Tmax** is the maximum number of iterations allowed.

Once the current iteration number exceeds the maximum allowed number of iterations, the algorithm terminates, and the best solutions found so far are returned. This termination criterion ensures that the algorithm does not continue indefinitely and provides a predefined stopping point for the optimization process.

6. Iterations:

Repeat steps 2-4 until the termination criterion is met.

7. Solution Retrieval:

Once the algorithm converges or reaches the termination criterion, retrieve the best solution or solutions found during the optimization process.

8. Post-processing:

Perform any necessary post-processing tasks, such as fine-tuning the parameters of the best solution or analysing the convergence behaviour of the algorithm.

In essence, the FA continuously adjusts the positions of fireflies, influenced by their mutual appeal, with the goal of enhancing brightness (fitness) while reducing distance. This iterative process enables the algorithm to navigate the search space effectively, ultimately uncovering optimal solutions for the given

optimization challenge.

http://doi.org/10.36893/JNAO.2024.V15I1N01.283-310

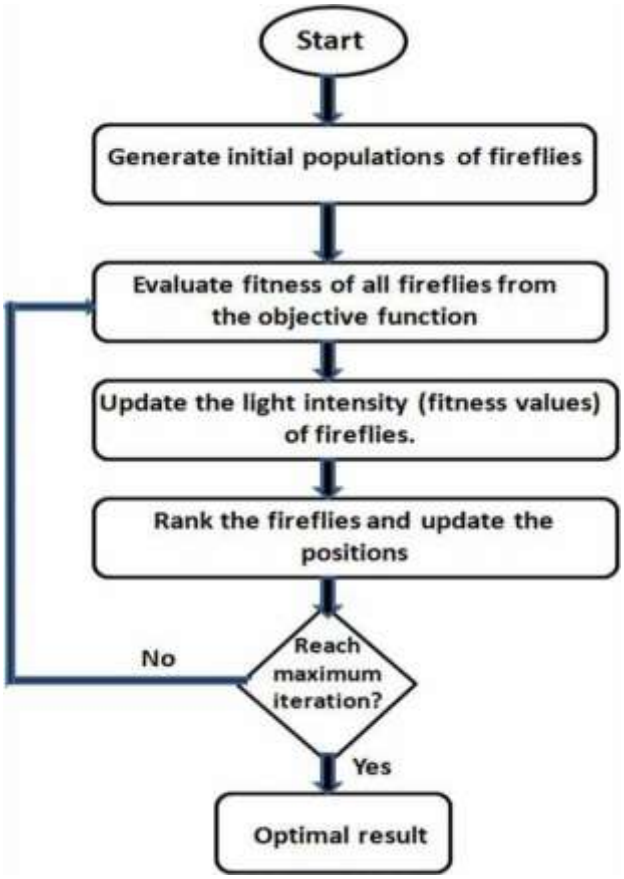

Figure 4.2: Flowchart Of Firefly Algorithm

4.5 Sample Code

4.5.1 Decision Tree

Training part:

```
from sklearn_nature_inspired_algorithms.model_selection import NatureInspiredSearchCV
from sklearn.tree import DecisionTreeClassifier param_grid = {
'max_depth': range(2, 40, 2),
'min_samples_split': range(2, 20, 2), 'max_features': [ "sqrt", "log2"],
}
clf = DecisionTreeClassifier(random_state=42) nia_search = NatureInspiredSearchCV(
clf, param_grid, algorithm='fa',
population_size=5, max_n_gen=10, max_stagnating_gen=10, runs=3,
random_state=None, # or any number if you want same results on each run
)
nia_search.fit(X_train, y_train)
```

Output:

```
Fitting at most 342 candidates
Optimization finished, 334 candidates were fitted

NatureInspiredSearchCV
estimator: DecisionTreeClassifier
        DecisionTreeClassifier(random_state=42)
```

Testing part :

```
from sklearn_nature_inspired_algorithms.model_selection import NatureInspiredSearchCV
from sklearn.tree import DecisionTreeClassifier param_grid = {
'max_depth': range(2, 40, 2),
'min_samples_split': range(2, 20, 2), 'max_features': [ "sqrt", "log2"],
}
clf = DecisionTreeClassifier(random_state=42) nia_search = NatureInspiredSearchCV(
clf, param_grid, algorithm='fa',
population_size=5, max_n_gen=10, max_stagnating_gen=10, runs=3,
random_state=None, # or any number if you want same results on each run
)
nia_search.fit(X_test, y_test)
```

Output:

```
Fitting at most 342 candidates
Optimization finished, 312 candidates were fitted

NatureInspiredSearchCV
estimator: DecisionTreeClassifier
```

4.5.2 Random Forest

Training part:

```
from sklearn_nature_inspired_algorithms.model_selection import NatureInspiredSearchCV
from sklearn.ensemble import RandomForestClassifier param_grid = {
'n_estimators': range(20, 100, 20),
'max_depth': range(2, 40, 2),
'min_samples_split': range(2, 20, 2), 'max_features': [ 'sqrt', 'log2']
}
clf = RandomForestClassifier(random_state=42) nia_search = NatureInspiredSearchCV(
clf, param_grid, cv=3, verbose=0, algorithm='fa',
population_size=5, max_n_gen=10, max_stagnating_gen=10, runs=3,
n_jobs=-1, scoring='f1_macro', random_state=42)

nia_search.fit(X_train, y_train)
```

Output:

```
NatureInspiredSearchCV
```

estimator: RandomForestClassifier
Testing part:

```
from sklearn_nature_inspired_algorithms.model_selection import NatureInspiredSearchCV
from sklearn.ensemble import RandomForestClassifier param_grid = {
'n_estimators': range(20, 100, 20),
'max_depth': range(2, 40, 2),
'min_samples_split': range(2, 20, 2), 'max_features': [ 'sqrt', 'log2']
}
clf = RandomForestClassifier(random_state=42) nia_search = NatureInspiredSearchCV(
clf, param_grid, cv=3, verbose=0, algorithm='fa',
population_size=5, max_n_gen=10, max_stagnating_gen=10, runs=3,
n_jobs=-1,
)
nia_search.fit(X_test, y_test)
```

Output:

NatureInspiredSearchCV
estimator: RandomForestClassifier

RandomForestClassifier(random_state=42)

4.5.3 K- Nearest Neighbours Training part:

```
from sklearn_nature_inspired_algorithms.model_selection import NatureInspiredSearchCV
from sklearn.neighbors import KNeighborsClassifier param_grid = {
'n_neighbors': range(1, 21), # Number of neighbors
'weights': ['uniform', 'distance'], # Weight function used in prediction 'p': [1, 2] # Power parameter for
Minkowski distance
}
clf = KNeighborsClassifier() nia_search = NatureInspiredSearchCV( clf,
param_grid,
algorithm='fa', # hybrid bat algorithm population_size=50, max_n_gen=100, max_stagnating_gen=10,
runs=3,
random_state=None, # or any number if you want the same results on each run
)
nia_search.fit(X_train, y_train) Output:
```

Fitting at most 80 candidates
Optimization finished, 80 candidates were fitted

NatureInspiredSearchCV
estimator: KNeighborsClassifier

Testing part:

```
from sklearn_nature_inspired_algorithms.model_selection import NatureInspiredSearchCV
```

```
from sklearn.neighbors import KNeighborsClassifier from sklearn.metrics import classification_report
param_grid = {
'n_neighbors': range(1, 21), # Number of neighbors
'weights': ['uniform', 'distance'], # Weight function used in prediction 'p': [1, 2] # Power parameter for
Minkowski distance
}
clf = KNeighborsClassifier() nia_search = NatureInspiredSearchCV(
clf, param_grid,
algorithm='fa', # hybrid bat algorithm population_size=50, max_n_gen=100, max_stagnating_gen=10,
runs=3,
random_state=None, # or any number if you want the same results on each run
)
nia_search.fit(X_test, y_test)
```

Output:

```
Fitting at most 80 candidates
Optimization finished, 80 candidates were fitted

NatureInspiredSearchCV
estimator: KNeighborsClassifier
```

4.5.4 Logistic Regression Training Part:

```
from sklearn.linear_model import LogisticRegression
from sklearn_nature_inspired_algorithms.model_selection import NatureInspiredSearchCV

param_grid = { 'penalty': ['l1', 'l2'],
'C': [0.001, 0.01, 0.1, 1, 10, 100],
'solver': ['liblinear', 'saga'] # 'liblinear' for small datasets, 'saga' for large datasets
}
clf = LogisticRegression(random_state=42) nia_search = NatureInspiredSearchCV(
clf, param_grid, algorithm='fa',
population_size=50, max_n_gen=100, max_stagnating_gen=10, runs=3,
random_state=None # or any number if you want the same results on each run
)
nia_search.fit(X_train, y_train)
```

Output:

```
Fitting at most 24 candidates
Optimization finished, 24 candidates were fitted
NatureInspiredSearchCV
estimator: LogisticRegression
```

Testing part:

```
from sklearn.linear_model import LogisticRegression
from sklearn_nature_inspired_algorithms.model_selection import NatureInspiredSearchCV
```

```
# Define the parameter grid for hyperparameter tuning param_grid = {
'penalty': ['l1', 'l2'],
'C': [0.001, 0.01, 0.1, 1, 10, 100],

'solver': ['liblinear', 'saga'] # 'liblinear' for small datasets, 'saga' for large datasets
}
# Initialize Logistic Regression Classifier clf = LogisticRegression(random_state=42)
# Perform hyperparameter tuning using NatureInspiredSearchCV nia_search = NatureInspiredSearchCV(
clf, param_grid, algorithm='fa',
population_size=50, max_n_gen=100, max_stagnating_gen=10, runs=3,
random_state=None # or any number if you want the same results on each run
)
nia_search.fit(X_test, y_test)
```

Output:

Fitting at most 24 candidates

Optimization finished, 24 candidates were fitted

NatureInspiredSearchCV
estimator: LogisticRegression

CHAPTER-5

SYSTEM TESTING AND RESULTS

5.1 Introduction

In the context of software development, testing refers to the process of evaluating a software application or system to identify any discrepancies between expected and actual results. Testing ensures that the software meets its requirements, functions correctly, and is free of defects before it is released to the end users. There are different types of testing, such as unit testing, integration testing, system testing, and acceptance testing, each serving a specific purpose in the software development lifecycle.

5.1.1 System Testing

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

http://doi.org/10.36893/JNAO.2024.V15I1N01.283-310

5.1.2 Test cases

Table 5.2 : Testcase table for Diabetic prediction

Test No.	Test Case	Expected output	Actual output	Status
1.	Heart Rate =100 Temperature =96.752 Sweating (1/0) = 0 Shivering (1/0)= 1	1	1	Pass
2.	Heart Rate =100 Temperature =97.812 Sweating (1/0) = 1 Shivering (1/0)= 0	0	0	Pass
3.	Heart Rate =85 Temperature =97.747 Sweating (1/0) = 0 Shivering (1/0)= 1	1	1	Pass
4.	Heart Rate =79 Temperature =97.53212 Sweating (1/0) = 0 Shivering (1/0)= 0	0	0	Pass

5.2 Testing Strategies

Testing strategies are approaches or plans that define how testing will be conducted. These strategies outline the scope, objectives, resources, and schedule for testing activities. They help ensure that testing is thorough, efficient, and effective. Some common testing strategies include:

5.2.1 White Box Testing

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level. It is performed by software developers.

5.2.2 Black Box Testing

Black Box Testing is testing the functionality of an application without knowing the details of its implementation including internal program structure, data structures etc. A method of software testing that verifies the functionality of an application without having specific knowledge of the application's code/internal structure. Tests are based on requirements and functionality. Test cases for black box testing are created based on the requirement specifications. Therefore, it is also called as specification-

based testing. This is one type of testing in which the software under test is treated as a black box, in which you cannot “see” in to it.

The test provides inputs and responds to outputs without considering how the software works.

5.2.3 Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail. Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.
- Features to be tested.
- Verify that the entries are of the correct format.
- No duplicate entries should be allowed.
- All links should take the user to the correct page.

5.2.4 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. Components in a software system or – one step up – software applications at the company level – interact without error.

Test Results:

All the test cases mentioned above passed successfully. No defects encountered.

5.2.4.1 Alpha Testing

This is one type of testing a software product or system conducted at the developer's site. Usually, it is performed by the end users.

5.2.4.2 Beta Testing

Final testing before releasing application for commercial purpose. It is typically done by end- users or others

5.2.4.3 Performance Testing

Functional testing conducted to evaluate the compliance of a system or component with specified performance requirements. It is usually conducted by the performance engineer.

5.3 Results

This section delves into the performance of the models using classification methods, presenting the results obtained from each algorithm. Evaluation of performance is conducted through the representation of confusion matrices. Based on the confusion matrix of each model, a series of equations are derived to determine the classification outcomes.

http://doi.org/10.36893/JNAO.2024.V15I1N01.283-310

5.3.1 Performance Evaluation

This subsection primarily emphasizes the confusion matrix as the fundamental visualization method for evaluating ML models. The confusion matrix displays the count of resulting outputs, which should collectively add up to the total testing set.

Table 5.1 illustrates the confusion matrix for each algorithm. True positives indicate that the predicted values are positive and match the actual positive cases. True negatives represent the predicted negative values that correspond to the actual negative cases. False negatives occur when predicted values are negative, but they should be positive. Conversely, false positives are predicted positive values that should be negative. The KNN, DT, RF and LR algorithms demonstrated no negative predictions of true cases, suggesting accurate prediction of all diabetic cases.

Figure 5.1, Figure 5.2, Figure 5.3, and Figure 5.4 shows the confusion matrix tables of KNN, DT, RF and LR.

TABLE 5.2: Confusion matrix values from test portion

ML ALGORITHM	TP	FN	FP	TN
KNN	3319	2	32	41
DT	4159	0	30	54
RF	3328	0	13	53
LR	5000	0	76	15

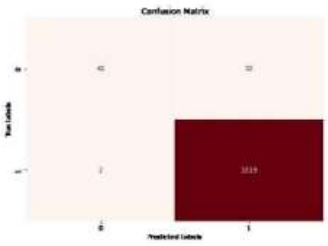

Figure 5.1: KNN Confusion Matrix

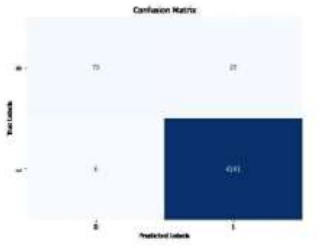

Figure 5.2: DT Confusion Matrix

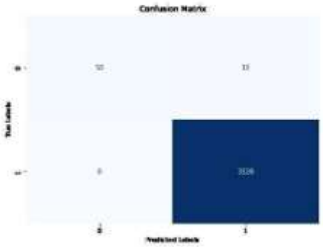

Figure 5.3: RF Confusion Matrix

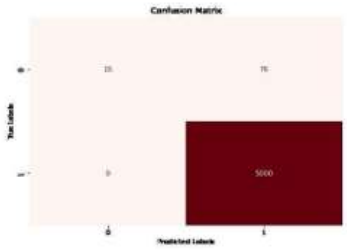

Figure 5.4: LR Confusion Matrix

5.3.2 Performance Analysis

In each classification algorithm, predictive parameters are configured to assess the entire model based on the values of the confusion matrix. The evaluation metrics considered in this study are accuracy,

recall, precision, and F1 score. Accuracy evaluates the classification model by calculating the total number of true values (diabetic individuals in this case) divided by the total number of records. The accuracy of a model is determined by the sum of true positive and true negative predictions divided by the total number of predictions, including true positives, true negatives, false positives, and false negatives, as shown in Eq.1.

$$Accuracy = \frac{(true\ positives + True\ negatives)}{total\ number\ of\ predictions} \quad (1)$$

Recall as a critical parameter in medical classifications, especially in gauging the ability of sensors to accurately detect positive values, such as diabetic cases in our context. It assesses the proportion of true positive predictions (TP) against the total actual positive cases. Recall is calculated by dividing the number of true positives by the sum of true positives and false negatives, as shown in Eq. 2.

$$Recall = \frac{True\ postives}{(True\ positives + True\ negatives)} \quad (2)$$

Precision, on the other hand, is equally crucial in medical classifications, focusing on the accuracy of positive predictions made by the model. It measures the proportion of true positive predictions against the total predicted positives. Precision is computed by dividing the number of true positives by the sum of true positives and false positives, as shown in Eq.3.

$$Precision = \frac{True\ positives}{(true\ positives + false\ positives)} \quad (3)$$

The F1 score serves as a comprehensive metric, combining both precision and recall to provide a balanced evaluation of the model's performance. It is calculated by taking the harmonic mean of precision and recall, providing a single value that indicates the overall effectiveness of the classification model in terms of both false positives and false negatives, as shown in Eq.4.

$$F1\ Score = 2\frac{(Precison * Recall)}{Precison + Recall} \quad (4)$$

5.3.3 Results and Discussion

Through the 16968 records of the data set, there were 16641 diabetic people, and the rest, which is only 328, were non-diabetic.

Four different machine-learning models were applied, each undergoing HT through FA, subsequent to splitting the dataset into the training and testing sets. The major parameters covered in this work are Accuracy. Moreover, a summarizing graph will be presented to compare the performance of models

trained and tested with optimization techniques against those without optimization, as shown in Fig.5.5.

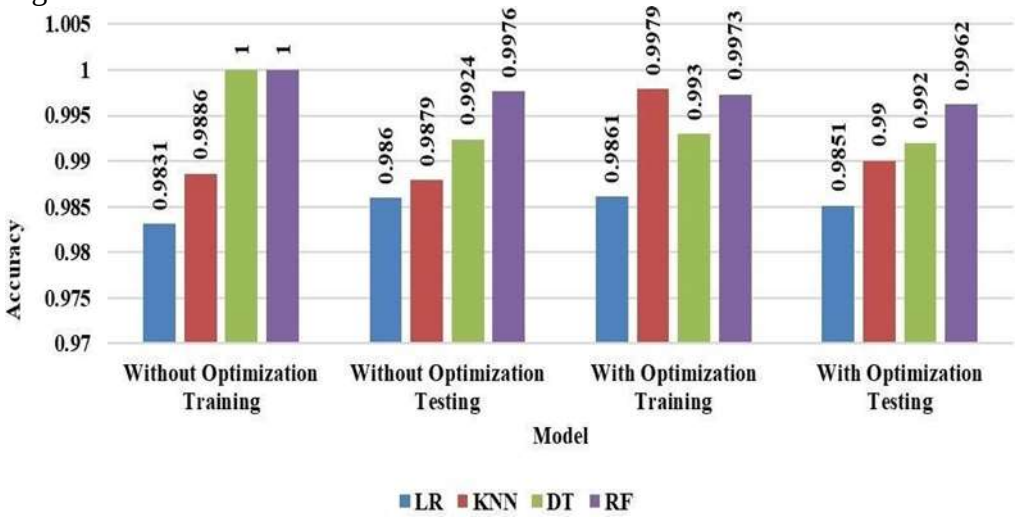

Figure 5.5: Comparison of Model Performance with and without Optimization Techniques

Receiver Operating Characteristic (ROC) curves are utilized to evaluate the performance of models trained and tested with optimization techniques by plotting the true positive rate (TPR) against the false positive rate (FPR) across various classification thresholds. Models trained with optimization techniques often yield ROC curves that exhibit superior performance, with curves tending towards the upper left corner of the plot, indicative of higher TPR and lower FPR. This suggests that optimization techniques enhance the model's ability to distinguish between positive and negative instances, resulting in improved discriminatory power and overall effectiveness in classification tasks. Figure 5.6,5.7,5.8 and 5.9 illustrate the ROC curves of KNN, DT, RF, and Algorithms for

with optimization testing, respectively, providing insights into their respective classification performance.

Figure 5.6: ROC curve for KNN

Figure 5.7 : ROC curve for RF

Figure 5.8: ROC curve for DT

Figure 5.9: ROC curve for LR

Precision-Recall (PR) curves offer another perspective on model performance, particularly in scenarios with imbalanced datasets, and are crucial when evaluating models trained and tested with optimization techniques. By plotting precision against recall, PR curves provide insights into the trade-off between correctly identifying positive instances and minimizing false positives. Models trained with optimization techniques typically produce PR curves that approach the upper right corner, indicating higher precision and recall values. This suggests that optimization techniques contribute to achieving a better balance between precision and recall, ultimately leading to more accurate and reliable classification outcomes, especially in situations where correctly identifying positive instances is paramount. Figure 5.10,5.11,5.12 and 5.13 illustrate the PR curves of KNN, DT, RF, and LR

algorithms for with optimization testing, respectively, providing insights into their respective classification performance.

http://doi.org/10.36893/JNAO.2024.V15I1N01.283-310

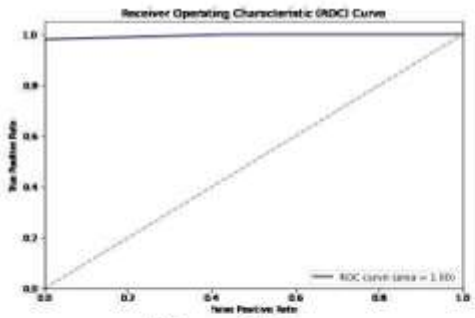

Figure 5.6:
ROC curve for KNN

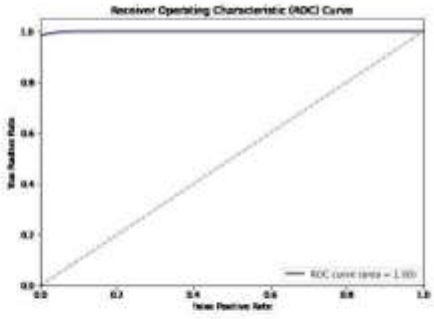

Figure 5.7 :
ROC curve for RF

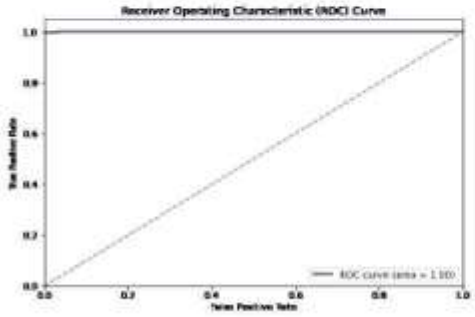

Figure 5.8:
ROC curve for DT

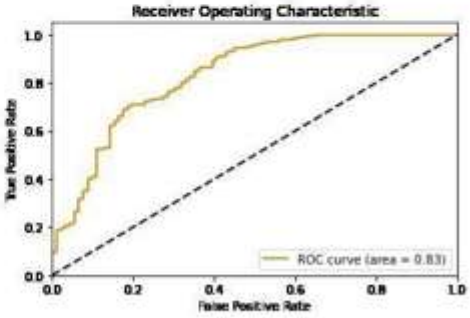

Figure 5.9:
ROC curve for LR

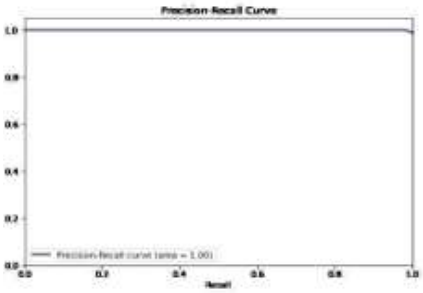

Figure 5.10:
PR curve for KNN

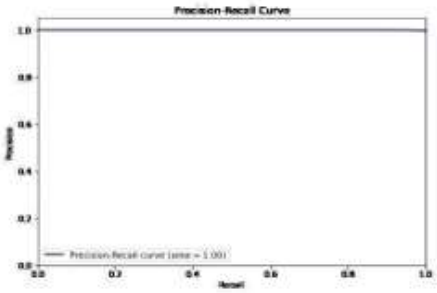

Figure 5.11
PR curve for DT

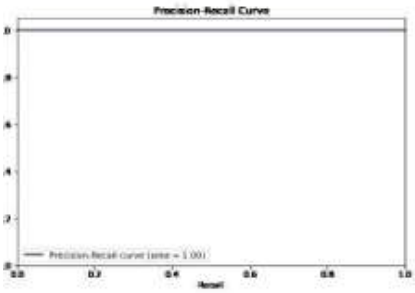

Figure 5.12:
PR curve for RF

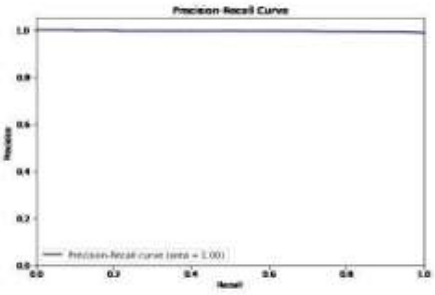

Figure 5.13:
PR curve for LR

5.3.4 Comparative Analysis with past related work

Table 2.1 presents the classification accuracy over the past few years, summarizing notable results obtained from various ML algorithms. It's noteworthy that while the datasets differ across studies, they share a common objective: predicting diabetic patients from medical datasets containing essential features. Notably, this study introduces a comparison to assess the performance of models trained and tested with optimization techniques against those without optimization.

CHAPTER – 6 CONCLUSION AND FUTURSCOPE

Conclusion

Diabetes needs to be detected early before it reaches a dangerous stage. Hence, this study aimed to identify the most effective and accurate machine-learning method for classifying objects by implementing four different ML methods with a comparative analysis of past work.
In contrast to others, we used HT with NIO (FA) technique on data to produce the most accurate result. Our experimental results demonstrate that using large datasets, the accuracy achieved by the RF algorithm was the highest among all methods, measuring 99.62%. This encompasses exploring alternative ML algorithms for predicting not only diabetes but also other diseases.

Future scope

By delving deeper into NIO algorithms and techniques, such as genetic algorithms, particle swarm optimization, and ant colony optimization, we can unlock new levels of efficiency and scalability. Leveraging advancements in computational power and algorithmic sophistication, the future scope for the project involves refining existing NIO methodologies, exploring novel hybridization approaches, and integrating cutting-edge optimization frameworks. This evolution aims to address complex optimization challenges across diverse fields, from finance and logistics to engineering and beyond.

PROJECT OUTCOMES – PO/PSO MAPPING

Batch No : C11
Domain : Machine Learning
Project Title : Harnessing Nature Inspired Optimization Technique Using Hyperparameter Tuning in Machine Learning Algorithms

CO	**Course Outcomes for Student Projects**	**Relevance to POs /PSOs**
CO1	Understand and apply various nature-inspired optimization algorithms to solve complex optimization problems in machine learning. (K4)	PO1 – PO12 PSO1
CO2	Develop expertise in the art of hyperparameter tuning by exploring different optimization strategies to fine-tune model parameters effectively, leading to improved model performance and generalization. (K4)	
CO3	Gain practical experience in implementing and experimenting with machine learning models across different domains, leveraging nature-inspired optimization techniques for model optimization. (K3)	
CO4	Enhance critical thinking abilities by analyzing the behavior and performance of nature-inspired optimization algorithms in the context of hyperparameter tuning, and develop strategies to overcome challenges and limitations. (K5)	

http://doi.org/10.36893/JNAO.2024.V15I1N01.283-310

CO5	Perceive effective communication skills, professional behavior and teamwork(K5)	

Project Outcome - POs/PSOs Mapping:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO 10	PO 11	PO 12	PS 01	PS O2
C409.1	3	3	3	3	-	3	-	3	3	3	3	3	3	3
C409.2	3	3	3	-	3	3	2	3	3	3	2	3	3	3
C409.3	3	3	-	3	3	3	3	3	3	3	3	3	3	3
C409.4	3	3	3	-	3	3	2	3	3	3	2	3	3	3
C409.5	3	3	-	3	-	-	3	-	-	3	3	3	3	3
Avg	**3.00**	**3.00**	**3.00**	**3.00**	**3.00**	**3.00**	**2.50**	**3.00**	**3.00**	**3.00**	**2.50**	**3.00**	**3.00**	**3.00**

REFERENCES

[1] R. Rastogi and M. Bansal, “Diabetes prediction model using data mining techniques”, Measurement: Journal of the International Measurement Confederation (IMEKO), Measurement: Sensors Volume 25, February 2023.

[2] M. E. Febrian, F. X. Ferdinan, G. P. Sendani, K. M. Suryanigrum, and R. Yunanda, “Diabetes Prediction using Supervised Machine Learning", Procedia Computer Science, 216, pp. 21-30, 2023.

[3] S. S. Bhat, V. Selvam, G. A. Ansari, M. D. Ansari, and M H. Rahman, “Prevalence and Early Prediction of Diabetes Using Machine Learning in North Kashmir: A Case Study of District Bandipora”, Computational Intelligence and Neuroscience, Volume 2022, Article ID 2789760.

[4] K. Patel, M. Nair and S. Phansekar, “Diabetes Prediction using Machine Learning”, International Journal of Scientific & Engineering Research Volume 12, Issue 3, March 2021.

[5] J. Xue, F. Min, and F. Ma, “Research on Diabetes Prediction Method Based on Machine Learning”, Journal of Physics: Conference Series, 2020.

[6] M. Soni and S. Varma, “Diabetes Prediction Using Machine Learning Techniques”, International Journal of Engineering Research & Technology, Vol. 9, Issue 09, September- 2020.

[7] L. J. Muhammad, E. A. Algehyne, and S. S. Usman, “Predictive Supervised Machine Learning Models for Diabetes Mellitus", SN Computer Science, 1, 240, 2020.

[8] A. Dutta, M. K. Hasan, M. Ahmad, M. A. Awal, M. A. Islam, M. Masud, and H. Meshref, “Early Prediction of Diabetes Using an Ensemble of Machine Learning Models”, Int. J. Environ. Res. Public Health, 19, 2022.

[9] D. Sisodia and D. S. Sisodia, “Prediction of Diabetes using Classification Algorithms”, International Conference on Computational Intelligence and Data Science, Procedia Computer Science, pp. 1578–1585, 2018.

[10] Q. Zou, K. Qu, Y. Luo, D. Yin, Y. Ju, and H. Tang, “Predicting Diabetes Mellitus with Machine Learning Techniques”, Front. Genet. Vol. 9, pp. 1-10, 2018.

[11] Dr. D. Manendra Sai, Et Al.,(2023)Utilizing Machine Learning Algorithms For Kidney Disease Prognosis, European Journal of Molecular & Clinical Medicine, Volume 10,Issue 01 Pages 37-50.

[12] Dr. D. Manendra Sai, Et Al.,(2023) Machine Learning Techniques Based Prediction for Crops in Agriculture, Journal of Survey in Fisheries Sciences, Volume 10, Issue 1s, Pages 3710- 3717.