INNOVATIVE APPRAOCHES FOR MULTICLASS IDENTIFICATION OF PADDY DISEASES THROUGH ENHANCED FEATURE TRANSFORMATION

Report submitted to SASTRA Deemed to be University
As per the requirement for the course

CSE300: MINI PROJECT

Submitted by

SUDEENDRA NANDIMANDALAM
(Reg No.: 125003465, B. Tech Computer Science and Engineering)

KIRAN KUMAR REDDY YENUMULA
(Reg. No.: 125014027, B. Tech Information & Communication Technology)

HARSHA SUMANTH REDDY YANNAM
(Reg. No.: 125015167, B. Tech Information Technology)

May 2024

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled **"Innovative Approaches for Multiclass Identification of Paddy Diseases Through Enhanced Feature Transformations"** submitted as a requirement for the course, **CSE300 : MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Mr. Sudeendra Nandimandalam (Reg. No.: 125003465, B. Tech Computer Science and Engineering**), **Mr. Kiran Kumar Reddy Yenumula (Reg. No.: 125014027, B. Tech Information and communication Technology) and Mr. Harsha Sumanth Reddy Yannam (Reg. No.: 125015167, B. Tech Information Technology)** during the academic year 2023-2024, in the School of Computing, under my supervision.

Signature of Project Supervisor :

Name with Affiliation : Smt. Gowri L, Associate Professor, School of Computing

Date :

Mini Project *Viva voice* held on ________________

Examiner 1 **Examiner 2**

ACKNOWLEDGEMENTS

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. V. S. Shankar Sriram**, Dean, School of Computing, **Dr. R. Muthaiah**, Associate Dean, Research, **Dr. K. Ramkumar**, Associate Dean, Academics, **Dr. D. Manivannan**, Associate Dean, Infrastructure, **Dr. R. Algeswaran**, Associate Dean, Students Welfare for their motivation and support offered in materializing this project.

Our guide **Smt. Gowri L**, Associate Professor, School of Computing was the driving force behind this whole idea from the start. Her deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing me an opportunity to showcase my skills through this project.

List of Figures

List of Tables

Abbreviations

KNN	K-Nearest Neighbor
RFC	Random Forest Classifier
LDA	Linear Discriminant Analysis
HGBC	Histogram Gradient Boosting Classifier
LOA	Lemurs Optimization Algorithm
MLOA	Modified Lemurs Optimization Algorithm
BLB	Bacterial Leaf Blight
BLT	Blast
HLT	Healthy
LF	Leaf Folder
LS	Leaf Spot
FRR	Free Risk Rate
HRR	High Risk Rate
LRR	Low Risk Rate
BAC	Balanced Accuracy Score
MCC	Matthews Correlation Co-efficient

ABSTRACT

This research addresses the significant challenges of pests and diseases in paddy production, causing a global loss of approximately 20% in rice yield. The focus is on early identification of rice leaf diseases using thermal image cameras to mitigate losses. The study introduces a Modified Lemurs Optimization Algorithm as a filter-based feature transformation technique to enhance the accuracy of paddy disease detection through machine learning.

Inspired by Sine Cosine Optimization, the Modified Lemurs Optimization Algorithm is applied to thermal images of paddy leaves, considering five diseases namely rice blast, brown leaf spot, leaf folder, hispa, and bacterial leaf blight are considered in our work. Six hundred thirty-six thermal images are analyzed, extracting statistical and Box-Cox transformed features. Four machine learning techniques are tested namely K-Nearest Neighbor (KNN), Random Forest Classifier, Linear Discriminant Analysis and Histogram Gradient Boosting Classifier and their performance is enhanced by the proposed feature transform, with KNN achieving a balanced accuracy of 90%.

KEY WORDS: KNN, RFC, LDA, HGBC, Modified Lemurs Optimization Algorithm.

Table of Contents

CHAPTER 1
SUMMARY OF THE BASE PAPER

Title	:	Multiclass Paddy Disease Detection Using Filter-Based Feature Transformation Technique
Publisher	:	IEEE
Year	:	2023
Journal name	:	IEEE Access
ISSN NO	:	2169-3536
DOI	:	10.1109/ACCESS.2023.3322587
Base Paper URL	:	https://ieeexplore.ieee.org/document/10273722

The main contributions of the base paper are:

- Usage of A Novel Optimization Technique inspired from Lemurs called Lemurs Optimization Technique.

Our method consists of 5 major steps:

1.1 Introduction

Rice serves as a crucial staple in diets worldwide, and its production is consistently expanding annually, projected to grow by 1.5 times by 2030. However, numerous obstacles, such as pests, climate variations, and pathogens, pose challenges to rice cultivation. Failure to promptly diagnose and address these issues can lead to significant losses, potentially up to 50%. Hence, the timely detection of these ailments is paramount to mitigate losses. Employing regular visual inspections via thermal imaging emerges as the optimal strategy to curtail such losses.

1.2 Statistical Feature Extraction

Leveraging the capabilities of the NumPy module, we conduct a comprehensive analysis of thermal images representing five distinct paddy diseases: blast, bacterial leaf blight (BLB), hispa, leaf spot, leaf folder, as well as a healthy state. Seven key statistical features, including mean, skewness, kurtosis, variance, variation value, standard error of the mean (SEM), and entropy, are computed to capture the unique thermal signatures associated with each disease. Notably, skewness and kurtosis values are flattened from a 3D array to ensure a concise 1D representation. This approach optimizes computational efficiency and facilitates precise characterization and differentiation of disease states within agricultural contexts.

1.3 Transformation and Scaling

The Box-Cox transformation is utilized to stabilize variance and promote data normality, particularly beneficial for handling heteroscedasticity. This technique involves applying a power function to the data, adjusting it to achieve optimal normality.

In contrast, robust scaling is employed to standardize features to a uniform range while being less influenced by outliers. Unlike traditional scaling methods using mean and standard deviation, robust scaling relies on more robust statistics like median and interquartile range, making it more resilient to outliers or skewed distributions.

By integrating both the Box-Cox transformation and robust scaling methods on the seven statistical features, we aim to address issues related to non-normality and outliers, thus improving the data's suitability for further analysis and modeling.

We'll pass 7 statistical features to Box-Cox transformation and get pass the output of them as input to Robust scaling and we'll obtain output as a mix of 'Transformed + Scaled' and on the other side of the coin we'll pass 7 statistical features directly into Robust scaling as input and we'll obtain output as a mix of 'Not Transformed + Scaled'. Together we ended with 14 statistical features in the end.

1.4 Filter-Based Feature Transformation

The initial population of lemurs is defined as shown in equation below:

$$T = \begin{bmatrix} L_1^1 & \cdots & L_1^d \\ \vdots & \ddots & \vdots \\ L_s^1 & \cdots & L_s^d \end{bmatrix}$$

LOA (Lemur Optimization Algorithm) predominantly employs two locomotive behaviors observed in lemurs: "leap up" and "dance-hop". The former inspires the exploitation phase, akin to local search, while the latter drives the exploration phase, similar to global search. Lemur positions serve as candidate solutions, updated in each iteration based on an objective or fitness function. The lemur population is typically defined by Equation above, with the population matrix size being $s * d$, where s denotes the number of candidate solutions (equivalent to the number of lemurs), and d denotes the number of decision variables.

The position of lemurs is randomly initialized as shown in equation below:

$$L_i^j = \text{rand} * ((ub_j - lb_j) + lb)$$

In every iteration the position of lemurs is updated from the equation below:

$$L_{i+1}^{j} = \begin{cases} L_i^j + \ (\mathrm{abs}(\mathrm{L_i^j}\ \text{-}\ \mathrm{gbest_j}) * (rand - 0.5) * 2) & if\ \mathrm{rand} \geq \mathrm{FRR} \\ L_i^j + \ (\mathrm{abs}(\mathrm{L_i^j}\ \text{-}\ \mathrm{nbest_j}) * (rand - 0.5) * 2) & if\ \mathrm{rand} < \mathrm{FRR} \end{cases}$$

During each iteration, a random number, denoted as rand and falling between 0 and 1, is generated. The upper and lower boundary limits are respectively represented as ubj and lbj. The global best lemur and the closest best lemur are identified based on the fitness function. Subsequently, the positions of lemurs are adjusted according to equation above.
In the above scenario, i and j represent the iteration number and decision variable, respectively. "gbest" indicates the global best lemur position, while "nbest" denotes the position of the nearest best lemur. "L" stands for the lemur's position, and "rand" represents a random number within the range of 0 to 1.
The Free Risk Rate (FRR) reflects the risk level of all lemurs in the troop, and this coefficient can be determined using equation below.

$$\mathrm{FRR} = \mathrm{FRR}\ \text{-}\ \mathrm{Crnt_{iter}} * ((\mathrm{HRR}\ \text{-}\ \mathrm{LRR})/\mathrm{Max_{iter}})$$

In the equation above, HRR refers to the High-Risk Rate, and LRR denotes the Low-Risk Rate, both being fixed predefined values. "$\mathrm{Crnt_{iter}}$" stands for the current iteration, and "$\mathrm{Max_{iter}}$" represents the maximum number of iterations.

Using LOA alone does not achieve the anticipated accuracy level, prompting the adoption of Modified LOA to address this limitation.

Algorithm of MLOA (Modifies Lemurs Optimization Technique):

The Lemurs Optimization Technique is a metaheuristic algorithm inspired by the social behavior of lemurs. It is designed to optimize feature selection for classification tasks based on thermal image data. Below is a detailed explanation of the algorithm steps:

1. Feature Extraction: Begin by extracting 14 statistical features from the thermal image data. These features represent key characteristics extracted from the images. Consider each statistical feature as a lemur so, we have 14 lemurs for a thermal image.

2. Initialization: Initialize the position of lemurs in the feature space using the extracted statistical features. Each lemur's position is represented by a vector of these features.

$$\mathrm{T} = [\mathrm{L_1 L_2 L_3} \ldots \mathrm{L_{14}}]$$

Here T is initial population, and $\mathrm{L_1 L_2 L_3} \ldots \mathrm{L_{14}}$ are 14 statistical features.

3. Parameter Initialization: Set up the parameters of the Maximum Lemurs Optimization Algorithm (MLOA). Determine the maximum and minimum values of the statistical

features extracted from the images (L_{max} and L_{min} respectively). Define other parameters such as the maximum number of iterations (Max_{iter}), initial false reject rate ($FRR_{initial}$), and learning rate (Lrate).

4. Fitness Computation: Calculate the fitness value of each lemur using a fitness function defined in Equation (8). This function evaluates how well a particular set of features contributes to the classification task.

$$F(L_i) = Var(L_{i-1}, L_i, L_{i+1})$$

5. Global and Neighborhood Bests: Identify the global best (gbest) and neighborhood best (nbest) lemurs based on their fitness values. The gbest represents the lemur with the best overall fitness, while nbest represents the best within a neighborhood.

6. Position Update: Update the position of all lemurs using Equations (5) and (6). This step involves adjusting the positions of lemurs based on their current positions, velocities, and the influence of gbest and nbest.

$$L_{i+1}^{j} = \begin{cases} L_i^j + r1 * \sin\left(abs\left(L_i^j - gbest^j\right)\right) * (rand - 0.5) * 2 & if\ FRR \geq 0.5 \\ L_i^j + r1 * \cos\left(abs\left(L_i^j - nbest^j\right)\right) * (rand - 0.5) * 2 & if\ FRR < 0.5 \end{cases}$$

$$r1 = a - Crnt_{iter} * (\frac{a}{Max_{iter}})$$

7. Worst Lemur Update: Identify the lemur with the lowest fitness value as the worst lemur. Update its position using Equation (7), allowing it to explore new regions of the feature space.

$$L_{worst,i} = L_{min} + (L_{max} - L_{min}) * rand$$

Here, L_{max} and L_{min} represent the maximum and minimum positions of Lemurs, respectively. These values are determined based on the initial population of Lemurs. The term rand denotes a random number between 0 and 1.

8. Loss Function and Weight Parameter Update: Compute the loss function and update the weight parameter FRR using Equations (9) and (10). This step adjusts the false reject rate based on the performance of the lemurs in feature selection.

$$W_{t+1} = W_t - Lrate * \frac{\partial L}{\partial W_t}$$

Here, W_t and W_{t+1} denote the old and new weights, respectively, while Lrate represents the learning rate.

$\frac{\partial L}{\partial W_t}$ denotes the gradient of the loss function concerning weight 'w'. In this study, the inverse of variance is considered as the loss function, aiming to minimize it to compute the optimal value for the weight parameter FRR, which is treated as weight w. The formulated loss function is as follows:

$$\frac{\partial L}{\partial FRR_t} = \begin{cases} (Ivar_t / FRR_{initial}) & \text{if } t = 1 \\ (Ivar_t - Ivar_{t-1}) / (FRR_t - FRR_{t-1}) & \text{if } t > 1 \end{cases}$$

Here, t denotes the current iteration number, and *I*var represents the inverse of variance of the entire Lemur population.

Using trial and error method, the ideal values for MLOA parameters are found as $Max_{iter} = 12$, $FRR_{initial} = 0.5$, and Lrate = 0.5.

9. Iteration: Repeat steps 4 to 8 until the maximum number of iterations is reached. If the maximum number of iterations is completed, proceed to step 10.

10. Output: Consider the final positions of lemurs as the output of the feature transformation process. These transformed features are then used as input to classifiers for the classification task.

In summary, the Lemurs Optimization Technique iteratively optimizes feature selection for classification tasks based on thermal image data by simulating the social behavior of lemurs and updating their positions in the feature space to maximize classification performance.

1.5 Supervised Classifiers:

In this section, we assess the effectiveness of the proposed MLOA-based feature transformation by comparing the performance of classifiers with and without this transformation. Four supervised classifiers—KNN, RFC, LDA, and HGBC—are employed in this study, chosen randomly from a selection of supervised classifiers. The main aim is to showcase the performance enhancement in classification achieved through the application of the MLOA-based feature transformation.

We evaluate classifier performance using various metrics available in the literature. Specifically, we utilize five metrics: Balanced Accuracy (BAC), Mathews Correlation Coefficient (MCC), Weighted Average F1-score, Weighted Average Precision, and Weighted Average Recall. These metrics are chosen for their suitability in assessing imbalanced multi-class datasets, as they consider the number of subjects in each class as weights. BAC and MCC offer an overarching overview of classification performance, and we observe clear improvements in these metrics when employing the proposed MLOA transform.

1.6 Performance Analysis

In our investigation, we thoroughly analyzed the performance to gauge the effectiveness of the proposed MLOA-based feature transformation method in bolstering classification outcomes. We employed four supervised classifiers—KNN, RFC, LDA, and HGBC—which were randomly selected to showcase the impact of the transformation. Our primary aim was to demonstrate how employing MLOA-based feature transformation leads to improved classification performance.

To assess the classifiers' effectiveness, we utilized a variety of performance metrics including Balanced Accuracy (BAC), Mathews Correlation Coefficient (MCC), Weighted Average F1-score, Weighted Average Precision, and Weighted Average Recall. These metrics are specifically tailored for evaluating classifiers in scenarios involving imbalanced multi-class datasets, as they consider the distribution of subjects across various classes.

Our analysis underscored notable enhancements in classification performance metrics when employing the proposed MLOA-based feature transformation. Notably, both BAC and MCC exhibited improvements, indicating that the MLOA-based feature transformation enhances the classifiers' ability to discern between different classes, thereby boosting accuracy and reliability in classifying instances across diverse classes.

CHAPTER 2

MERITS AND DEMERITS OF THE BASE PAPER

LITERATURE SURVEY:

There are various methods to classify/predict paddy diseases based on the thermal images

- **"Damaged paddy leaf detection using image processing"** by M. Manoj, T. Pal, and D. Samanta. This paper highlights that in agriculture, crop yield loss often arises from the spread of diseases, which are typically identified only when they reach an advanced stage, leading to significant losses in yield, time, and finances. The proposed system aims to address this issue by enabling early detection of diseases as soon as they appear on the leaves, thus mitigating losses and reducing the reliance on experts to some extent. This system can offer assistance to individuals with limited knowledge about plant diseases. To achieve these objectives, we need to extract disease-specific features. Plant disease detection is crucial in agriculture, given the natural occurrence of diseases in plants. Automated techniques for disease detection alleviate the burden of monitoring large crop farms by identifying symptoms at an early stage when they manifest on plant leaves. Through this paper, we present a methodology for detecting and diagnosing plant leaf diseases, which we believe will aid farmers in maintaining crop health and addressing affected plants promptly. We anticipate that our proposed system will contribute positively to the field of agriculture.

- **"Identification of rice diseases using deep convolutional neural networks"** by Y. Lu, S. Yi, N. Zeng, Y. Liu, and Y. Zhang. This paper underscores the importance of rice as a crucial global food crop and the threats posed by rice diseases to its production and food security. It examines the drawbacks of conventional diagnosis methods and explores the promise of automated approaches using deep convolutional neural networks (CNNs) to enhance accuracy and efficiency. The objective is to create a CNN-based model for swift and precise identification of rice diseases, harnessing the benefits of direct image input and robust classification. The study outlines the CNN architecture and learning algorithm, describes the method for rice disease identification, and presents findings from experiments along with conclusions.

- **"A comparative study of fine-tuning deep learning models for plant disease identification"** by E. C. Too, L. Yujian, S. Njuki, and L. Yingchun. According to this paper, there has been a recent surge in interest in deep learning, driven by its potential to create swift, automated, and precise systems for image identification and classification. This study delves into the fine-tuning and assessment of cutting-edge deep convolutional neural network architectures for categorizing plant diseases based on images. It conducts a practical comparison of architectures like VGG 16, Inception V4, ResNet with 50, 101, and 152 layers, and DenseNets with 121 layers. Using a dataset featuring 38 different classes of diseased and healthy leaf images sourced from PlantVillage, the objective is to develop rapid and accurate models for plant disease identification, thereby tackling concerns regarding food security. The experiment reveals that DenseNets consistently enhance accuracy as epochs progress, without succumbing to overfitting or performance decline. Additionally, DenseNets demand fewer parameters and less computation time to achieve state-of-the-art results. They achieve an impressive testing accuracy score of 99.75%, outperforming other architectures. Training of these architectures was conducted using Keras with Theano backend.

- **"Preliminary study for identifying rice plant disease based on thermal images"** by M. S. Lydia, I. Aulia, I. Jaya, D. S. Hanafiah, and R. H. Lubis. As outlined in this paper, precise and prompt identification of plant diseases is crucial for stakeholders and farmers to minimize losses caused by pests and diseases. Thermal imaging technology offers an indirect detection method known for its speed and non-invasiveness, making it a valuable tool in disease identification. This paper details a preliminary study on gathering thermal images to identify rice plant diseases on leaf canopies using a thermal imaging camera.

- **"'Image processing based rice plant leaves diseases in Thanjavur, Tamilnadu"** by T. G. Devi and P. Neelamegam. As stated in this paper, India, being largely agricultural with 60.3% of its land dedicated to farming, grapples with a substantial decline in rice production, estimated at 20–30%, due to various diseases like leaf blast, leaf blight, false smut, brown spot, and leaf streak. This study focuses on automating the detection of these leaf diseases through image processing techniques. The proposed method involves several steps: image acquisition, preprocessing, segmentation, and classification of paddy leaf diseases. Features are extracted using a hybrid approach combining discrete wavelet transform, scale-invariant feature transform, and gray-scale co-occurrence matrix techniques. These features are then fed into various classifiers such as K-nearest neighbors, backpropagation neural network, Naïve Bayesian, and multiclass SVM to distinguish between diseased and healthy plants. The effectiveness of different classification methods is assessed through experiments conducted using MATLAB, with performance evaluated based on accuracy. The research finds that the multiclass SVM classifier achieves the highest accuracy, reaching 98.63%, outperforming other classifiers.

MERITS AND DEMERITS

Merits:

- Combined effort to tackle the significant issue of crop loss due to diseases, emphasizing the urgency of early detection.
- Examination of diverse methodologies, including deep learning, thermal imaging, and image processing, for automating disease identification in plants.
- Provision of methodologies and insights into the potential of various technologies for improving disease detection and diagnosis in agriculture.
- Practical experimentation and results demonstrating the efficacy of proposed approaches in detecting and diagnosing plant diseases.

Demerits:

- Insufficient elaboration on specific methodologies, experimental setups, and performance evaluation metrics in certain papers.
- Potential gaps in the validation and comparison of proposed methods with existing techniques.
- Limited exploration of the scalability and robustness of proposed approaches across different datasets and environmental conditions.
- Room for further research to enhance the effectiveness and practical application of automated disease identification systems in agriculture.

CHAPTER 3

SOURCE CODE

3.1 Extracting and Pre-Processing of Thermal Images

```
[1]: import cv2
import os
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow
import numpy as np
import math as m
from skimage.measure import shannon_entropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img
from scipy.stats import skew,kurtosis,boxcox
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit, StratifiedKFold
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from sklearn.metrics import recall_score
from skimage.io import imread
from skimage.color import rgb2gray
from sklearn.metrics import balanced_accuracy_score, matthews_corrcoef
from sklearn.preprocessing import RobustScaler

WARNING:tensorflow:From C:\Users\Sudeendra\AppData\Roaming\Python\Python311\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_
softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

```
[580]: folder_path = r"C:/Users/Sudeendra/Downloads/thermal images UL/"
total_mean_list = []
total_variance_list = []
total_entropy_list = []
total_skewness_list = []
total_kurtosis_list = []
total_varvalue_list = []
total_sem_list = []
```

```
[581]:

# Function to load and preprocess image
def load_and_preprocess_image(image_path):
    # Load image
    img = imread(image_path)
    return img

# Initialize lists to store statistical features
blb_img_list=[]
blb_mean_list = []
blb_variance_list = []
blb_entropy_list = []
blb_skewness_list = []
blb_kurtosis_list = []
blb_varvalue_list = []
blb_sem_list = []

# Define folder path and disease label
disease = "BLB"

# List all files in the folder
file_list = os.listdir(folder_path + disease)

# Iterate over images
for i in range(1, min(221, len(file_list)), 1):
    # Load and preprocess image
    img_path = os.path.join(folder_path, disease, file_list[i])
    img = load_and_preprocess_image(img_path)

    # Calculate statistical features
    blb_mean_list.append(np.mean(img))
    blb_variance_list.append(np.var(img))
    blb_entropy_list.append(shannon_entropy(img))
    blb_skewness_list.append(skew(img.flatten()))  # Flatten the image array before calculating skewness
    blb_kurtosis_list.append(kurtosis(img.flatten()))  # Flatten the image array before calculating kurtosis
```

```
    blb_varvalue_list.append((np.var(img) / np.mean(img)))
    blb_sem_list.append((np.std(img) / np.sqrt(img.size)))

    blb_img_list.append(img)


    total_mean_list.append(np.mean(img))
    total_variance_list.append(np.var(img))
    total_entropy_list.append(shannon_entropy(img))
    total_skewness_list.append(skew(img.flatten()))
    total_kurtosis_list.append(kurtosis(img.flatten()))
    total_varvalue_list.append((np.var(img)/np.mean(img)))
    total_sem_list.append((np.std(img)/np.sqrt(img.size)))

# Plotting the images (optional)
plt.figure(figsize=(12, 12))
for i in range(len(blb_img_list)):
    plt.subplot(19, 12, i + 1)
    plt.imshow(blb_img_list[i])
plt.show()
```

#Similarly do for all the other diseases namely hispa,leaf folder,leaf spot,blast,healthy…

3.2 Box-Cox transformations of 7 Statistical Features

```
blb_mean_list_transformed, _ = boxcox(blb_mean_list)
blb_variance_list_transformed, _ = boxcox(blb_variance_list)
blb_entropy_list_transformed, _ = boxcox(blb_entropy_list)
blb_skewness_list_shifted = blb_skewness_list - np.min(blb_skewness_list) + 1
blb_skewness_list_transformed, _ = boxcox(np.ravel(blb_skewness_list_shifted))
blb_kurtosis_list_shifted = blb_kurtosis_list - np.min(blb_kurtosis_list) + 1
blb_kurtosis_list_transformed, _ = boxcox(np.ravel(blb_kurtosis_list_shifted))
blb_varvalue_list_transformed, _ = boxcox(blb_varvalue_list)
blb_sem_list_shifted = blb_sem_list - np.min(blb_sem_list) + 1
blb_sem_list_transformed, _ = boxcox(np.ravel(blb_sem_list_shifted))
```

```
fig, axes = plt.subplots(nrows=6, ncols=2, figsize=(15, 12))
fig.suptitle('Histograms of Multiple statistical measures before and after Box-Cox transformation', fontsize=16)

axes[0,0].hist(total_mean_list,label='Mean',color='blue')
axes[0,1].hist(total_mean_list_transformed,label='bc mean',color='blue')

axes[1,0].hist(total_variance_list,label='variance',color='red')
axes[1,1].hist(total_variance_list_transformed,label='bc variance',color='red')

axes[2,0].hist(total_entropy_list,label='entropy',color='green')
axes[2,1].hist(total_entropy_list_transformed,label='bc entropy',color='green')

axes[5,0].hist(total_skewness_list_transformed,label='bc skewness',color='pink')
axes[5,1].hist(total_kurtosis_list_transformed,label='bc kurtosis',color='purple')

axes[3, 0].hist(total_varvalue_list,label='varvalue',color='brown')
axes[3, 1].hist(total_varvalue_list_transformed,label='bc varvalue',color='brown')

axes[4, 0].hist(total_sem_list,label='semvalue')
axes[4, 1].hist(total_sem_list_transformed,label='bc semvalue')

for ax in axes.flat:
    ax.set(xlabel='statistical Values', ylabel='Frequency')
    ax.legend()

# Adjust layout
plt.tight_layout(rect=[0, 0.05, 0.5, 0.95])

# Show the plot
plt.show()
```

3.3 Robust scaling of statistical features before and after tansformation

```
rc=RobustScaler()
```

```
fig, axes = plt.subplots(nrows=7, ncols=2, figsize=(15, 12))
fig.suptitle('Histograms of Multiple statistical measures before scaling and after Robust scaling', fontsize=16)

msd_b=rc.fit_transform(np.array(total_mean_list).reshape(-1,1))
vsd_b=rc.fit_transform(np.array(total_variance_list).reshape(-1,1))
esd_b=rc.fit_transform(np.array(total_entropy_list).reshape(-1,1))
sksd_b=rc.fit_transform(np.array(total_skewness_list).reshape(-1,1))
ksd_b=rc.fit_transform(np.array(total_kurtosis_list).reshape(-1,1))
vvsd_b=rc.fit_transform(np.array(total_varvalue_list).reshape(-1,1))
ssd_b=rc.fit_transform(np.array(total_sem_list).reshape(-1,1))

msd_a=rc.fit_transform(total_mean_list_transformed.reshape(-1,1))
vsd_a=rc.fit_transform(total_variance_list_transformed.reshape(-1,1))
esd_a=rc.fit_transform(total_entropy_list_transformed.reshape(-1,1))
sksd_a=rc.fit_transform(total_skewness_list_transformed.reshape(-1,1))
ksd_a=rc.fit_transform(total_kurtosis_list_transformed.reshape(-1,1))
vvsd_a=rc.fit_transform(total_varvalue_list_transformed.reshape(-1,1))
ssd_a=rc.fit_transform(total_sem_list_transformed.reshape(-1,1))
```

```
axes[0,0].hist(msd_b,label='rs_b mean',color='blue')
axes[0,1].hist(msd_a,label='rs_a mean',color='blue')

axes[1,0].hist(vsd_b,label='rs_b variance',color='red')
axes[1,1].hist(vsd_a,label='rs_a variance',color='red')

axes[2,0].hist(esd_b,label='rs_b entropy',color='green')
axes[2,1].hist(esd_a,label='rs_a entropy',color='green')

axes[5,0].hist(sksd_b,label='rs_b skewness',color='pink')
axes[5,1].hist(sksd_a,label='rs_a skewness',color='pink')

axes[6,0].hist(ksd_b,label='rs_b kurtosis',color='purple')
axes[6,1].hist(ksd_a,label='rs_a kurtosis',color='purple')

axes[3,0].hist(vvsd_b,label='rs_b varvalue',color='brown')
axes[3,1].hist(vvsd_a,label='rs_a varvalue',color='brown')

axes[4,0].hist(ssd_b,label='rs_b semvalue')
axes[4,1].hist(ssd_a,label='rs_a semvalue')

for ax in axes.flat:
    ax.set(xlabel='statistical Values', ylabel='Frequency')
    ax.legend()

# Adjust layout
plt.tight_layout(rect=[0, 0.05, 0.5, 0.95])

# Show the plot
plt.show()
```

3.4 14 Statistical Features as Input for All Classifiers

```
#blb disease

#before trans
blb_msd_b=rc.fit_transform(np.array(blb_mean_list).reshape(-1,1))
blb_vsd_b=rc.fit_transform(np.array(blb_variance_list).reshape(-1,1))
blb_esd_b=rc.fit_transform(np.array(blb_entropy_list).reshape(-1,1))
blb_sksd_b=rc.fit_transform(np.array(blb_skewness_list).reshape(-1,1))
blb_ksd_b=rc.fit_transform(np.array(blb_kurtosis_list).reshape(-1,1))
blb_vvsd_b=rc.fit_transform(np.array(blb_varvalue_list).reshape(-1,1))
blb_ssd_b=rc.fit_transform(np.array(blb_sem_list).reshape(-1,1))


#after trans
blb_msd_a=rc.fit_transform(blb_mean_list_transformed.reshape(-1,1))
blb_vsd_a=rc.fit_transform(blb_variance_list_transformed.reshape(-1,1))
blb_esd_a=rc.fit_transform(blb_entropy_list_transformed.reshape(-1,1))
blb_sksd_a=rc.fit_transform(blb_skewness_list_transformed.reshape(-1,1))
blb_ksd_a=rc.fit_transform(blb_kurtosis_list_transformed.reshape(-1,1))
blb_vvsd_a=rc.fit_transform(blb_varvalue_list_transformed.reshape(-1,1))
blb_ssd_a=rc.fit_transform(blb_sem_list_transformed.reshape(-1,1))
```

```
#before transformation-robust scaling

blb_msdl_b = []
for s in blb_msd_b:
    blb_msdl_b.extend(s)

blb_vsdl_b = []
for s in blb_vsd_b:
    blb_vsdl_b.extend(s)

blb_esdl_b = []
for s in blb_esd_b:
    blb_esdl_b.extend(s)

blb_sksdl_b = []
for s in blb_sksd_b:
    blb_sksdl_b.extend(s)

blb_ksdl_b = []
for s in blb_ksd_b:
    blb_ksdl_b.extend(s)

blb_vvsdl_b = []
for s in blb_vvsd_b:
    blb_vvsdl_b.extend(s)

blb_ssdl_b = []
for s in blb_ssd_b:
    blb_ssdl_b.extend(s)
```

```python
#after transformation-robust scaling

blb_msdl_a=[]
for s in blb_msd_a:
    blb_msdl_a.extend(s)

blb_vsdl_a=[]
for s in blb_vsd_a:
    blb_vsdl_a.extend(s)

blb_esdl_a=[]
for s in blb_esd_a:
    blb_esdl_a.extend(s)

blb_sksdl_a=[]
for s in blb_sksd_a:
    blb_sksdl_a.extend(s)

blb_ksdl_a=[]
for s in blb_ksd_a:
    blb_ksdl_a.extend(s)

blb_vvsdl_a=[]
for s in blb_vvsd_a:
    blb_vvsdl_a.extend(s)

blb_ssdl_a=[]
for s in blb_ssd_a:
    blb_ssdl_a.extend(s)
```

```python
msd1=np.array(blb_msdl_b)
vsd1=np.array(blb_vsdl_b)
esd1=np.array(blb_esdl_b)
sksd1=np.array(blb_sksdl_b)
ksd1=np.array(blb_ksdl_b)
vvsd1=np.array(blb_vvsdl_b)
ssd1=np.array(blb_ssdl_b)


msd11=np.array(blb_msdl_a)
vsd11=np.array(blb_vsdl_a)
esd11=np.array(blb_esdl_a)
sksd11=np.array(blb_sksdl_a)
ksd11=np.array(blb_ksdl_a)
vvsd11=np.array(blb_vvsdl_a)
ssd11=np.array(blb_ssdl_a)
```

```python
msd1=msd1.reshape(-1,1)
vsd1=vsd1.reshape(-1,1)
esd1=esd1.reshape(-1,1)
sksd1=sksd1.reshape(-1,1)
ksd1=ksd1.reshape(-1,1)
ssd1=ssd1.reshape(-1,1)
vvsd1=vvsd1.reshape(-1,1)


msd11=msd11.reshape(-1,1)
vsd11=vsd11.reshape(-1,1)
esd11=esd11.reshape(-1,1)
sksd11=sksd11.reshape(-1,1)
ksd11=ksd11.reshape(-1,1)
ssd11=ssd11.reshape(-1,1)
vvsd11=vvsd11.reshape(-1,1)
```

```python
X_blb = np.hstack((msd1,vsd1,esd1,sksd1,ksd1,vvsd1,ssd1,msd11,vsd11,esd11,sksd11,ksd11,vvsd11,ssd11))
y_blb = np.full((len(sksd1),), "BLB")
```

#similarly do we have done for all diseases namely, (hisp, lf, ls, blt, hlt)

```python
#X=np.concatenate((X_blb,X_hispa,X_lf,X_ls,X_blt,X_hlt),axis=0)
#y=np.concatenate((y_blb,y_hispa,y_lf,y_ls,y_blt,y_hlt),axis=0)


# Create an instance of StratifiedShuffleSplit
sss = StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=42)

# Create an instance of StratifiedKFold for k-fold cross-validation
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Assuming X and y are your data and target labels
X = np.concatenate((X_blb, X_hispa, X_lf, X_ls, X_blt, X_hlt), axis=0)
y = np.concatenate((y_blb, y_hispa, y_lf, y_ls, y_blt, y_hlt), axis=0)
```

```python
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Stratified Shuffle Split
for train_index, test_index in sss.split(X, y):
    X_train_nor, X_test_nor = X[train_index], X[test_index]
    y_train_nor, y_test_nor = y[train_index], y[test_index]
    # Use X_train, X_test, y_train, y_test for model training and evaluation

# K-Fold Cross-Validation
for train_index, test_index in skf.split(X, y):
    X_train_nor, X_test_nor = X[train_index], X[test_index]
    y_train_nor, y_test_nor = y[train_index], y[test_index]
    # Use X_train, X_test, y_train, y_test for model training and evaluation
```

```python
model=KNeighborsClassifier()

from sklearn.preprocessing import StandardScaler

se=StandardScaler()
X_train_scaled_nor=se.fit_transform(X_train_nor)
X_test_scaled_nor=se.fit_transform(X_test_nor)

model.fit(X_train_nor,y_train_nor)

y_pred_nor_knn=model.predict(X_test_nor)

accuracy = accuracy_score(y_pred_nor_knn,y_test_nor)
print("accuracy : ",accuracy)

print(classification_report(y_pred_nor_knn,y_test_nor,zero_division=0))
bac_knn = balanced_accuracy_score(y_test_nor, y_pred_nor_knn)

# Calculate MCC
mcc_knn = matthews_corrcoef(y_test_nor, y_pred_nor_knn)
print("Balanced Accuracy:", bac_knn)
print("Matthews Correlation Coefficient:", mcc_knn)
```

```python
rec0=recall_score(y_test_nor,y_pred_nor_knn,average=None)
prec0=precision_score(y_test_nor,y_pred_nor_knn,average=None,zero_division=1.0)
f10=f1_score(y_test_nor,y_pred_nor_knn,average=None)
```

```python
# Create the Linear Discriminant Analysis classifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
model= LinearDiscriminantAnalysis()# Choose a suitable solver

# Train the model on the training data
model.fit(X_train_nor, y_train_nor)

# Make predictions on the testing data
y_pred_nor_lda = model.predict(X_test_nor)
bac_lda = balanced_accuracy_score(y_test_nor, y_pred_nor_lda)

# Calculate MCC
mcc_lda = matthews_corrcoef(y_test_nor, y_pred_nor_lda)

# Evaluate the model performance
accuracy = accuracy_score(y_test_nor, y_pred_nor_lda)
print("Accuracy:", accuracy)
print(classification_report(y_pred_nor_lda,y_test_nor,zero_division=0))
print("Balanced Accuracy:", bac_lda)
print("Matthews Correlation Coefficient:", mcc_lda)
```

```python
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(X_train_nor, y_train_nor)

# Make predictions on the testing data
y_pred_nor_rfc = model.predict(X_test_nor)
bac_rfc = balanced_accuracy_score(y_test_nor, y_pred_nor_rfc)

# Calculate MCC
mcc_rfc = matthews_corrcoef(y_test_nor, y_pred_nor_rfc)

# Evaluate the model performance
accuracy = accuracy_score(y_test_nor, y_pred_nor_rfc)
print("Accuracy:", accuracy)
print(classification_report(y_pred_nor_rfc,y_test_nor,zero_division=0))
print("Balanced Accuracy:", bac_rfc)
print("Matthews Correlation Coefficient:", mcc_rfc)
```

```python
from sklearn.ensemble import HistGradientBoostingClassifier
model = HistGradientBoostingClassifier()

# Train the model on the training data
model.fit(X_train_nor, y_train_nor)

# Make predictions on the testing data
y_pred_nor_hgbc = model.predict(X_test_nor)
bac_hgbc = balanced_accuracy_score(y_test_nor, y_pred_nor_hgbc)

# Calculate MCC
mcc_hgbc = matthews_corrcoef(y_test_nor, y_pred_nor_hgbc)

# Evaluate the model performance
accuracy = accuracy_score(y_test_nor, y_pred_nor_hgbc)
print("Accuracy:", accuracy)
print(classification_report(y_pred_nor_hgbc,y_test_nor,zero_division=0))
print("Balanced Accuracy:", bac_hgbc)
print("Matthews Correlation Coefficient:", mcc_hgbc)
```

3.5 Implementation of **Lemurs Optimization Algorithm(LOA)**

```
#Population Initialization Matrix

def initialize_population(population_size,num_features,features):
    population = np.zeros((population_size,num_features))
    min_features = []
    for k in range(num_features):
        min_features.append(min(features[k]))

    for i in range(population_size):
        for j in range(num_features):
            rand_value = np.random.rand()
            population[i,j] = rand_value * ((max(features[j]) - min(features[j])) + min(min_features))
    return population,min_features


population_size_blb = 219
population_size_hisp = 141
population_size_lf = 33
population_size_ls = 79
population_size_blt = 66
population_size_hlt = 92
num_features = 14

features_blb = [blb_msdl_b,blb_vsdl_b,blb_esdl_b,blb_sksdl_b,blb_ksdl_b,blb_vvsdl_b,blb_ssdl_b,blb_msdl_a,blb_vsdl_a,blb_esdl_a,blb_sksdl_a,bl
features_hisp = [hisp_msdl_b,hisp_vsdl_b,hisp_esdl_b,hisp_sksdl_b,hisp_ksdl_b,hisp_vvsdl_b,hisp_ssdl_b,hisp_msdl_a,hisp_vsdl_a,hisp_esdl_a,his
features_lf = [lf_msdl_b,lf_vsdl_b,lf_esdl_b,lf_sksdl_b,lf_ksdl_b,lf_vvsdl_b,lf_ssdl_b,lf_msdl_a,lf_vsdl_a,lf_esdl_a,lf_sksdl_a,lf_ksdl_a,lf_v
features_ls = [ls_msdl_b,ls_vsdl_b,ls_esdl_b,ls_sksdl_b,ls_ksdl_b,ls_vvsdl_b,ls_ssdl_b,ls_msdl_a,ls_vsdl_a,ls_esdl_a,ls_sksdl_a,ls_ksdl_a,ls_v
features_blt = [blt_msdl_b,blt_vsdl_b,blt_esdl_b,blt_sksdl_b,blt_ksdl_b,blt_vvsdl_b,blt_ssdl_b,blt_msdl_a,blt_vsdl_a,blt_esdl_a,blt_sksdl_a,bl
features_hlt = [hlt_msdl_b,hlt_vsdl_b,hlt_esdl_b,hlt_sksdl_b,hlt_ksdl_b,hlt_vvsdl_b,hlt_ssdl_b,hlt_msdl_a,hlt_vsdl_a,hlt_esdl_a,hlt_sksdl_a,hl

population_blb,min_feature_blb = initialize_population(population_size_blb,num_features,features_blb)
population_hisp,min_feature_hisp = initialize_population(population_size_hisp,num_features,features_hisp)
population_lf,min_feature_lf = initialize_population(population_size_lf,num_features,features_lf)
population_ls,min_feature_ls = initialize_population(population_size_ls,num_features,features_ls)
population_blt,min_feature_blt = initialize_population(population_size_blt,num_features,features_blt)
population_hlt,min_feature_hlt = initialize_population(population_size_hlt,num_features,features_hlt)
```

```
def combine_disease_features(features,population):
    combined_features = []
    for i in range(len(population)):
        l=[]
        for j in range(len(features)):
            l.append(features[j][i])
        combined_features.append(l)
    return combined_features

combined_features_blb = combine_disease_features(features_blb,population_blb)
combined_features_hisp = combine_disease_features(features_hisp,population_hisp)
combined_features_lf = combine_disease_features(features_lf,population_lf)
combined_features_ls = combine_disease_features(features_ls,population_ls)
combined_features_blt = combine_disease_features(features_blt,population_blt)
combined_features_hlt = combine_disease_features(features_hlt,population_hlt)
```

```
def fitness_function(features):
    """
    Calculates the fitness value based on the given features.

    Args:
        features: Features represented by the Lemur.

    Returns:`
        fitness_value: Fitness value for the Lemur.
    """
    # Replace this with your actual fitness function
    # For demonstration purposes, let's just sum the features
    fitness_value = np.var(features)
    return fitness_value
```

```python
def calculate_fitness_values(population, combined_features):
    """
    Calculates the fitness value for each Lemur in the population.

    Args:
        population: A NumPy array representing the current population.
        combined_features: A list containing the combined features for each Lemur.

    Returns:
        fitness_values: A list containing the fitness value for each Lemur.
    """
    fitness_values = []

    for i in range(len(population)):
        features = []
        if i == 0 :
            features.extend(combined_features[(len(population)-1)])
            features.extend(combined_features[i])
            features.extend(combined_features[i+1])
        elif i == len(population) - 1 :
            features.extend(combined_features[i-1])
            features.extend(combined_features[i])
            features.extend(combined_features[0])
        else :
            features.extend(combined_features[i-1])
            features.extend(combined_features[i])
            features.extend(combined_features[i+1])

        fitness_value = fitness_function(features)  # Calculate the fitness value
        fitness_values.append(fitness_value)
    return fitness_values
```

```python
def compute_frr(HRR, LRR, current_iter, max_iter):
    """
    Computes the Free Risk Rate (FRR) based on the given parameters.

    Args:
        HRR: High-Risk Rate (constant predefined value).
        LRR: Low-Risk Rate (constant predefined value).
        current_iter: Current iteration number.
        max_iter: Maximum number of iterations.

    Returns:
        FRR: Free Risk Rate.
    """
    FRR = HRR - current_iter * ((HRR - LRR) / max_iter)
    return FRR
```

```python
def global_best_lemurs(population, fitness_values):
    """
    Finds the global best Lemur and the indices of the top 2 nearest neighbors.

    Args:
        population: A NumPy array representing the current population.
        fitness_values: A list containing the fitness values of each Lemur.

    Returns:
        gbest: Index of the Lemur with the best fitness value.
        nbest: Indices of the top 2 nearest neighbors of gbest.
    """
    gbest = np.argmax(fitness_values)
    return gbest
```

```python
def update_positions(population, gbest, FRR, fitness_values):
    """
    Updates the positions (selected features) of Lemurs in the population.

    Args:
        population: A NumPy array representing the current population.
        gbest: Global best Lemur position.
        nbest: Best nearest Lemur position.
        FRR: Free Risk Rate.

    Returns:
        The updated population array.
    """
    nbest_print = []
    for i in range(len(population)):
        rand = np.random.rand()
        find_nbest = []
        if i == 0 :
            find_nbest.append(fitness_values[(len(population)-1)])
            find_nbest.append(fitness_values[i])
            find_nbest.append(fitness_values[i+1])

            nbest_array = np.array(find_nbest)
            nbest_index = np.argmax(nbest_array)

            if nbest_index == 0 :
                nbest = (len(population) - 1)
            elif nbest_index == 1 :
                nbest = 0
            elif nbest_index == 2 :
                nbest = i+1
```

```python
        elif i == (len(population) - 1) :
            find_nbest.append(fitness_values[i-1])
            find_nbest.append(fitness_values[i])
            find_nbest.append(fitness_values[0])

            nbest_array = np.array(find_nbest)
            nbest_index = np.argmax(nbest_array)

            if nbest_index == 0 :
                nbest = i-1
            elif nbest_index == 1 :
                nbest = i
            elif nbest_index == 2 :
                nbest = 0

        else :
            find_nbest.append(fitness_values[i-1])
            find_nbest.append(fitness_values[i])
            find_nbest.append(fitness_values[i+1])

            nbest_array = np.array(find_nbest)
            nbest_index = np.argmax(nbest_array)

            if nbest_index == 0 :
                nbest = i-1
            elif nbest_index == 1 :
                nbest = i
            elif nbest_index == 2 :
                nbest = i+1
```

```python
        nbest_print.append(nbest)
        for j in range(len(population[i])):
            if rand >= FRR:
                population[i][j] = population[i][j] + np.abs(population[i][j] - population[gbest][j]) * (rand - 0.5) * 2
            else:
                population[i][j] = population[i][j] + np.abs(population[i][j] - population[nbest][j]) * (rand - 0.5) * 2

    return population,nbest_print
```

```python
current_iter = 1
max_iter = 5
HRR = 0.9
LRR = 0.2

while current_iter <= max_iter:

    fitness_values_blb = calculate_fitness_values(population_blb, combined_features_blb)
    fitness_values_hisp = calculate_fitness_values(population_hisp, combined_features_hisp)
    fitness_values_lf = calculate_fitness_values(population_lf, combined_features_lf)
    fitness_values_ls = calculate_fitness_values(population_ls, combined_features_ls)
    fitness_values_blt = calculate_fitness_values(population_blt, combined_features_blt)
    fitness_values_hlt = calculate_fitness_values(population_hlt, combined_features_hlt)



    FRR = compute_frr(HRR, LRR, current_iter, max_iter)


    gbest_blb = global_best_lemurs(population_blb, fitness_values_blb)
    gbest_hisp = global_best_lemurs(population_hisp, fitness_values_hisp)
    gbest_lf = global_best_lemurs(population_lf, fitness_values_lf)
    gbest_ls = global_best_lemurs(population_ls, fitness_values_ls)
    gbest_blt = global_best_lemurs(population_blt, fitness_values_blt)
    gbest_hlt = global_best_lemurs(population_hlt, fitness_values_hlt)


    population_blb,nbest_blb = update_positions(population_blb,gbest_blb,FRR,fitness_values_blb)
    population_hisp,nbest_hisp = update_positions(population_hisp,gbest_hisp,FRR,fitness_values_hisp)
    population_lf,nbest_lf = update_positions(population_lf,gbest_lf,FRR,fitness_values_lf)
    population_ls,nbest_ls = update_positions(population_ls,gbest_ls,FRR,fitness_values_ls)
    population_blt,nbest_blt = update_positions(population_blt,gbest_blt,FRR,fitness_values_blt)
    population_hlt,nbest_hlt = update_positions(population_hlt,gbest_hlt,FRR,fitness_values_hlt)

    current_iter = current_iter + 1
```

```python
#blb disease

#before trans-robust
loa_blb_mean_b = []
for i in range(len(population_blb)):
    loa_blb_mean_b.append(population_blb[i][0])

loa_blb_variance_b = []
for i in range(len(population_blb)):
    loa_blb_variance_b.append(population_blb[i][1])

loa_blb_entropy_b = []
for i in range(len(population_blb)):
    loa_blb_entropy_b.append(population_blb[i][2])

loa_blb_skewness_b = []
for i in range(len(population_blb)):
    loa_blb_skewness_b.append(population_blb[i][3])

loa_blb_kurtosis_b = []
for i in range(len(population_blb)):
    loa_blb_kurtosis_b.append(population_blb[i][4])

loa_blb_varvalue_b = []
for i in range(len(population_blb)):
    loa_blb_varvalue_b.append(population_blb[i][5])

loa_blb_sem_b = []
for i in range(len(population_blb)):
    loa_blb_sem_b.append(population_blb[i][6])
```

```
#after trans-robust
loa_blb_mean_a = []
for i in range(len(population_blb)):
    loa_blb_mean_a.append(population_blb[i][7])

loa_blb_variance_a = []
for i in range(len(population_blb)):
    loa_blb_variance_a.append(population_blb[i][8])

loa_blb_entropy_a = []
for i in range(len(population_blb)):
    loa_blb_entropy_a.append(population_blb[i][9])

loa_blb_skewness_a = []
for i in range(len(population_blb)):
    loa_blb_skewness_a.append(population_blb[i][10])

loa_blb_kurtosis_a = []
for i in range(len(population_blb)):
    loa_blb_kurtosis_a.append(population_blb[i][11])

loa_blb_varvalue_a = []
for i in range(len(population_blb)):
    loa_blb_varvalue_a.append(population_blb[i][12])

loa_blb_sem_a = []
for i in range(len(population_blb)):
    loa_blb_sem_a.append(population_blb[i][13])
```

```
#coverting lists to arrays
loa_msd1=np.array(loa_blb_mean_b)
loa_vsd1=np.array(loa_blb_variance_b)
loa_esd1=np.array(loa_blb_entropy_b)
loa_sksd1=np.array(loa_blb_skewness_b)
loa_ksd1=np.array(loa_blb_kurtosis_b)
loa_vvsd1=np.array(loa_blb_varvalue_b)
loa_ssd1=np.array(loa_blb_sem_b)


loa_msd11=np.array(loa_blb_mean_a)
loa_vsd11=np.array(loa_blb_variance_a)
loa_esd11=np.array(loa_blb_entropy_a)
loa_sksd11=np.array(loa_blb_skewness_a)
loa_ksd11=np.array(loa_blb_kurtosis_a)
loa_vvsd11=np.array(loa_blb_varvalue_a)
loa_ssd11=np.array(loa_blb_sem_a)


#reshapinf arrays
loa_msd1=loa_msd1.reshape(-1,1)
loa_vsd1=loa_vsd1.reshape(-1,1)
loa_esd1=loa_esd1.reshape(-1,1)
loa_sksd1=loa_sksd1.reshape(-1,1)
loa_ksd1=loa_ksd1.reshape(-1,1)
loa_ssd1=loa_ssd1.reshape(-1,1)
loa_vvsd1=loa_vvsd1.reshape(-1,1)


loa_msd11=loa_msd11.reshape(-1,1)
loa_vsd11=loa_vsd11.reshape(-1,1)
loa_esd11=loa_esd11.reshape(-1,1)
loa_sksd11=loa_sksd11.reshape(-1,1)
loa_ksd11=loa_ksd11.reshape(-1,1)
loa_ssd11=loa_ssd11.reshape(-1,1)
loa_vvsd11=loa_vvsd11.reshape(-1,1)
```

```
X_loa_blb = np.hstack((loa_msd1,loa_vsd1,loa_esd1,loa_sksd1,loa_ksd1,loa_vvsd1,loa_ssd1,loa_msd11,loa_vsd11,loa_esd11,loa_sksd11,loa_ksd11,loa
y_loa_blb = np.full((len(loa_sksd1),), "BLB")
```

```python
#X_loa=np.concatenate((X_loa_blb,X_loa_hisp,X_loa_lf,X_loa_ls,X_loa_blt,X_loa_hlt),axis=0)
#y_loa=np.concatenate((y_loa_blb,y_loa_hisp,y_loa_lf,y_loa_ls,y_loa_blt,y_loa_hlt),axis=0)


# Create an instance of StratifiedShuffleSplit
sss = StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=42)

# Create an instance of StratifiedKFold for k-fold cross-validation
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Assuming X and y are your data and target labels
X = np.concatenate((X_loa_blb, X_loa_hisp, X_loa_lf, X_loa_ls, X_loa_blt, X_loa_hlt), axis=0)
y = np.concatenate((y_loa_blb, y_loa_hisp, y_loa_lf, y_loa_ls, y_loa_blt, y_loa_hlt), axis=0)
```

```python
#X_train, X_test, y_train, y_test = train_test_split(X_loa, y_loa, test_size=0.2, random_state=42)

#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Stratified Shuffle Split
for train_index, test_index in sss.split(X, y):
    X_train_loa, X_test_loa = X[train_index], X[test_index]
    y_train_loa, y_test_loa = y[train_index], y[test_index]
    # Use X_train, X_test, y_train, y_test for model training and evaluation

# K-Fold Cross-Validation
for train_index, test_index in skf.split(X, y):
    X_train_loa, X_test_loa = X[train_index], X[test_index]
    y_train_loa, y_test_loa = y[train_index], y[test_index]
    # Use X_train, X_test, y_train, y_test for model training and evaluation
```

```python
model=KNeighborsClassifier()

from sklearn.preprocessing import StandardScaler

se=StandardScaler()
X_train_scaled_loa=se.fit_transform(X_train_loa)
X_test_scaled_loa=se.fit_transform(X_test_loa)

model.fit(X_train_scaled_loa,y_train_loa)

y_pred_loa_knn=model.predict(X_test_scaled_loa)

print(accuracy_score(y_pred_loa_knn,y_test_loa))

print(classification_report(y_pred_loa_knn,y_test_loa,zero_division=0))
bac_loa_knn = balanced_accuracy_score(y_test_loa, y_pred_loa_knn)

# Calculate MCC
mcc_loa_knn = matthews_corrcoef(y_test_loa, y_pred_loa_knn)
print("Balanced Accuracy:", bac_loa_knn)
print("Matthews Correlation Coefficient:", mcc_loa_knn)
```

```python
rec2=recall_score(y_test_loa,y_pred_loa_knn,average=None)
prec2=precision_score(y_test_loa,y_pred_loa_knn,average=None,zero_division=1.0)
f12=f1_score(y_test_loa,y_pred_loa_knn,average=None)
```

```python
# Create the Linear Discriminant Analysis classifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
model= LinearDiscriminantAnalysis()# Choose a suitable solver

# Train the model on the training data
model.fit(X_train_scaled_loa, y_train_loa)

# Make predictions on the testing data
y_pred_loa_lda = model.predict(X_test_scaled_loa)
bac_loa_lda = balanced_accuracy_score(y_test_loa, y_pred_loa_lda)

# Calculate MCC
mcc_loa_lda = matthews_corrcoef(y_test_loa, y_pred_loa_lda)

# Evaluate the model performance
accuracy = accuracy_score(y_test_loa, y_pred_loa_lda)
print("Accuracy:", accuracy)
print(classification_report(y_pred_loa_lda,y_test_loa,zero_division=0))
print("Balanced Accuracy:", bac_loa_lda)
print("Matthews Correlation Coefficient:", mcc_loa_lda)
```

```python
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(X_train_scaled_loa, y_train_loa)

# Make predictions on the testing data
y_pred_loa_rfc = model.predict(X_test_scaled_loa)
bac_loa_rfc = balanced_accuracy_score(y_test_loa, y_pred_loa_rfc)

# Calculate MCC
mcc_loa_rfc = matthews_corrcoef(y_test_loa, y_pred_loa_rfc)

# Evaluate the model performance
accuracy = accuracy_score(y_test_loa, y_pred_loa_rfc)
print("Accuracy:", accuracy)
print(classification_report(y_pred_loa_rfc,y_test_loa,zero_division=0))
print("Balanced Accuracy:", bac_loa_rfc)
print("Matthews Correlation Coefficient:", mcc_loa_rfc)
```

```python
from sklearn.ensemble import HistGradientBoostingClassifier
model = HistGradientBoostingClassifier()

# Train the model on the training data
model.fit(X_train_scaled_loa, y_train_loa)

# Make predictions on the testing data
y_pred_loa_hgbc = model.predict(X_test_scaled_loa)
bac_loa_hgbc = balanced_accuracy_score(y_test_loa, y_pred_loa_hgbc)

# Calculate MCC
mcc_loa_hgbc = matthews_corrcoef(y_test_loa, y_pred_loa_hgbc)

# Evaluate the model performance
accuracy = accuracy_score(y_test_loa, y_pred_loa_hgbc)
print("Accuracy:", accuracy)
print(classification_report(y_pred_loa_hgbc,y_test_loa,zero_division=0))
print("Balanced Accuracy:", bac_loa_hgbc)
print("Matthews Correlation Coefficient:", mcc_loa_hgbc)
```

3.6 Implementation of **Modified Lemurs Optimization Algorithm(MLOA)**

```python
import numpy as np

# Function to initialize population
def initialize_population(population_size, num_features, features):
    population = np.zeros((population_size, num_features))

    for i in range(len(population)):
        for j in range(num_features):
            population[i, j] = features[j][i]

    return population

# Function to calculate fitness value
def fitness_function(features):
    return np.var(features)
```

```python
def calculate_fitness_values(population):
    """
    Calculates the fitness value for each Lemur in the population.

    Args:
        population: A NumPy array representing the current population.
        combined_features: A list containing the combined features for each Lemur.

    Returns:
        fitness_values: A list containing the fitness value for each Lemur.
    """
    features = []
    pop_1 = []
    for i in range(len(population)):
        pop_1.append(population[i][0])
        features.append(pop_1)

    pop_2 = []
    for i in range(len(population)):
        pop_2.append(population[i][1])
        features.append(pop_2)

    pop_3 = []
    for i in range(len(population)):
        pop_3.append(population[i][2])
        features.append(pop_3)

    pop_4 = []
    for i in range(len(population)):
        pop_4.append(population[i][3])
        features.append(pop_4)
```

```
pop_5 = []
for i in range(len(population)):
    pop_5.append(population[i][4])
    features.append(pop_5)

pop_6 = []
for i in range(len(population)):
    pop_6.append(population[i][5])
    features.append(pop_6)

pop_7 = []
for i in range(len(population)):
    pop_7.append(population[i][6])
    features.append(pop_7)

pop_8 = []
for i in range(len(population)):
    pop_8.append(population[i][7])
    features.append(pop_8)

pop_9 = []
for i in range(len(population)):
    pop_9.append(population[i][8])
    features.append(pop_9)

pop_10 = []
for i in range(len(population)):
    pop_10.append(population[i][9])
    features.append(pop_10)

pop_11 = []
for i in range(len(population)):
    pop_11.append(population[i][10])
    features.append(pop_11)
```

```
pop_12 = []
for i in range(len(population)):
    pop_12.append(population[i][11])
    features.append(pop_12)

pop_13 = []
for i in range(len(population)):
    pop_13.append(population[i][12])
    features.append(pop_13)

pop_14 = []
for i in range(len(population)):
    pop_14.append(population[i][13])
    features.append(pop_14)

fitness_values_all = []
num_features = 14
for i in range(num_features):
    features_1 = []
    if i == 0 :
        features_1.extend(features[13])
        features_1.extend(features[i])
        features_1.extend(features[i+1])
    elif i == 13 :
        features_1.extend(features[i-1])
        features_1.extend(features[i])
        features_1.extend(features[0])
    else :
        features_1.extend(features[i-1])
        features_1.extend(features[i])
        features_1.extend(features[i+1])

    fitness_value = fitness_function(features_1)  # Calculate the fitness value

    fitness_values_all.append(fitness_value)

return fitness_values_all
```

```python
def global_best_lemurs(population, fitness_values):
    """
    Finds the global best Lemur and the indices of the top 2 nearest neighbors.

    Args:
        population: A NumPy array representing the current population.
        fitness_values: A list containing the fitness values of each Lemur.

    Returns:
        gbest: Index of the Lemur with the best fitness value.
        nbest: Indices of the top 2 nearest neighbors of gbest.
    """
    num_features = 14
    nbest_print = []

    gbest = np.argmax(fitness_values)

    for i in range(num_features):
        find_nbest = []
        if i == 0 :
            find_nbest.append(fitness_values[13])
            find_nbest.append(fitness_values[i])
            find_nbest.append(fitness_values[i+1])

            nbest_array = np.array(find_nbest)
            nbest_index = np.argmax(nbest_array)
            #print(find_nbest)
            #print(nbest_array)
            #print(nbest_index)
            if nbest_index == 0 :
                nbest = 13
            elif nbest_index == 1 :
                nbest = i
            elif nbest_index == 2 :
                nbest = i+1
```

```python
        elif i == 13 :
            find_nbest.append(fitness_values[i-1])
            find_nbest.append(fitness_values[i])
            find_nbest.append(fitness_values[0])


            nbest_array = np.array(find_nbest)
            nbest_index = np.argmax(nbest_array)

            if nbest_index == 0 :
                nbest = i-1
            elif nbest_index == 1 :
                nbest = i
            elif nbest_index == 2 :
                nbest = 0


        else :
            find_nbest.append(fitness_values[i-1])
            find_nbest.append(fitness_values[i])
            find_nbest.append(fitness_values[i+1])

            nbest_array = np.array(find_nbest)
            nbest_index = np.argmax(nbest_array)

            if nbest_index == 0 :
                nbest = i-1
            elif nbest_index == 1 :
                nbest = i
            elif nbest_index == 2 :
                nbest = i+1

        nbest_print.append(nbest)
    return gbest,nbest_print
```

```python
def worst_lemur(population, fitness_values):

    rand = np.random.rand()
    L_worst = np.argmin(fitness_values)

    for i in range(len(population)):
        Lmax = max(population[i])
        Lmin = min(population[i])
        L_new = Lmin + (Lmax - Lmin) * rand

        population[i][L_worst] = L_new

    return population

# Function to update positions of Lemurs
def update_positions(population, gbest_indices, nbest_indices, FRR, current_iter, max_iter):
    a = 3
    r1 = a - current_iter * (a / max_iter)

    for j in range(14):
        rand = np.random.rand()
        for i in range(len(population)):

            if FRR >= 0.5:
                population[i][j] += r1 * (np.sin(np.abs(population[i][j] - population[i][gbest_indices]))) * (rand - 0.5) * 2
            else:
                population[i][j] += r1 * (np.cos(np.abs(population[i][j] - population[i][nbest_indices[j]]))) * (rand - 0.5) * 2

    return population
```

```python
def compute_frr(current_iter, population, Lrate, FRR_list,Ivar_list):

    features_1 = []
    features = []
    pop_1 = []
    for i in range(len(population)):
        pop_1.append(population[i][0])
        features.append(pop_1)

    pop_2 = []
    for i in range(len(population)):
        pop_2.append(population[i][1])
        features.append(pop_2)

    pop_3 = []
    for i in range(len(population)):
        pop_3.append(population[i][2])
        features.append(pop_3)

    pop_4 = []
    for i in range(len(population)):
        pop_4.append(population[i][3])
        features.append(pop_4)

    pop_5 = []
    for i in range(len(population)):
        pop_5.append(population[i][4])
        features.append(pop_5)

    pop_6 = []
    for i in range(len(population)):
        pop_6.append(population[i][5])
        features.append(pop_6)
```

```
pop_7 = []
for i in range(len(population)):
    pop_7.append(population[i][6])
    features.append(pop_7)

pop_8 = []
for i in range(len(population)):
    pop_8.append(population[i][7])
    features.append(pop_8)

pop_9 = []
for i in range(len(population)):
    pop_9.append(population[i][8])
    features.append(pop_9)

pop_10 = []
for i in range(len(population)):
    pop_10.append(population[i][9])
    features.append(pop_10)

pop_11 = []
for i in range(len(population)):
    pop_11.append(population[i][10])
    features.append(pop_11)

pop_12 = []
for i in range(len(population)):
    pop_12.append(population[i][11])
    features.append(pop_12)

pop_13 = []
for i in range(len(population)):
    pop_13.append(population[i][12])
    features.append(pop_13)
```

```
pop_14 = []
for i in range(len(population)):
    pop_14.append(population[i][13])
    features.append(pop_14)

FRR_list_1 = []
Ivar_list_1 = []

for i in range(14):
    features_1.extend(features[i])
    |
Ivar = 1 / ( np.var(features_1))



if current_iter == 1:
    loss = Ivar / FRR_list[current_iter - 1]
else:
    loss = (Ivar - Ivar_list[current_iter - 2]) / (FRR_list[current_iter - 1] - FRR_list[current_iter - 2])

FRR = FRR_list[current_iter - 1] - Lrate * loss

return FRR,Ivar
```

```python
def run_mloa(population_size, num_features, features):
    # Initialize population
    population = initialize_population(population_size, num_features, features)
    max_iter = 12
    Lrate = 0.5
    FRR = 0.5
    FRR_list = []
    Ivar_list = []

    for current_iter in range(1, max_iter + 1):
        # Calculate fitness values
        fitness_values = calculate_fitness_values(population)

        # Find global best and top 2 nearest neighbors
        gbest_indices, nbest_indices = global_best_lemurs(population, fitness_values)
        #print(gbest_indices)
        #print(nbest_indices)
        print("Currrnt : ",current_iter)
        # Update positions
        population = update_positions(population, gbest_indices, nbest_indices, FRR, current_iter, max_iter)
        # Compute FRR
        FRR_list.append(FRR)
        FRR,Ivar = compute_frr(current_iter, population, Lrate, FRR_list,Ivar_list)
        Ivar_list.append(Ivar)

        population = worst_lemur(population, fitness_values)
        print(population)

    print(FRR_list)
    print(Ivar_list)

    return population
```

```python
# Example usage
population_size_blb = 219
population_size_hisp = 141
population_size_lf = 33
population_size_ls = 79
population_size_blt = 66
population_size_hlt = 92
num_features = 14

features_blb = [blb_msdl_b,blb_vsdl_b,blb_esdl_b,blb_sksdl_b,blb_ksdl_b,blb_vvsdl_b,blb_ssdl_b,blb_msdl_a,blb_vsdl_a,blb_esdl_a,blb_sksd
features_hisp = [hisp_msdl_b,hisp_vsdl_b,hisp_esdl_b,hisp_sksdl_b,hisp_ksdl_b,hisp_vvsdl_b,hisp_ssdl_b,hisp_msdl_a,hisp_vsdl_a,hisp_esdl
features_lf = [lf_msdl_b,lf_vsdl_b,lf_esdl_b,lf_sksdl_b,lf_ksdl_b,lf_vvsdl_b,lf_ssdl_b,lf_msdl_a,lf_vsdl_a,lf_esdl_a,lf_sksdl_a,lf_ksdl_
features_ls = [ls_msdl_b,ls_vsdl_b,ls_esdl_b,ls_sksdl_b,ls_ksdl_b,ls_vvsdl_b,ls_ssdl_b,ls_msdl_a,ls_vsdl_a,ls_esdl_a,ls_sksdl_a,ls_ksdl_
features_blt = [blt_msdl_b,blt_vsdl_b,blt_esdl_b,blt_sksdl_b,blt_ksdl_b,blt_vvsdl_b,blt_ssdl_b,blt_msdl_a,blt_vsdl_a,blt_esdl_a,blt_sksd
features_hlt = [hlt_msdl_b,hlt_vsdl_b,hlt_esdl_b,hlt_sksdl_b,hlt_ksdl_b,hlt_vvsdl_b,hlt_ssdl_b,hlt_msdl_a,hlt_vsdl_a,hlt_esdl_a,hlt_sksd

mloa_population_blb = run_mloa(population_size_blb,num_features,features_blb)
mloa_population_hisp = run_mloa(population_size_hisp,num_features,features_hisp)
mloa_population_lf = run_mloa(population_size_lf,num_features,features_lf)
mloa_population_ls = run_mloa(population_size_ls,num_features,features_ls)
mloa_population_blt = run_mloa(population_size_blt,num_features,features_blt)
mloa_population_hlt = run_mloa(population_size_hlt,num_features,features_hlt)
```

```
#before trans-robust
mloa_blb_mean_b = []
for i in range(len(mloa_population_blb)):
    mloa_blb_mean_b.append(mloa_population_blb[i][0])

mloa_blb_variance_b = []
for i in range(len(mloa_population_blb)):
    mloa_blb_variance_b.append(mloa_population_blb[i][1])

mloa_blb_entropy_b = []
for i in range(len(mloa_population_blb)):
    mloa_blb_entropy_b.append(mloa_population_blb[i][2])

mloa_blb_skewness_b = []
for i in range(len(mloa_population_blb)):
    mloa_blb_skewness_b.append(mloa_population_blb[i][3])

mloa_blb_kurtosis_b = []
for i in range(len(mloa_population_blb)):
    mloa_blb_kurtosis_b.append(mloa_population_blb[i][4])

mloa_blb_varvalue_b = []
for i in range(len(mloa_population_blb)):
    mloa_blb_varvalue_b.append(mloa_population_blb[i][5])

mloa_blb_sem_b = []
for i in range(len(mloa_population_blb)):
    mloa_blb_sem_b.append(mloa_population_blb[i][6])
```

```
#after trans-robust
mloa_blb_mean_a = []
for i in range(len(mloa_population_blb)):
    mloa_blb_mean_a.append(mloa_population_blb[i][7])

mloa_blb_variance_a = []
for i in range(len(mloa_population_blb)):
    mloa_blb_variance_a.append(mloa_population_blb[i][8])

mloa_blb_entropy_a = []
for i in range(len(mloa_population_blb)):
    mloa_blb_entropy_a.append(mloa_population_blb[i][9])

mloa_blb_skewness_a = []
for i in range(len(mloa_population_blb)):
    mloa_blb_skewness_a.append(mloa_population_blb[i][10])

mloa_blb_kurtosis_a = []
for i in range(len(mloa_population_blb)):
    mloa_blb_kurtosis_a.append(mloa_population_blb[i][11])

mloa_blb_varvalue_a = []
for i in range(len(mloa_population_blb)):
    mloa_blb_varvalue_a.append(mloa_population_blb[i][12])

mloa_blb_sem_a = []
for i in range(len(mloa_population_blb)):
    mloa_blb_sem_a.append(mloa_population_blb[i][13])
```

```python
#coverting lists to arrays
mloa_msd1=np.array(mloa_blb_mean_b)
mloa_vsd1=np.array(mloa_blb_variance_b)
mloa_esd1=np.array(mloa_blb_entropy_b)
mloa_sksd1=np.array(mloa_blb_skewness_b)
mloa_ksd1=np.array(mloa_blb_kurtosis_b)
mloa_vvsd1=np.array(mloa_blb_varvalue_b)
mloa_ssd1=np.array(mloa_blb_sem_b)


mloa_msd11=np.array(mloa_blb_mean_a)
mloa_vsd11=np.array(mloa_blb_variance_a)
mloa_esd11=np.array(mloa_blb_entropy_a)
mloa_sksd11=np.array(mloa_blb_skewness_a)
mloa_ksd11=np.array(mloa_blb_kurtosis_a)
mloa_vvsd11=np.array(mloa_blb_varvalue_a)
mloa_ssd11=np.array(mloa_blb_sem_a)


#reshapinf arrays
mloa_msd1=mloa_msd1.reshape(-1,1)
mloa_vsd1=mloa_vsd1.reshape(-1,1)
mloa_esd1=mloa_esd1.reshape(-1,1)
mloa_sksd1=mloa_sksd1.reshape(-1,1)
mloa_ksd1=mloa_ksd1.reshape(-1,1)
mloa_ssd1=mloa_ssd1.reshape(-1,1)
mloa_vvsd1=mloa_vvsd1.reshape(-1,1)


mloa_msd11=mloa_msd11.reshape(-1,1)
mloa_vsd11=mloa_vsd11.reshape(-1,1)
mloa_esd11=mloa_esd11.reshape(-1,1)
mloa_sksd11=mloa_sksd11.reshape(-1,1)
mloa_ksd11=mloa_ksd11.reshape(-1,1)
mloa_ssd11=mloa_ssd11.reshape(-1,1)
mloa_vvsd11=mloa_vvsd11.reshape(-1,1)
```

```python
X_mloa_blb = np.hstack((mloa_msd1,mloa_vsd1,mloa_esd1,mloa_sksd1,mloa_ksd1,mloa_vvsd1,mloa_ssd1,mloa_msd11,mloa_vsd11,mloa_esd11,mloa_sksd11,m
y_mloa_blb = np.full((len(mloa_sksd1),), "BLB")
```

```python
#X_mloa=np.concatenate((X_mloa_blb,X_mloa_hisp,X_mloa_lf,X_mloa_ls,X_mloa_blt,X_mloa_hlt),axis=0)
#y_mloa=np.concatenate((y_mloa_blb,y_mloa_hisp,y_mloa_lf,y_mloa_ls,y_mloa_blt,y_mloa_hlt),axis=0)

# Create an instance of StratifiedShuffleSplit
sss = StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=42)

# Create an instance of StratifiedKFold for k-fold cross-validation
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Assuming X and y are your data and target labels
X = np.concatenate((X_mloa_blb, X_mloa_hisp, X_mloa_lf, X_mloa_ls, X_mloa_blt, X_mloa_hlt), axis=0)
y = np.concatenate((y_mloa_blb, y_mloa_hisp, y_mloa_lf, y_mloa_ls, y_mloa_blt, y_mloa_hlt), axis=0)
```

```python
#X_train, X_test, y_train, y_test = train_test_split(X_loa, y_loa, test_size=0.2, random_state=42)

#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Stratified Shuffle Split
for train_index, test_index in sss.split(X, y):
    X_train_mloa, X_test_mloa = X[train_index], X[test_index]
    y_train_mloa, y_test_mloa = y[train_index], y[test_index]
    # Use X_train, X_test, y_train, y_test for model training and evaluation

# K-Fold Cross-Validation
for train_index, test_index in skf.split(X, y):
    X_train_mloa, X_test_mloa = X[train_index], X[test_index]
    y_train_mloa, y_test_mloa = y[train_index], y[test_index]
    # Use X_train, X_test, y_train, y_test for model training and evaluation
```

```
model=KNeighborsClassifier(n_neighbors = 5)
from sklearn.preprocessing import StandardScaler

se=StandardScaler()
X_train_scaled_mloa=se.fit_transform(X_train_mloa)
X_test_scaled_mloa=se.fit_transform(X_test_mloa)

model.fit(X_train_scaled_mloa,y_train_mloa)

y_pred_mloa_knn=model.predict(X_test_scaled_mloa)

accuracy = accuracy_score(y_pred_mloa_knn,y_test_mloa)

print("accuracy : ",accuracy)

print(classification_report(y_pred_mloa_knn,y_test_mloa,zero_division=0))
bac_mloa_knn = balanced_accuracy_score(y_test_mloa, y_pred_mloa_knn)

# Calculate MCC
mcc_mloa_knn = matthews_corrcoef(y_test_mloa, y_pred_mloa_knn)
print("Balanced Accuracy:", bac_mloa_knn)
print("Matthews Correlation Coefficient:", mcc_mloa_knn)
```

```
rec1=recall_score(y_test_mloa,y_pred_mloa_knn,average=None)
prec1=precision_score(y_test_mloa,y_pred_mloa_knn,average=None)
f11=f1_score(y_test_mloa,y_pred_mloa_knn,average=None)
```

```
# Create the Linear Discriminant Analysis classifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
model= LinearDiscriminantAnalysis()# Choose a suitable solver

# Train the model on the training data
model.fit(X_train_scaled_mloa, y_train_mloa)

# Make predictions on the testing data
y_pred_mloa_lda = model.predict(X_test_scaled_mloa)
bac_mloa_lda = balanced_accuracy_score(y_test_mloa, y_pred_mloa_lda)

# Calculate MCC
mcc_mloa_lda = matthews_corrcoef(y_test_mloa, y_pred_mloa_lda)

# Evaluate the model performance
accuracy = accuracy_score(y_test_mloa, y_pred_mloa_lda)
print("Accuracy:", accuracy)
print(classification_report(y_pred_mloa_lda,y_test_mloa,zero_division=0))
print("Balanced Accuracy:", bac_mloa_lda)
print("Matthews Correlation Coefficient:", mcc_mloa_lda)
```

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(X_train_scaled_mloa, y_train_mloa)

# Make predictions on the testing data
y_pred_mloa_rfc = model.predict(X_test_scaled_mloa)
bac_mloa_rfc = balanced_accuracy_score(y_test_mloa, y_pred_mloa_rfc)

# Calculate MCC
mcc_mloa_rfc = matthews_corrcoef(y_test_mloa, y_pred_mloa_rfc)

# Evaluate the model performance
accuracy = accuracy_score(y_test_mloa, y_pred_mloa_rfc)
print("Accuracy:", accuracy)
print(classification_report(y_pred_mloa_rfc,y_test_mloa,zero_division=0))
print("Balanced Accuracy:", bac_mloa_rfc)
print("Matthews Correlation Coefficient:", mcc_mloa_rfc)
```

```python
from sklearn.ensemble import HistGradientBoostingClassifier
model = HistGradientBoostingClassifier()

# Train the model on the training data
model.fit(X_train_scaled_mloa, y_train_mloa)

# Make predictions on the testing data
y_pred_mloa_hgbc = model.predict(X_test_scaled_mloa)
bac_mloa_hgbc = balanced_accuracy_score(y_test_mloa, y_pred_mloa_hgbc)

# Calculate MCC
mcc_mloa_hgbc = matthews_corrcoef(y_test_mloa, y_pred_mloa_hgbc)

# Evaluate the model performance
accuracy = accuracy_score(y_test_mloa, y_pred_mloa_hgbc)
print("Accuracy:", accuracy)
print(classification_report(y_pred_mloa_hgbc,y_test_mloa,zero_division=0))
print("Balanced Accuracy:", bac_mloa_hgbc)
print("Matthews Correlation Coefficient:", mcc_mloa_hgbc)
```

3.7 Comparing All Models using Graphs

```
bars1=[bac_mloa_knn,bac_loa_knn,bac_knn]
bars2=[mcc_mloa_knn,mcc_loa_knn,mcc_knn]
barwidth=0.25
r1=np.arange(len(bars1))
r2=[x+barwidth for  x in r1]
plt.bar(r1,bars1,width=barwidth,label="BAC")
plt.bar(r2,bars2,width=barwidth,label='MCC')
plt.xticks([r+barwidth for r in range(len(bars1))],['MLOA_KNN','LOA_KNN','KNN'])
plt.ylabel('Percentage')
plt.legend()
```

```
bars1=[bac_mloa_lda,bac_loa_lda,bac_lda]
bars2=[mcc_mloa_lda,mcc_loa_lda,mcc_lda]
barwidth=0.25
r1=np.arange(len(bars1))
r2=[x+barwidth for  x in r1]
plt.bar(r1,bars1,width=barwidth,label="BAC")
plt.bar(r2,bars2,width=barwidth,label='MCC')
plt.xticks([r+barwidth for r in range(len(bars1))],['MLOA_LDA','LOA_LDA','LDA'])
plt.ylabel('Percentage')
plt.legend()
```

```
bars1=[bac_mloa_rfc,bac_loa_rfc,bac_rfc]
bars2=[mcc_mloa_rfc,mcc_loa_rfc,mcc_rfc]
barwidth=0.25
r1=np.arange(len(bars1))
r2=[x+barwidth for  x in r1]
plt.bar(r1,bars1,width=barwidth,label="BAC")
plt.bar(r2,bars2,width=barwidth,label='MCC')
plt.xticks([r+barwidth for r in range(len(bars1))],['MLOA_RFC','LOA_RFC','RFC'])
plt.ylabel('Percentage')
plt.legend()
```

```
bars1=[bac_mloa_hgbc,bac_loa_hgbc,bac_hgbc]
bars2=[mcc_mloa_hgbc,mcc_loa_hgbc,mcc_hgbc]
barwidth=0.25
r1=np.arange(len(bars1))
r2=[x+barwidth for  x in r1]
plt.bar(r1,bars1,width=barwidth,label="BAC")
plt.bar(r2,bars2,width=barwidth,label='MCC')
plt.xticks([r+barwidth for r in range(len(bars1))],['MLOA_HGBC','LOA_HGBC','HGBC'])
plt.ylabel('Percentage')
plt.legend()
```

```
per_inc_knn=(bac_mloa_knn-bac_knn)
per_inc_lda=(bac_mloa_lda-bac_lda)
per_inc_rfc=(bac_mloa_rfc-bac_rfc)
per_inc_hgbc=(bac_mloa_hgbc-bac_hgbc)
per_inc=[per_inc_knn,per_inc_lda,per_inc_rfc,per_inc_hgbc]
xx=np.arange(len(per_inc))
```

```
plt.bar(xx,per_inc)
plt.xticks([r+barwidth for r in range(len(per_inc))],['MLOA_KNN','MLOA_LDA','MLOA_RFC','MLOA_HGBC'])
plt.title("percentage increase of bac before and after MLOA")
```

```
bars1=rec0
bars2=prec0
bars3=f10
barwidth=0.25
r1=np.arange(len(bars1))
r2=[x+barwidth for x in r1]
r3=[x+barwidth for x in r2]
plt.bar(r1,bars1,width=barwidth,label="knn Recall")
plt.bar(r2,bars2,width=barwidth,label='knn Precision')
#plt.bar(r3,bars3,width=barwidth,label='F1-Score')
plt.plot(r3,bars3,marker='o',label='knn F1-Score',color='r')
plt.xticks([r+barwidth for r in range(len(bars1))],['BLB','Hispa','Blast','Healthy','leaf folder','leaf spot'])
plt.legend()
plt.title('Metrics before any Optimization')
```

```
bars1=rec2
bars2=prec2
bars3=f12
barwidth=0.25
r1=np.arange(len(bars1))
r2=[x+barwidth for x in r1]
r3=[x+barwidth for x in r2]
plt.bar(r1,bars1,width=barwidth,label="loa-knn Recall")
plt.bar(r2,bars2,width=barwidth,label='loa-knn Precision')
#plt.bar(r3,bars3,width=barwidth,label='F1-Score')
plt.plot(r3,bars3,marker='o',label='loa-knn F1-Score',color='r')
plt.xticks([r+barwidth for r in range(len(bars1))],['BLB','Hispa','Blast','Healthy','leaf folder','leaf spot'])
plt.legend()
plt.title('Metrics After Optimization')
```

```
bars1=rec1
bars2=prec1
bars3=f11
barwidth=0.25
r1=np.arange(len(bars1))
r2=[x+barwidth for x in r1]
r3=[x+barwidth for x in r2]
plt.bar(r1,bars1,width=barwidth,label="mloa-knn Recall")
plt.bar(r2,bars2,width=barwidth,label='mloa-knn Precision')
#plt.bar(r3,bars3,width=barwidth,label='F1-Score')
plt.plot(r3,bars3,marker='o',label='mloa-knn F1-Score',color='r')
plt.xticks([r+barwidth for r in range(len(bars1))],['BLB','Hispa','Blast','Healthy','leaf folder','leaf spot'])
plt.legend()
plt.title('Metrics After Optimization')
```

```
plt.scatter(msd1,vsd1,label='BLB')
plt.scatter(msd5,vsd5,label='Blast')
plt.legend()
plt.xlabel('Mean')
plt.ylabel('Variance')
plt.title('Initial Data points of mean and variance of BLB and Blast')
```

```
plt.scatter(mloa_msd11,mloa_vsd11,label='BLB')
plt.scatter(mloa_msd55,mloa_vsd55,label='Blast')
plt.legend()
plt.xlabel('Mean')
plt.ylabel('Variance')
plt.title('Initial Data points of mean and variance of BLB and Blast after MLOA')
```

CHAPTER 4
OUTPUT SNAPSHOTS

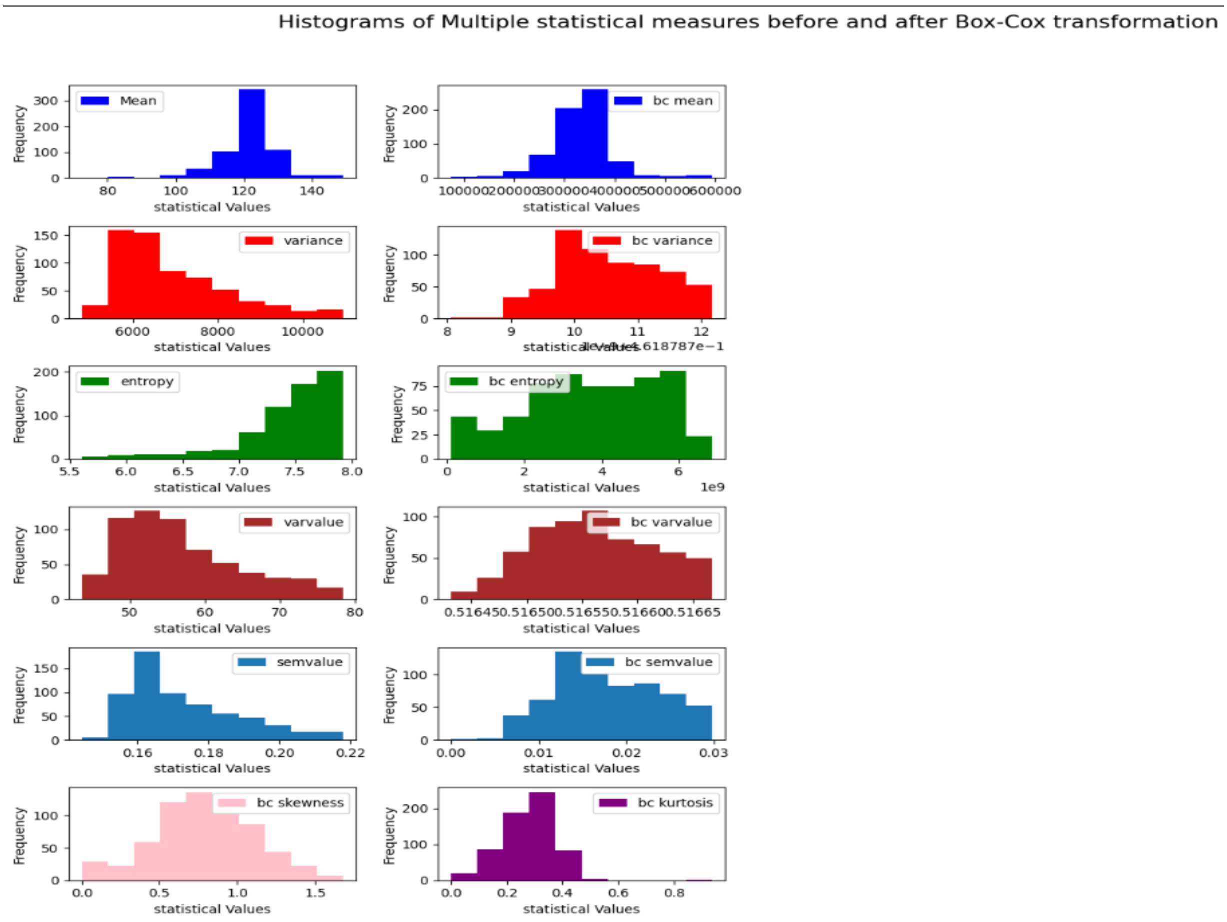

Fig 4.1 Histograms of Multiple Statistical Measures before & after Box-Cox Transformation

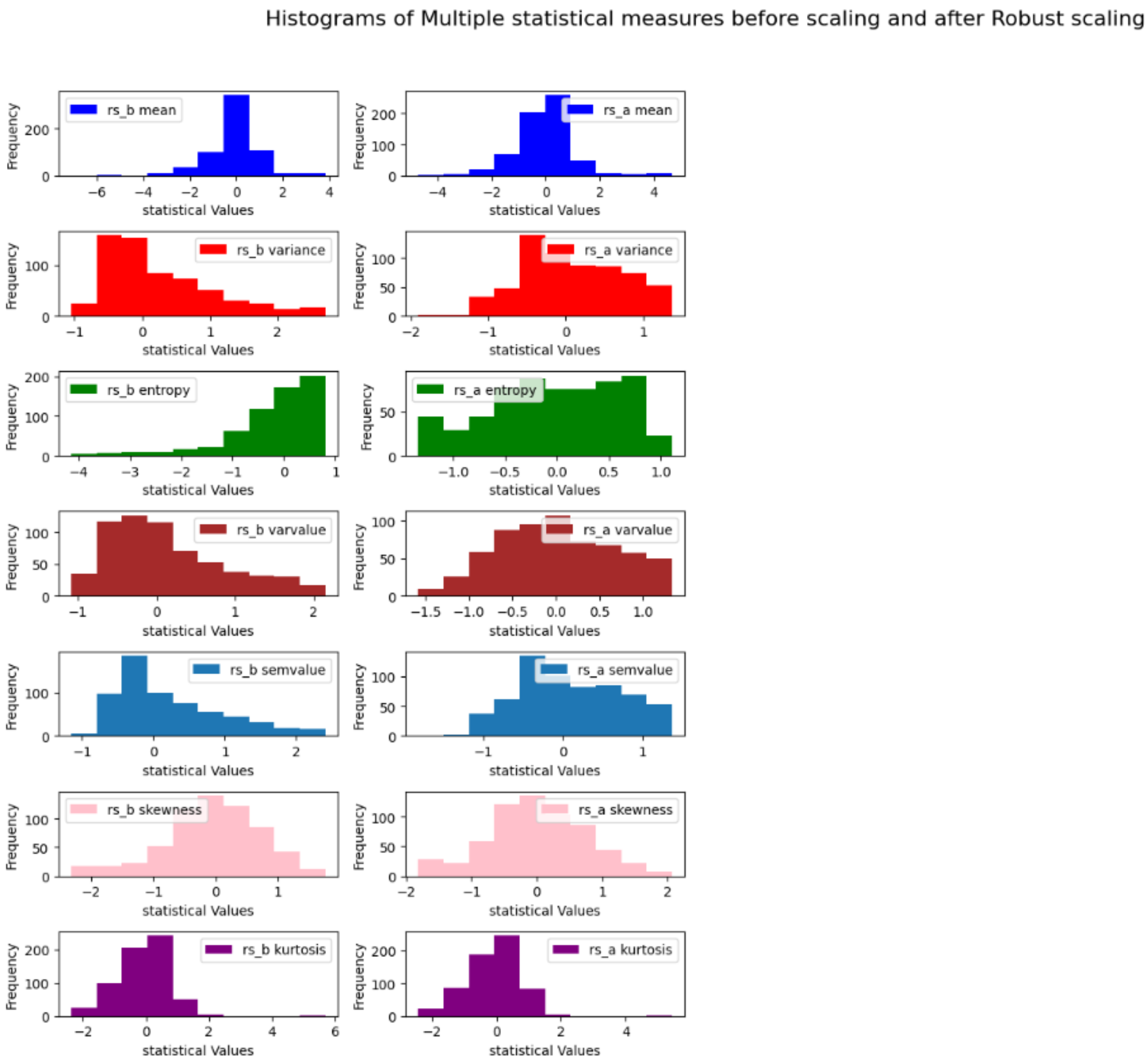

Fig 4.2 Histograms Of Multiple Statistical Features after Robust Scaling

```
accuracy :  0.7777777777777778
              precision    recall  f1-score   support

         BLB       1.00      0.68      0.81        31
       Blast       0.83      0.83      0.83         6
       HISPA       0.87      0.93      0.90        14
 Leaf Folder       1.00      1.00      1.00         3
   Leaf spot       0.38      0.75      0.50         4
     healthy       0.40      0.80      0.53         5

    accuracy                           0.78        63
   macro avg       0.75      0.83      0.76        63
weighted avg       0.87      0.78      0.80        63

Balanced Accuracy: 0.7458333333333335
Matthews Correlation Coefficient: 0.7227520932399434
```

Fig 4.3a KNN Model Classification Report

```
Accuracy: 0.6984126984126984
              precision    recall  f1-score   support

         BLB       0.90      0.63      0.75        30
       Blast       0.67      0.57      0.62         7
       HISPA       0.87      0.87      0.87        15
 Leaf Folder       0.67      1.00      0.80         2
   Leaf spot       0.38      0.60      0.46         5
     healthy       0.30      0.75      0.43         4

    accuracy                           0.70        63
   macro avg       0.63      0.74      0.65        63
weighted avg       0.78      0.70      0.72        63

Balanced Accuracy: 0.6299603174603174
Matthews Correlation Coefficient: 0.6133153565877141
```

Fig 4.3b LDA Model Classification Report

Accuracy: 0.8412698412698413

	precision	recall	f1-score	support
BLB	1.00	0.75	0.86	28
Blast	0.83	0.83	0.83	6
HISPA	0.87	0.93	0.90	14
Leaf Folder	1.00	1.00	1.00	3
Leaf spot	0.50	0.80	0.62	5
healthy	0.70	1.00	0.82	7
accuracy			0.84	63
macro avg	0.82	0.89	0.84	63
weighted avg	0.88	0.84	0.85	63

Balanced Accuracy: 0.8166666666666668
Matthews Correlation Coefficient: 0.8006990925016039

Fig 4.3c RFC Model Classification Report

Accuracy: 0.9047619047619048

	precision	recall	f1-score	support
BLB	1.00	0.91	0.95	23
Blast	0.83	1.00	0.91	5
HISPA	0.87	0.93	0.90	14
Leaf Folder	1.00	0.75	0.86	4
Leaf spot	0.62	1.00	0.77	5
healthy	1.00	0.83	0.91	12
accuracy			0.90	63
macro avg	0.89	0.90	0.88	63
weighted avg	0.93	0.90	0.91	63

Balanced Accuracy: 0.8875000000000001
Matthews Correlation Coefficient: 0.8799377404612962

Fig 4.3d HGBC Model Classification Report

0.9365079365079365

	precision	recall	f1-score	support
BLB	1.00	0.95	0.98	22
HISPA	1.00	0.88	0.94	17
blast	0.50	1.00	0.67	3
healthy	0.90	1.00	0.95	9
leaf folder	1.00	1.00	1.00	3
leaf spot	1.00	0.89	0.94	9
accuracy			0.94	63
macro avg	0.90	0.95	0.91	63
weighted avg	0.96	0.94	0.94	63

Balanced Accuracy: 0.9
Matthews Correlation Coefficient: 0.9202059187793642

Fig 4.4a KNN-LOA Model Classification Report

Accuracy: 0.8412698412698413

	precision	recall	f1-score	support
BLB	1.00	0.91	0.95	23
HISPA	0.80	0.80	0.80	15
blast	0.33	0.67	0.44	3
healthy	1.00	1.00	1.00	10
leaf folder	0.67	0.50	0.57	4
leaf spot	0.75	0.75	0.75	8
accuracy			0.84	63
macro avg	0.76	0.77	0.75	63
weighted avg	0.87	0.84	0.85	63

Balanced Accuracy: 0.7583333333333333
Matthews Correlation Coefficient: 0.7964543941271558

Fig 4.4b LDA-LOA Model ClassificationReport

Accuracy: 0.9206349206349206

	precision	recall	f1-score	support
BLB	1.00	0.88	0.93	24
HISPA	0.93	1.00	0.97	14
blast	0.50	0.60	0.55	5
healthy	1.00	1.00	1.00	10
leaf folder	1.00	1.00	1.00	3
leaf spot	0.88	1.00	0.93	7
accuracy			0.92	63
macro avg	0.88	0.91	0.90	63
weighted avg	0.93	0.92	0.92	63

Balanced Accuracy: 0.8847222222222223
Matthews Correlation Coefficient: 0.8988983228219788

Fig 4.4c RFC-LOA Model Classification Report

Accuracy: 0.7936507936507936

	precision	recall	f1-score	support
BLB	1.00	0.88	0.93	24
HISPA	0.80	0.92	0.86	13
blast	1.00	0.67	0.80	9
healthy	1.00	0.62	0.77	16
leaf folder	0.33	1.00	0.50	1
leaf spot	0.00	0.00	0.00	0
accuracy			0.79	63
macro avg	0.69	0.68	0.64	63
weighted avg	0.95	0.79	0.85	63

Balanced Accuracy: 0.6888888888888888
Matthews Correlation Coefficient: 0.7476095676938572

Fig 4.4d HGBC-LOA Model Classification Report

Accuracy: 0.9841269841269841

	precision	recall	f1-score	support
BLB	1.00	0.95	0.98	22
HISPA	1.00	1.00	1.00	15
blast	1.00	1.00	1.00	6
healthy	1.00	1.00	1.00	10
leaf folder	0.67	1.00	0.80	2
leaf spot	1.00	1.00	1.00	8
accuracy			0.98	63
macro avg	0.94	0.99	0.96	63
weighted avg	0.99	0.98	0.99	63

Balanced Accuracy: 0.9444444444444445
Matthews Correlation Coefficient: 0.9798561031888163

Fig 4.5a KNN-MLOA Model Classification Report

Accuracy: 0.9682539682539683

	precision	recall	f1-score	support
BLB	1.00	1.00	1.00	21
HISPA	1.00	0.94	0.97	16
blast	1.00	1.00	1.00	6
healthy	1.00	1.00	1.00	10
leaf folder	0.67	1.00	0.80	2
leaf spot	0.88	0.88	0.88	8
accuracy			0.97	63
macro avg	0.92	0.97	0.94	63
weighted avg	0.97	0.97	0.97	63

Balanced Accuracy: 0.9236111111111112
Matthews Correlation Coefficient: 0.959437297565166

Fig 4.5b LDA-MLOA Model Classification Report

```
accuracy :  0.9682539682539683
              precision    recall  f1-score   support

         BLB       1.00      1.00      1.00        21
       HISPA       0.93      1.00      0.97        14
       blast       1.00      1.00      1.00         6
     healthy       1.00      1.00      1.00        10
 leaf folder       0.67      1.00      0.80         2
   leaf spot       1.00      0.80      0.89        10

    accuracy                           0.97        63
   macro avg       0.93      0.97      0.94        63
weighted avg       0.97      0.97      0.97        63

Balanced Accuracy: 0.9333333333333335
Matthews Correlation Coefficient: 0.9602328338916196
```

Fig 4.5c RFC-MLOA Model Classification Report

```
Accuracy: 0.9682539682539683
              precision    recall  f1-score   support

         BLB       1.00      0.95      0.98        22
       HISPA       1.00      0.94      0.97        16
       blast       1.00      1.00      1.00         6
     healthy       1.00      1.00      1.00        10
 leaf folder       0.67      1.00      0.80         2
   leaf spot       0.88      1.00      0.93         7

    accuracy                           0.97        63
   macro avg       0.92      0.98      0.95        63
weighted avg       0.98      0.97      0.97        63

Balanced Accuracy: 0.9236111111111112
Matthews Correlation Coefficient: 0.9596067973347746
```

Fig 4.5d HGBC-MLOA Model Classification Report

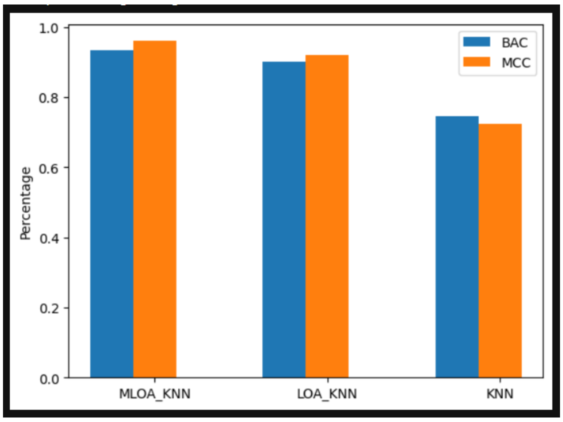

Fig 4.6a BAC-MCC Percentage of KNN-MLOA

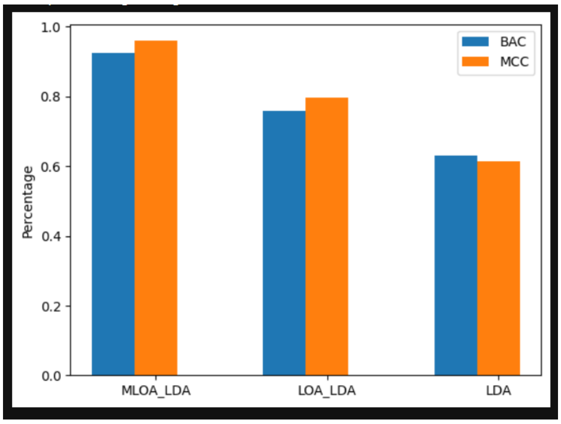

Fig 4.6b BAC-MCC Percentage of LDA-MLOA

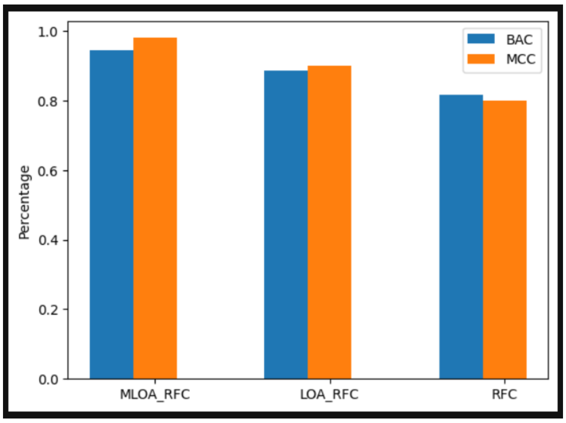

Fig 4.6a BAC-MCC Percentage of RFC-MLOA

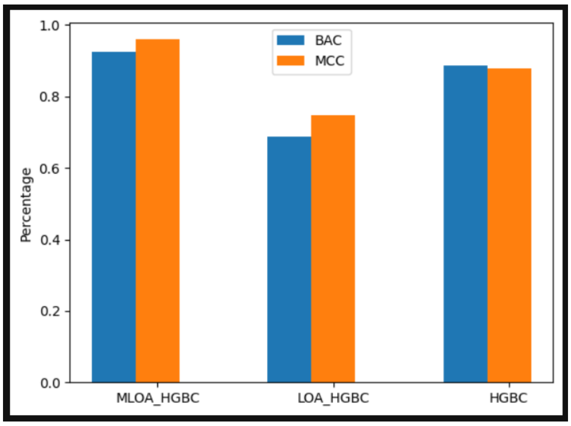

Fig 4.6a BAC-MCC Percentage of HGBC-MLOA

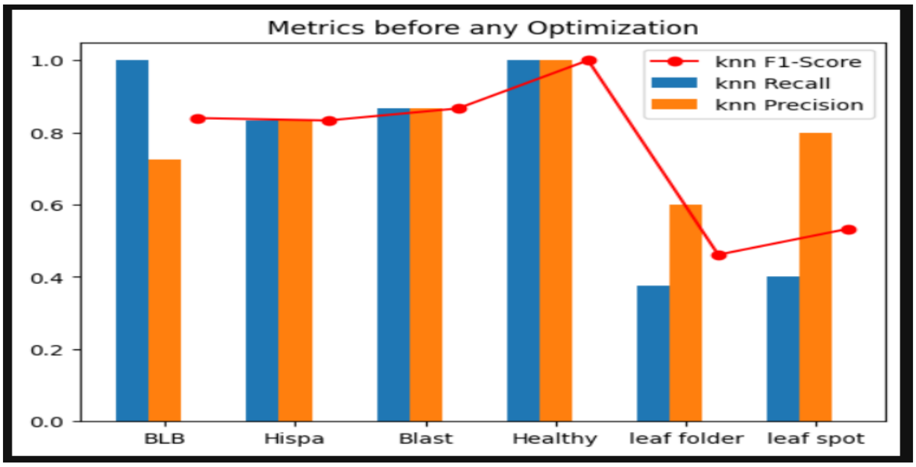

Fig 4.7a Metrics before any Optimization

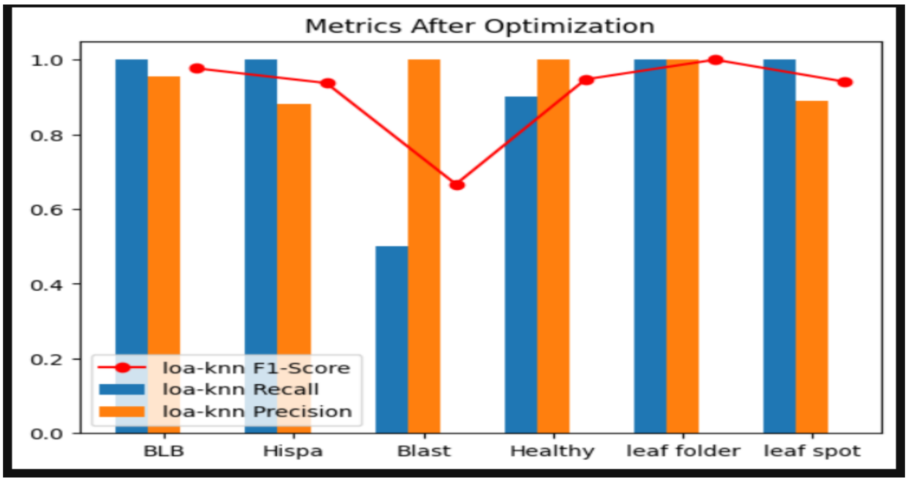

Fig 4.7b Metrics after LOA Optimization

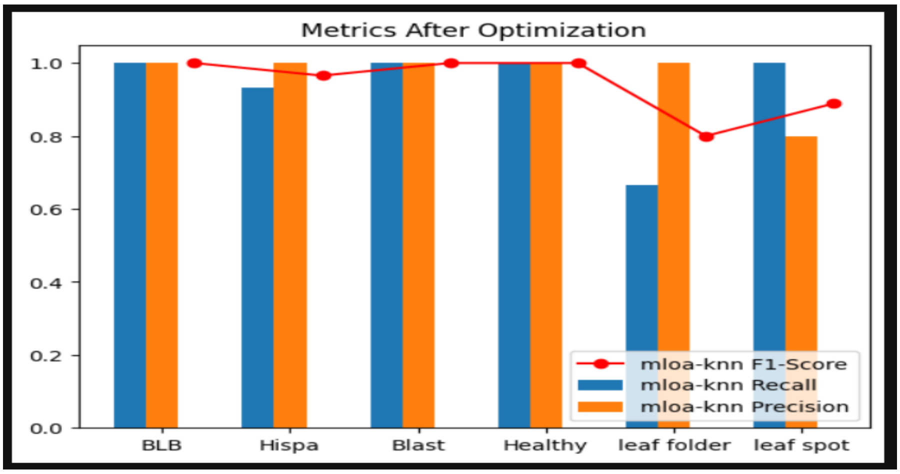

Fig 4.7c Metrics after MLOA Optimization

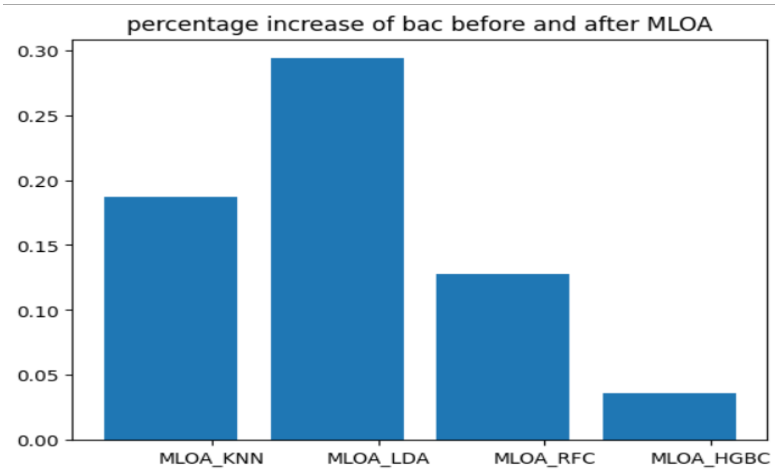

Fig 4.8 Percentage increase of BAC after MLOA

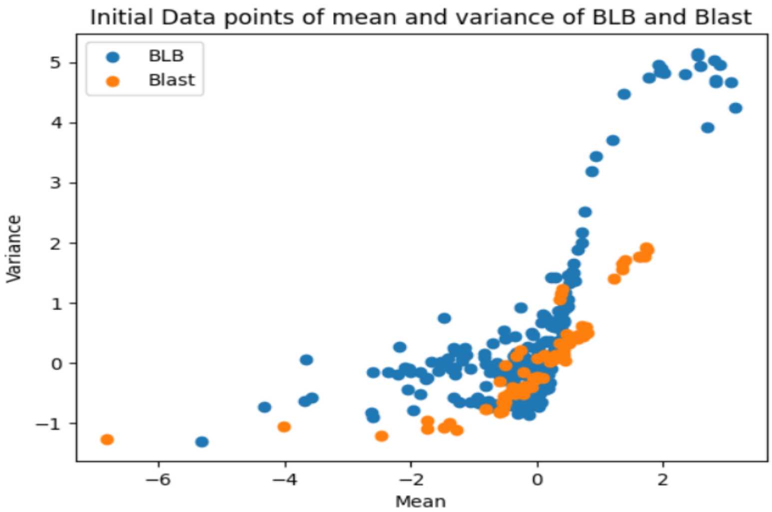

Fig 4.9a Intial Data Points of mean & variance of BLB & Blast

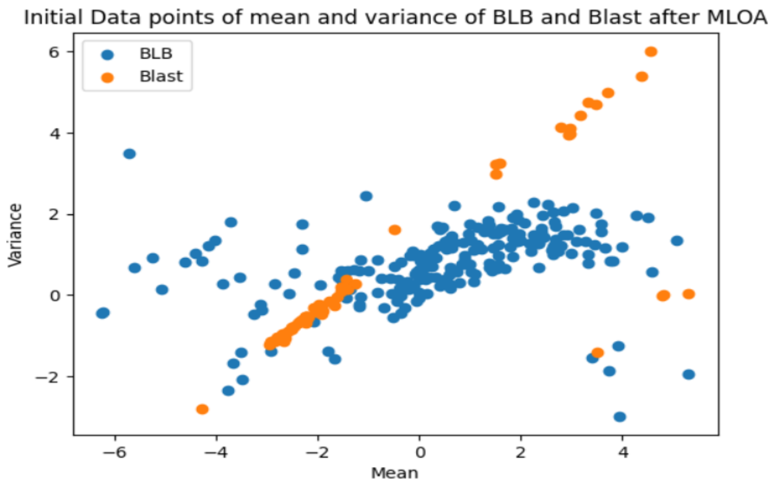

Fig 4.9b Intial Data Points of mean & variance of BLB & Blast after MLO

CHAPTER 5
COMPARISON OF RESULTS

	BAC(%)	MCC(%)	Weighted Average F1-score(%)	Weighted Average Recall(%)	Weighted Average Precision(%)
KNN	**75**	**72**	**80**	**78**	**87**
LDA	**63**	**61**	**72**	**70**	**78**
RFC	**82**	**80**	**85**	**84**	**88**
HGBC	**89**	**88**	**91**	**90**	**93**
KNN-LOA	**90**	**92**	**94**	**94**	**96**
LDA-LOA	**76**	**80**	**85**	**84**	**87**
RFC-LOA	**88**	**90**	**92**	**92**	**93**
HGBC-LOA	**69**	**75**	**85**	**79**	**95**
KNN-MLOA	**94**	**98**	**99**	**98**	**99**
LDA-MLOA	**92**	**96**	**97**	**97**	**97**
RFC-MLOA	**93**	**96**	**97**	**97**	**97**
HGBC-MLOA	**92**	**96**	**97**	**97**	**98**

Table 5.1 Accuracies of all models with and without optimization

CHAPTER 6
CONCLUSION AND FUTURE PLANS

The preceding section demonstrates the efficacy of MLOA as a feature transformation method for enhancing the classification accuracy of various machine learning techniques in detecting rice leaf diseases. Notably, the MLOA approach outperforms LOA based transformations. Effective initialization of population and parameters, selection of suitable fitness functions, and dynamic adjustment of the weight parameter FRR using filter-based techniques based on SGD contribute to the enhanced classification performance compared to alternative methods.

Particularly noteworthy is the achievement of 90% BAC with the MLOA-KNN classifier, a significant improvement over the 64% BAC offered by KNN alone. Other tested classifiers also exhibit substantial performance improvements when coupled with the MLOA feature transformation in rice leaf disease detection. Future research should explore the applicability of MLOA feature transformation in other domains and with different types of features. Furthermore, the proposed transformation should be evaluated with a broader range of classifiers to achieve a BAC of 95% or higher in rice leaf disease detection**.**

CHAPTER 7

REFERENCES

1. **LOA Algorithm**:
    a. https://www.sciencedirect.com/science/article/pii/S1319157823002586
    b. https://www.mdpi.com/2076-3417/12/19/10057

2. **Robust Scaler:**
    a. https://pubmed.ncbi.nlm.nih.gov/27612635/

3. H**istogram Gradient Boosting Classifier:**
    a. https://journals.sagepub.com/doi/10.1177/03611981221074370

4. **Supervised Models:**
    a. https://www.semanticscholar.org/paper/Supervised-Machine-Learning-Algorithms%3A-and-F.Y-J.E.T/bed9dc37c6597136eb5ae761a14b2d7f8e0204a1
    b. https://pubmed.ncbi.nlm.nih.gov/27386492/

CHAPTER 8

APPENDIX

BASE PAPER

N. Bharanidharan, S. R. S. Chakravarthy, H. Rajaguru, V. V. Kumar, T. R. Mahesh and S. Guluwadi, "Multiclass Paddy Disease Detection Using Filter-Based Feature Transformation Technique," in IEEE Access, vol. 11, pp. 109477-109487, 2023

DOI : 10.1109/ACCESS.2023.3322587.

Keywords: {Diseases, Optimization, Transforms, Feature extraction, Classification algorithms, Training, Machine learning algorithms, Particle swarm optimization, Thermal analysis, Paddy disease, machine learning, swarm optimization, modified lemurs optimization, thermal image processing}

URL: https://ieeexplore.ieee.org/document/10273722