**Plant disease identification from the drone captured images**

**Synopsis:**

Plant diseases pose a significant threat to agricultural productivity and food security worldwide. Timely detection and accurate diagnosis of these diseases are essential for effective disease management and crop protection. This project presents an innovative approach to plant disease identification utilizing drone-captured images and a variety of machine learning (ML) algorithms. Through extensive experimentation and validation, our project aims to demonstrate the efficacy and practical utility of using drone-captured images for automated plant disease identification. By harnessing the capabilities of ML algorithms, we strive to provide farmers and agricultural stakeholders with a cost-effective and efficient tool for early disease detection.

**SYSTEM ENVIRONMENT**

2.1 Hardware Requirements:

Processor : Intel Core i4 (10th Gen)

Ram : 4.0 GB

2.2 Software Requirements

Operating System : Windows 10

Framework : Google colab

Language : python

**2.3 About the technology:**

Python:

Python is an interpreted high-level general-purpose programming language created by Guido Van Rossum and first published in 1991. Python's design philosophy emphasizes code readability with significant whitespace. Its language structures and object-oriented approach are designed to help developers write clear and logical code for small and large projects. Python is dynamically typed and garbage

Google Colab:

Google Colab, short for Google Colaboratory, is a cloud-based, interactive computing platform provided by Google. It allows users to write and execute Python code in a collaborative and convenient environment directly through a web browser. Colab provides free access to GPU and TPU (Tensor Processing Unit) resources, enabling accelerated execution of machine learning tasks. Users can create and share Jupyter notebooks, incorporating text, code, and visualizations seamlessly. Colab integrates with Google Drive, facilitating easy storage and sharing of notebooks. Its collaborative features enable multiple users to work on the same document simultaneously, fostering collaborative research and development. Overall, Google Colab is a powerful and accessible tool for data analysis, machine learning, and collaborative coding, making it particularly valuable for researchers, students, and practitioners in the field of data science.

Scikit Learn:

Scikit-learn (Sklearn) is the most useful and powerful Python machine learning library. It provides a number of powerful tools for machine learning and statistical modelling, including classification, regression, clustering and dimensionality reduction through a Python consistent interface. Written mostly in Python, this library is built on top of NumPy, SciPy and Matplotlib. Originally called scikits. learn, it was originally developed by David Cournapeau as a Google Summer Code Project in 2007. Later, in 2010, Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, and Vincent Michel from FIRCA (French Institute for Informatics and Automation) adopted it this project to a new level and released the first public release (v0.1 beta) on February 1, 2010

**EXISTING SYSTEM**

There are several existing systems for plant disease identification from drone-captured images using machine learning techniques.

PlantVillage is a platform that utilizes machine learning for plant disease identification. It offers a mobile application where users can take pictures of plant leaves and submit them for analysis. The system then employs machine learning algorithms to identify potential diseases or pests affecting the plant.

Plantix is another mobile application that employs machine learning for plant disease identification. Users can capture images of diseased plants using their smartphones and upload them to the app. The system uses deep learning algorithms to analyze the images and provide users with information about the disease, including possible treatment options.

AgroView is a drone-based system for monitoring crop health and detecting diseases. It utilizes machine learning algorithms to analyze images captured by drones flying over agricultural fields. The system can identify patterns associated with diseased plants and alert farmers to potential problems, enabling timely intervention.

DeepAgro is a system that combines drone imagery with machine learning for plant disease detection. It uses high-resolution images of crops captured by drones to analyze. The system can identify specific disease symptoms, such as leaf discoloration or lesions, and provide farmers with actionable insights to mitigate crop damage.

Agremo is a platform that offers drone-based agricultural analytics, including disease detection. It utilizes machine learning algorithms to analyze aerial images of crops and identify signs of disease or stress. The system provides farmers with detailed reports on the health status of their fields, helping them make informed decisions about pest management and crop protection.

These systems typically involve a pipeline where drone-captured images are preprocessed to extract relevant features, which are then fed into machine learning models trained on labeled data to classify the presence of diseases or pests. The models are continuously improved through feedback loops to enhance their accuracy and effectiveness in real-world agricultural settings.

**PROPOSED SYSTEM**

Here's a proposed system for plant disease identification from drone-captured images using the GLCM (Gray-Level Co-occurrence Matrix) method for feature extraction, followed by preprocessing using Min-Max scaling, and then applying machine learning algorithms such as Decision Tree and Gradient Boosting. Finally, we evaluate the models using confusion matrix and classification report:

Drone-captured images are preprocessed to extract texture features using the GLCM method. GLCM computes the frequency of co-occurring pixel values at a specified spatial relationship within an image, capturing texture information. Features such as contrast, correlation, energy, and homogeneity can be extracted from GLCM.

After feature extraction the feature data is exported to a CSV file for storage and further analysis. The extracted features are preprocessed using Min-Max scaling. Min-Max scaling scales the features to a fixed range, usually between 0 and 1, preserving the relative distances between data points.

A decision tree classifier is trained using the preprocessed feature data. Decision trees are simple yet powerful models that learn decision rules from the data. A gradient boosting classifier is trained using the preprocessed feature data. Gradient boosting combines the predictions of multiple weak learners (typically decision trees) to build a strong predictive model.

The confusion matrix is used to evaluate the performance of the trained classifiers. It shows the true positive, true negative, false positive, and false negative counts. The classification report provides precision, recall, F1-score, and support for each class in the dataset. It gives a more detailed summary of the classifier's performance.

**Advantages of the proposed system:**

The proposed system for plant disease identification from drone-captured images using the GLCM method for feature extraction, Min-Max scaling for preprocessing, and machine learning algorithms like Decision Tree and Gradient Boosting offers several advantages:

**Accurate Disease Identification:**

By utilizing texture features extracted using GLCM, the system can capture subtle patterns and textures indicative of various plant diseases. This allows for more accurate identification compared to methods solely based on color or shape features.

**Robust Feature Scaling:**

The Min-Max scaling method used for preprocessing ensures that all feature values are scaled to a fixed range, preventing the dominance of certain features due to their magnitude. This improves the robustness of the machine learning models and prevents bias towards features with larger scales.

**Interpretability of Decision Tree:**

Decision Tree classifiers offer interpretability, as they generate decision rules that can be easily understood by domain experts. This transparency allows farmers or agronomists to gain insights into the factors influencing disease identification, aiding in decision-making processes.

**High Predictive Performance of Gradient Boosting:**

Gradient Boosting algorithms, such as Gradient Boosted Trees, often provide high predictive performance by combining the strengths of multiple weak learners. This results in more accurate and robust disease identification, especially when dealing with complex datasets or subtle disease symptoms.

**Comprehensive Evaluation Metrics:**

The use of confusion matrices and classification reports enables a comprehensive evaluation of the system's performance. These metrics provide insights into the model's ability to correctly identify diseased and healthy plants, as well as its precision, recall, and F1-score for each class, aiding in performance assessment and model refinement.

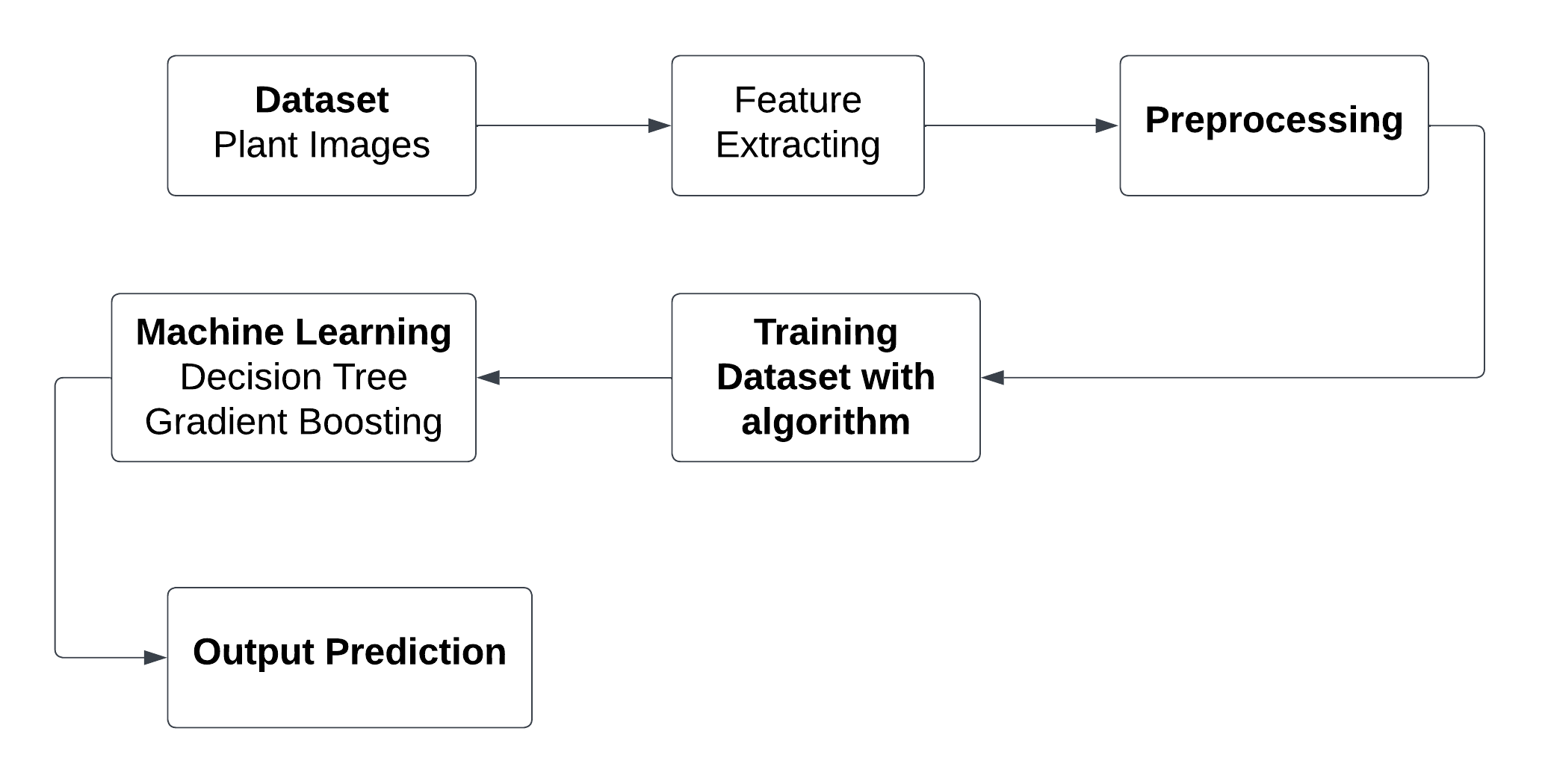
**Automated Processing:**

The system can be automated to process large volumes of drone-captured images efficiently. This scalability allows for timely and cost-effective monitoring of crop health across extensive agricultural areas, facilitating early detection and intervention to prevent widespread crop damage.

The proposed system offers a robust, scalable, and interpretable approach to plant disease identification from drone-captured images, providing valuable insights for farmers, agronomists, and researchers to optimize crop health and productivity.

**SYSTEM DESIGN:**

Plant disease identification from the drone captured images is designed by the below systematic diagram:



**Dataset Description:**

The Plant Village dataset is a widely used collection of images for research and development in the field of plant disease detection and classification. Here's a description of the PlantVillage dataset:

The dataset consists of a large number of images depicting various plant diseases, along with images of healthy plants for comparison. It contains tens of thousands of images covering multiple plant species and a variety of diseases and pest damage. The dataset includes images of different plant species commonly cultivated in agriculture, such as tomatoes, potatoes, apples, grapes, wheat, maize, etc. Each plant species may be affected by various diseases and pests. The dataset covers a wide range of plant diseases caused by pathogens like fungi, bacteria, viruses, as well as damage caused by pests and environmental stressors. Common diseases include leaf spot, powdery mildew, blight, rust, rot, and many others. The images exhibit variability in terms of lighting conditions, background clutter, plant growth stages, and the severity of disease symptoms. This variability reflects real-world conditions encountered in agricultural settings, making the dataset suitable for robust model training and evaluation. Many images in the Plant Village dataset are annotated with labels indicating the presence of specific diseases or conditions. These annotations serve as ground truth labels for training and evaluating machine learning models. The Plant Village dataset is publicly available for non-commercial research purposes and can be accessed through the official Plant Village website or other research repositories. Its diverse collection of images facilitates the development of robust and accurate machine learning models for automated plant disease diagnosis and monitoring.

**Feature Extraction:**

Feature extraction using the Grey-Level Co-occurrence Matrix (GLCM) is a widely utilized technique in image processing and pattern recognition. GLCM quantifies the spatial relationships between pixel intensity values in an image by computing the frequency of occurrence of pairs of intensity values at various spatial displacements. By analysing these relationships, GLCM generates a set of statistical measures, such as contrast, correlation, energy, and homogeneity, which serve as texture descriptors capturing different aspects of image texture and structure. These descriptors effectively summarize the texture information within an image, providing a compact and informative representation for subsequent analysis tasks, such as classification, segmentation, and object detection. Overall, feature extraction using GLCM enables the extraction of discriminative texture features from images, facilitating the characterization and understanding of complex visual patterns in diverse application

**Pre-Processing:**

After applying the Min-Max scaler for preprocessing, the data is scaled to a fixed range, typically between 0 and 1, preserving the relative distances between data points. This ensures that all features contribute equally to the analysis. The preprocessed data is then fed into machine learning algorithms such as Decision Trees and Gradient Boosting for training. These algorithms learn patterns from the data and create models capable of accurately classifying plant diseases from drone-captured images. Finally, the trained models are evaluated using techniques like confusion matrices and classification reports to assess their performance and effectiveness in disease identification.

**Machine learning algorithm**

**1.Decision Tree**

Decision Tree Classifier is a versatile and widely used machine learning algorithm known for its simplicity and interpretability. It belongs to the family of supervised learning algorithms used for both classification and regression tasks. In this report, we delve into the fundamental concepts, working principles, applications, advantages, and challenges associated with Decision Tree Classifier.

**Working Principles:**

At its core, a Decision Tree is a flowchart-like structure where each node represents a feature or attribute, each branch represents a decision rule, and each leaf node represents an outcome or a class label. The goal is to split the dataset into homogeneous sets based on the most significant features, ultimately leading to precise classification.

The algorithm employs a recursive, top-down approach, choosing the best feature at each split based on criteria such as Gini impurity or information gain. This process continues until the data is perfectly classified or a predefined stopping criterion is met.:

**Applications:**

Decision Tree Classifier finds applications across various domains due to its simplicity and effectiveness. Some notable applications include:

Finance: Predicting creditworthiness and fraud detection.

Medicine: Identifying diseases based on patient data.

Marketing: Customer segmentation and targeted advertising.

Manufacturing: Quality control and fault detection.

Agriculture: Crop disease prediction and yield estimation.

**Advantages:**

Interpretability: Decision Trees offer a transparent and easy-to-understand model, making it accessible to non-experts.

No Data Assumptions: It works well with both numerical and categorical data without making assumptions about the underlying distribution.

Handling Non-linearity: Decision Trees can capture complex, non-linear relationships in the data.

Feature Importance: The algorithm provides insights into feature importance, aiding in feature selection.

**Challenges:**

Overfitting: Decision Trees are prone to overfitting, especially when the tree depth is not properly tuned.

Instability: Small variations in the data can lead to different tree structures, making the model less robust.

Bias Towards Dominant Classes: In imbalanced datasets, Decision Trees may favor the majority class.

Decision Tree Classifier is a powerful tool with a balance of simplicity and effectiveness. Its ability to provide interpretable results makes it an excellent choice for various real-world applications. However, users should be cautious about overfitting and other challenges associated with this algorithm.

**2. Gradient Boosting:**

Boosting is one of the popular learning ensemble modeling techniques used to build strong classifiers from various weak classifiers. It starts with building a primary model from available training data sets then it identifies the errors present in the base model. After identifying the error, a secondary model is built, and further, a third model is introduced in this process. In this way, this process of introducing more models is continued until we get a complete training data set by which model predicts correctly. AdaBoost (Adaptive boosting) was the first boosting algorithm to combine various weak classifiers into a single strong classifier in the history of machine learning. It primarily focuses to solve classification tasks such as binary classification.

GBM in Machine Learning:

Further, instead of using these models separately to predict the outcome if we use them in form of series or combination, then we get a resulting model with correct information than all base models. In other words, instead of using each model's individual prediction, if we use average prediction from these models then we would be able to capture more information from the data. It is referred to as ensemble learning and boosting is also based on ensemble methods in machine learning.

Gradient Boosting Machine (GBM) is one of the most popular forward learning ensemble methods in machine learning. It is a powerful technique for building predictive models for regression and classification tasks. GBM helps us to get a predictive model in form of an ensemble of weak prediction models such as decision trees. Whenever a decision tree performs as a weak learner then the resulting algorithm is called gradient-boosted trees.

It enables us to combine the predictions from various learner models and build a final predictive model having the correct prediction.

But here one question may arise if we are applying the same algorithm then how multiple decision trees can give better predictions than a single decision tree? Moreover, how does each decision tree capture different information from the same data?

How do GBM works?

Generally, most supervised learning algorithms are based on a single predictive model such as linear regression, penalized regression model, decision trees, etc. But there are some supervised algorithms in ML that depend on a combination of various models together through the ensemble. In other words, when multiple base models contribute their predictions, an average of all predictions is adapted by boosting algorithms.

Gradient boosting machines consist 3 elements as follows:

Loss function

Weak learners

Additive model

1. Loss function:

Although, there is a big family of Loss functions in machine learning that can be used depending on the type of tasks being solved. The use of the loss function is estimated by the demand of specific characteristics of the conditional distribution such as robustness. While using a loss function in our task, we must specify the loss function and the function to calculate the corresponding negative gradient. Once, we get these two functions, they can be implemented into gradient boosting machines easily. However, there are several loss functions have been already proposed for GBM algorithms.

2. Weak Learner:

Weak learners are the base learner models that learn from past errors and help in building a strong predictive model design for boosting algorithms in machine learning. Generally, decision trees work as a weak learners in boosting algorithms.

Boosting is defined as the framework that continuously works to improve the output from base models. Many gradient boosting applications allow you to "plugin" various classes of weak learners at your disposal. Hence, decision trees are most often used for weak (base) learners.

How to train weak learners:

Machine learning uses training datasets to train base learners and based on the prediction from the previous learner, it improves the performance by focusing on the rows of the training data where the previous tree had the largest errors or residuals. E.g. shallow trees are considered weak learner to decision trees as it contains a few splits. Generally, in boosting algorithms, trees having up to 6 splits are most common.

Below is a sequence of training the weak learner to improve their performance where each tree is in the sequence with the previous tree's residuals. Further, we are introducing each new tree so that it can learn from the previous tree's errors. These are as follows:

Consider a data set and fit a decision tree into it.

F1(x)=y

Fit the next decision tree with the largest errors of the previous tree.

h1(x)=y?F1(x)

Add this new tree to the algorithm by adding both in steps 1 and 2.

F2(x)=F1(x)+h1(x)

Again fit the next decision tree with the residuals of the previous tree.

h2(x)=y?F2(x)

Repeat the same which we have done in step 3.

F3(x)=F2(x)+h2(x)

Continue this process until some mechanism (i.e. cross-validation) tells us to stop. The final model here is a stagewise additive model of b individual trees:

f(x)=B∑b=1fb(x)

Hence, trees are constructed greedily, choosing the best split points based on purity scores like Gini or minimizing the loss.

3. Additive Model:

The additive model is defined as adding trees to the model. Although we should not add multiple trees at a time, only a single tree must be added so that existing trees in the model are not changed. Further, we can also prefer the gradient descent method by adding trees to reduce the loss.

In the past few years, the gradient descent method was used to minimize the set of parameters such as the coefficient of the regression equation and weight in a neural network. After calculating error or loss, the weight parameter is used to minimize the error. But recently, most ML experts prefer weak learner sub-models or decision trees as a substitute for these parameters. In which, we have to add a tree in the model to reduce the error and improve the performance of that model. In this way, the prediction from the newly added tree is combined with the prediction from the existing series of trees to get a final prediction. This process continues until the loss reaches an acceptable level or is no longer improvement required. This method is also known as functional gradient descent or gradient descent with functions.

**Advantages of Boosting Algorithms:**

* Boosting algorithms follow ensemble learning which enables a model to give a more accurate prediction that cannot be trumped.
* Boosting algorithms are much more flexible than other algorithms as can optimize different loss functions and provides several hyperparameter tuning options.
* It does not require data pre-processing because it is suitable for both numeric as well as categorical variables.
* It does not require imputation of missing values in the dataset, it handles missing data automatically.

**Disadvantages of Boosting Algorithms:**

* Boosting algorithms may cause overfitting as well as overemphasizing the outliers.
* Gradient boosting algorithm continuously focuses to minimize the errors and requires multiple trees hence, it is computationally expensive.
* It is a time-consuming and memory exhaustive algorithm.
* Less interpretative in nature, although this is easily addressed with various tools.

In this way, we have learned boosting algorithms for predictive modeling in machine learning. Also, we have discussed various important boosting algorithms used in ML such as GBM, XGBM, light GBM, and Catboost. Further, we have seen various components (loss function, weak learner, and additive model) and how GBM works with them. How boosting algorithms are advantageous for deployment in real-world scenarios, etc.

**Libraries used in the implementation:**

NumPy: NumPy is a fundamental library for numerical computing in Python, providing support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions. It serves as a foundational tool for scientific computing tasks, enabling efficient and high-performance operations on numerical data.

Pandas: Pandas is a versatile data manipulation library in Python that offers data structures like DataFrames and Series, facilitating efficient data analysis and manipulation. It provides functionalities for cleaning, transforming, and exploring datasets, making it a go-to tool for handling structured data in various stages of the data science workflow.

Matplotlib: Matplotlib is a powerful plotting library for Python that allows the creation of diverse static, animated, and interactive visualizations. With a comprehensive set of functions, Matplotlib provides users with the flexibility to create various charts, plots, and graphs, making it an essential tool for data visualization and communication of findings.

Seaborn: Seaborn is a statistical data visualization library built on top of Matplotlib. It provides a high-level interface for creating aesthetically pleasing and informative statistical graphics. Seaborn simplifies the process of generating complex visualizations, including heatmaps, pair plots, and violin plots, while maintaining customization options for advanced users.

Metrics (Accuracy, Classification, Confusion Matrix, ROC AUC): In the context of machine learning evaluation, metrics play a crucial role. Accuracy represents the proportion of correctly classified instances, serving as a fundamental measure of model performance. Classification metrics, such as precision, recall, and F1-score, provide insights into the model's ability to correctly identify instances of a particular class. The confusion matrix presents a comprehensive summary of true positive, true negative, false positive, and false negative predictions. Lastly, the ROC AUC (Receiver Operating Characteristic - Area Under the Curve) is a performance metric for binary classification models, illustrating the trade-off between sensitivity and specificity across different thresholds, providing a holistic view of the model's discriminatory power. These metrics collectively aid in assessing and optimizing the performance of machine learning models.

**CODING**

%matplotlib inline

import math

import numpy as np

import matplotlib.pyplot as plt

import os

import pandas as pd

from skimage import measure

from skimage.color import rgb2gray

from skimage.util import img\_as\_ubyte

from skimage import io

from skimage.feature import graycomatrix, graycoprops

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

imgList = os.listdir('/content/drive/MyDrive/Plant\_Village')

imgPath = []

for path in imgList:

pathImg = f'/content/drive/MyDrive/Plant\_Village/{path}'

imgPath.append(pathImg)

imgPath.sort()

ax = []

ay = []

az = []

aa = []

ab = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[0], levels=256)

ax.append(graycoprops(glcm, 'dissimilarity')[0, 0])

ay.append(graycoprops(glcm, 'correlation')[0, 0])

az.append(graycoprops(glcm, 'homogeneity')[0, 0])

aa.append(graycoprops(glcm, 'contrast')[0, 0])

ab.append(graycoprops(glcm, 'energy')[0, 0])

# 4.18879, 2.35619, 3.92699, 1.74533, 3.49066, 5.23599, 5.93412

bx = []

by = []

bz = []

ba = []

bb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[0.785398], levels=256)

bx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

by.append(graycoprops(glcm, 'correlation')[0, 0])

bz.append(graycoprops(glcm, 'homogeneity')[0, 0])

ba.append(graycoprops(glcm, 'contrast')[0, 0])

bb.append(graycoprops(glcm, 'energy')[0, 0])

cx = []

cy = []

cz = []

ca = []

cb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[1.0472], levels=256)

cx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

cy.append(graycoprops(glcm, 'correlation')[0, 0])

cz.append(graycoprops(glcm, 'homogeneity')[0, 0])

ca.append(graycoprops(glcm, 'contrast')[0, 0])

cb.append(graycoprops(glcm, 'energy')[0, 0])

dx = []

dy = []

dz = []

da = []

db = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[2.0944], levels=256)

dx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

dy.append(graycoprops(glcm, 'correlation')[0, 0])

dz.append(graycoprops(glcm, 'homogeneity')[0, 0])

da.append(graycoprops(glcm, 'contrast')[0, 0])

db.append(graycoprops(glcm, 'energy')[0, 0])

ex = []

ey = []

ez = []

ea = []

eb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[3.14159], levels=256)

ex.append(graycoprops(glcm, 'dissimilarity')[0, 0])

ey.append(graycoprops(glcm, 'correlation')[0, 0])

ez.append(graycoprops(glcm, 'homogeneity')[0, 0])

ea.append(graycoprops(glcm, 'contrast')[0, 0])

eb.append(graycoprops(glcm, 'energy')[0, 0])

fx = []

fy = []

fz = []

fa = []

fb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[4.18879], levels=256)

fx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

fy.append(graycoprops(glcm, 'correlation')[0, 0])

fz.append(graycoprops(glcm, 'homogeneity')[0, 0])

fa.append(graycoprops(glcm, 'contrast')[0, 0])

fb.append(graycoprops(glcm, 'energy')[0, 0])

imgName = os.listdir('/content/drive/MyDrive/Plant\_Village')

imgName.sort()

plant = pd.DataFrame(dict(label=0, image\_name=imgName, ax=ax, ay=ay, az=az, aa=aa, ab=ab, bx=bx, by=by, bz=bz, ba=ba, bb=bb, cx=cx, cy=cy, cz=cz, ca=ca, cb=cb, dx=dx, dy=dy, dz=dz, da=da, db=db, ex=ex, ey=ey, ez=ez, ea=ea, eb=eb, fx=fx, fy=fy,fz=fz, fa=fa, fb=fb ))

plant.rename(columns = {0: "image\_name", 1: "label"}, inplace = True)

plant = plant.sort\_values('image\_name')

plant.to\_csv('plant.csv', index=False)

plant.head()

df = pd.DataFrame(plant)

array=df.values

x\_feature=array[:,2:]

y\_label=array[:,2].astype('int')

print(x\_feature.shape)

print(y\_label.shape)

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(x\_feature,y\_label,test\_size=0.10,random\_state=7)

# Normalise the data after splitting to avoid information leak between train and test set.

scaler\_norm = MinMaxScaler()

X\_train = scaler\_norm.fit\_transform(X\_train)

X\_test = scaler\_norm.fit\_transform(X\_test)

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

train\_x, test\_x, train\_y, test\_y = train\_test\_split(x\_feature, y\_label, test\_size=0.30)

dtc = DecisionTreeClassifier(criterion='gini')

dtc.fit(train\_x, train\_y)

y\_pred = dtc.predict(test\_x)

accuracy = accuracy\_score(test\_y, y\_pred)

print("Accuracy Score of Decision Tree Classifier:", accuracy)

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(test\_y, y\_pred)

print('Confusion matrix\n\n', cm)

import seaborn as sns

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='cividis', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

# Generate a classification report

clr = print(classification\_report(test\_y, y\_pred, zero\_division=0))

import matplotlib.pyplot as plt

import seaborn as sns

class\_report = classification\_report(test\_y, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='RdPu')

plt.title('Classification Report Heatmap')

plt.show()

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

train\_x, test\_x, train\_y, test\_y = train\_test\_split(x\_feature, y\_label, test\_size=0.30)

gbm = GradientBoostingClassifier(n\_estimators=4000, learning\_rate=0.1, max\_depth=3, random\_state=42)

gbm.fit(train\_x, train\_y)

y\_pred = gbm.predict(test\_x)

accuracy = accuracy\_score(test\_y, y\_pred)

print("Accuracy Score of Gradient Boosting Classifier: ", accuracy)

from sklearn.metrics import confusion\_matrix

cm1 = confusion\_matrix(test\_y, y\_pred)

print('Confusion matrix\n\n', cm1)

import seaborn as sns

plt.figure(figsize=(8, 6))

sns.heatmap(cm1, annot=True, fmt='d', cmap='cividis', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

# Generate a classification report

clr1 = print(classification\_report(test\_y, y\_pred, zero\_division=0))

import matplotlib.pyplot as plt

import seaborn as sns

class\_report = classification\_report(test\_y, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='RdPu')

plt.title('Classification Report Heatmap')

plt.show()

**FRAMEWORK CODING:**

# !pip install --user --upgrade scikit-image

import tkinter as tk

from tkinter import ttk

%matplotlib inline

import math

import matplotlib.pyplot as plt

import os

from skimage import measure

from skimage.color import rgb2gray

from skimage.util import img\_as\_ubyte

from skimage import io

from skimage.feature import graycomatrix, graycoprops

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.metrics import roc\_auc\_score, roc\_curve, auc, precision\_recall\_fscore\_support

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from PIL import Image, ImageTk

from sklearn.model\_selection import train\_test\_split

import numpy as np

import pandas as pd

# Load your dataset here

imgList = os.listdir('Plant\_Village')

imgPath = []

for path in imgList:

pathImg = f'Plant\_Village/{path}'

imgPath.append(pathImg)

imgPath.sort()

ax = []

ay = []

az = []

aa = []

ab = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[0], levels=256)

ax.append(graycoprops(glcm, 'dissimilarity')[0, 0])

ay.append(graycoprops(glcm, 'correlation')[0, 0])

az.append(graycoprops(glcm, 'homogeneity')[0, 0])

aa.append(graycoprops(glcm, 'contrast')[0, 0])

ab.append(graycoprops(glcm, 'energy')[0, 0])

# 4.18879, 2.35619, 3.92699, 1.74533, 3.49066, 5.23599, 5.93412

bx = []

by = []

bz = []

ba = []

bb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[0.785398], levels=256)

bx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

by.append(graycoprops(glcm, 'correlation')[0, 0])

bz.append(graycoprops(glcm, 'homogeneity')[0, 0])

ba.append(graycoprops(glcm, 'contrast')[0, 0])

bb.append(graycoprops(glcm, 'energy')[0, 0])

cx = []

cy = []

cz = []

ca = []

cb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[1.0472], levels=256)

cx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

cy.append(graycoprops(glcm, 'correlation')[0, 0])

cz.append(graycoprops(glcm, 'homogeneity')[0, 0])

ca.append(graycoprops(glcm, 'contrast')[0, 0])

cb.append(graycoprops(glcm, 'energy')[0, 0])

dx = []

dy = []

dz = []

da = []

db = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[2.0944], levels=256)

dx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

dy.append(graycoprops(glcm, 'correlation')[0, 0])

dz.append(graycoprops(glcm, 'homogeneity')[0, 0])

da.append(graycoprops(glcm, 'contrast')[0, 0])

db.append(graycoprops(glcm, 'energy')[0, 0])

ex = []

ey = []

ez = []

ea = []

eb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[3.14159], levels=256)

ex.append(graycoprops(glcm, 'dissimilarity')[0, 0])

ey.append(graycoprops(glcm, 'correlation')[0, 0])

ez.append(graycoprops(glcm, 'homogeneity')[0, 0])

ea.append(graycoprops(glcm, 'contrast')[0, 0])

eb.append(graycoprops(glcm, 'energy')[0, 0])

fx = []

fy = []

fz = []

fa = []

fb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[4.18879], levels=256)

fx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

fy.append(graycoprops(glcm, 'correlation')[0, 0])

fz.append(graycoprops(glcm, 'homogeneity')[0, 0])

fa.append(graycoprops(glcm, 'contrast')[0, 0])

fb.append(graycoprops(glcm, 'energy')[0, 0])

imgName = os.listdir('Plant\_Village')

imgName.sort()

plant = pd.DataFrame(dict(label=0, image\_name=imgName, ax=ax, ay=ay, az=az, aa=aa, ab=ab, bx=bx, by=by, bz=bz, ba=ba, bb=bb, cx=cx, cy=cy, cz=cz, ca=ca, cb=cb, dx=dx, dy=dy, dz=dz, da=da, db=db, ex=ex, ey=ey, ez=ez, ea=ea, eb=eb, fx=fx, fy=fy,fz=fz, fa=fa, fb=fb ))

plant.rename(columns = {0: "image\_name", 1: "label"}, inplace = True)

plant = plant.sort\_values('image\_name')

plant.to\_csv('plant.csv', index=False)

plant.head()

df = pd.DataFrame(plant)

array=df.values

x\_feature=array[:,2:]

y\_label=array[:,2].astype('int')

print(x\_feature.shape)

print(y\_label.shape)

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(x\_feature,y\_label,test\_size=0.10,random\_state=7)

scaler\_norm = MinMaxScaler()

X\_train = scaler\_norm.fit\_transform(X\_train)

X\_test = scaler\_norm.fit\_transform(X\_test)

train\_x, test\_x, train\_y, test\_y = train\_test\_split(x\_feature, y\_label, test\_size=0.30)

# Initialize classifiers

dtc = DecisionTreeClassifier(criterion='gini')

gbm = GradientBoostingClassifier(n\_estimators=4000, learning\_rate=0.1, max\_depth=3, random\_state=42)

# Tkinter GUI

root = tk.Tk()

root.title("Classifier Metrics")

root.geometry("400x400")

# Load background image

background\_image = Image.open("sample1.jpg") # Replace with your image file

background\_photo = ImageTk.PhotoImage(background\_image)

background\_label = tk.Label(root, image=background\_photo)

background\_label.place(relwidth=1, relheight=1)

# Project label

project\_label = tk.Label(root, text="Plant disease identification from the drone captured images", font=("Helvetica", 12), bg="white")

project\_label.pack(pady=10)

# Labels for dataset information

r\_dataset\_label = tk.Label(root, text="Dataset: Plant\_Village", font=("Helvetica", 11),foreground="blue",width=20)

r\_dataset\_label.pack(pady=10, padx=10)

# Training Data Label

r\_train\_data\_label = tk.Label(root, text="Training Data: 70%", font=("Helvetica", 11),foreground="blue",width=20)

r\_train\_data\_label.pack(pady=10, padx=10)

# Testing Data Label

r\_test\_data\_label = tk.Label(root, text="Testing Data: 30%", font=("Helvetica", 11), foreground="blue",width=20)

r\_test\_data\_label.pack(pady=10, padx=10)

# Function to train classifiers

def train\_dtc\_classifier():

global dtc, train\_x, train\_y

dtc.fit(train\_x, train\_y)

print("DTC Classifier trained successfully.")

def train\_gbm\_classifier():

global gbm, train\_x, train\_y

gbm.fit(train\_x, train\_y)

print("GBM Classifier trained successfully.")

# Function to calculate metrics and show charts for DTC

def show\_dtc\_metrics():

global dtc, test\_x, test\_y

# Predict the Test set results

y\_pred = dtc.predict(test\_x)

# Confusion Matrix

cm\_dtc = confusion\_matrix(test\_y, y\_pred)

print('Confusion matrix of dtc\n\n', cm\_dtc)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_dtc, annot=True, fmt='d', cmap='cividis', cbar=False)

plt.title('Confusion Matrix of dtc')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_dtc():

# Predict the Test set results

y\_pred = dtc.predict(test\_x)

# Classification Report

clr\_dtc = print(classification\_report(test\_y, y\_pred, zero\_division=0))

# Plot Classification Report

class\_report = classification\_report(test\_y, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap of dtc')

plt.show()

def calculate\_accuracy\_dtc():

global dtc, test\_x, test\_y

# Predict the Test set results

y\_pred = dtc.predict(test\_x)

# Accuracy

accuracy\_dtc = accuracy\_score(test\_y, y\_pred)

print('Model accuracy score of dtc:', accuracy\_dtc)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_dtc], color='blue')

plt.title('Model Accuracy of dtc')

plt.ylabel('Accuracy')

plt.show()

# Function to calculate metrics and show charts for GBM

def show\_gbm\_metrics():

global gbm, test\_x, test\_y

# Predict the Test set results

y\_pred = gbm.predict(test\_x)

# Confusion Matrix

cm\_gbm = confusion\_matrix(test\_y, y\_pred)

print('Confusion matrix of gbm\n\n', cm\_gbm)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_gbm, annot=True, fmt='d', cmap='cividis', cbar=False)

plt.title('Confusion Matrix of gbm')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_gbm():

# Predict the Test set results

y\_pred = gbm.predict(test\_x)

# Classification Report

clr\_gbm = print(classification\_report(test\_y, y\_pred, zero\_division=0))

# Plot Classification Report

class\_report = classification\_report(test\_y, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap of gbm')

plt.show()

def calculate\_accuracy\_gbm():

global gbm, test\_x, test\_y

# Predict the Test set results

y\_pred = gbm.predict(test\_x)

# Accuracy

accuracy\_gbm = accuracy\_score(test\_y, y\_pred)

print('Model accuracy score of gbm:', accuracy\_gbm)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_gbm], color='blue')

plt.title('Model Accuracy of gbm')

plt.ylabel('Accuracy')

plt.show()

# DTC Frame

dtc\_frame = tk.Frame(root)

dtc\_frame.pack(side=tk.TOP, pady=10)

# DTC Train Button

dtc\_train\_button = tk.Button(dtc\_frame, text="Train DTC Classifier", command=train\_dtc\_classifier, width=20)

dtc\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Metrics Button

dtc\_metrics\_button = tk.Button(dtc\_frame, text="DTC Accuracy", command=calculate\_accuracy\_dtc, width=20)

dtc\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Matrix Button

dtc\_matrix\_button = tk.Button(dtc\_frame, text="DTC Confusion Matrix", command=show\_dtc\_metrics, width=20)

dtc\_matrix\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Matrix Button

dtc\_report\_button = tk.Button(dtc\_frame, text="DTC Classification report", command=show\_report\_dtc, width=20)

dtc\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Frame

gbm\_frame = tk.Frame(root)

gbm\_frame.pack(side=tk.TOP, pady=10)

# RFC Train Button

gbm\_train\_button = tk.Button(gbm\_frame, text="Train GBM Classifier", command=train\_gbm\_classifier, width=20)

gbm\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Metrics Button

gbm\_metrics\_button = tk.Button(gbm\_frame, text="GBM Accuracy", command=calculate\_accuracy\_gbm, width=20)

gbm\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Matrix Button

gbm\_matrix\_button = tk.Button(gbm\_frame, text="GBM Confusion Matrix", command=show\_gbm\_metrics, width=20)

gbm\_matrix\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC report Button

gbm\_report\_button = tk.Button(gbm\_frame, text="GBM Classification report", command=show\_report\_gbm, width=20)

gbm\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# Run the Tkinter event loop

root.mainloop()

**RESULTS AND DISCUSSION:**

**Dataset:**

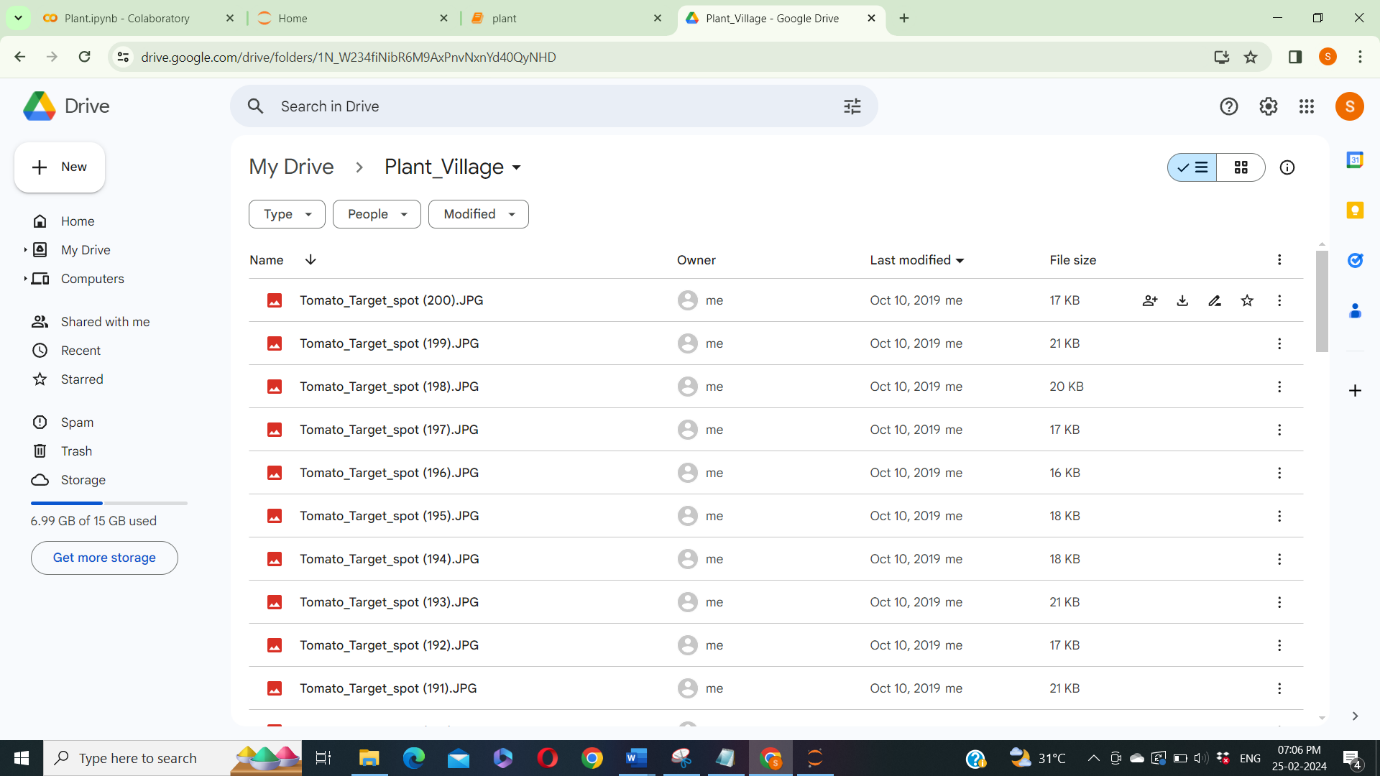


Figure 1: Image dataset

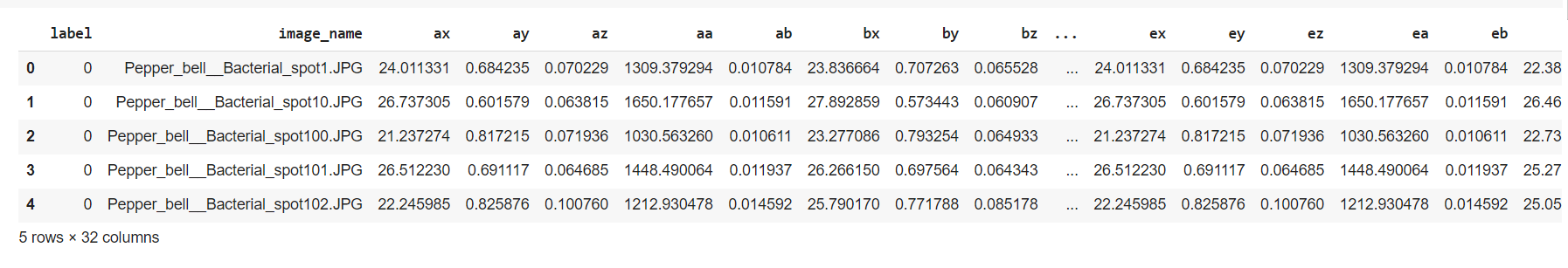


Figure 2: CSV dataset

**Results:**

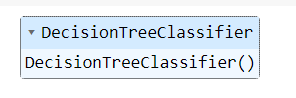


Figure 3: Decision Tree classifier algorithm



Figure 4: Accuracy calculation of Decision Tree classifier algorithm

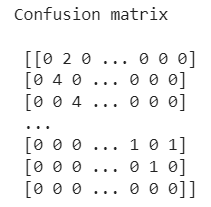


Figure 5: Confusion matrix calculation of Decision Tree classifier algorithm

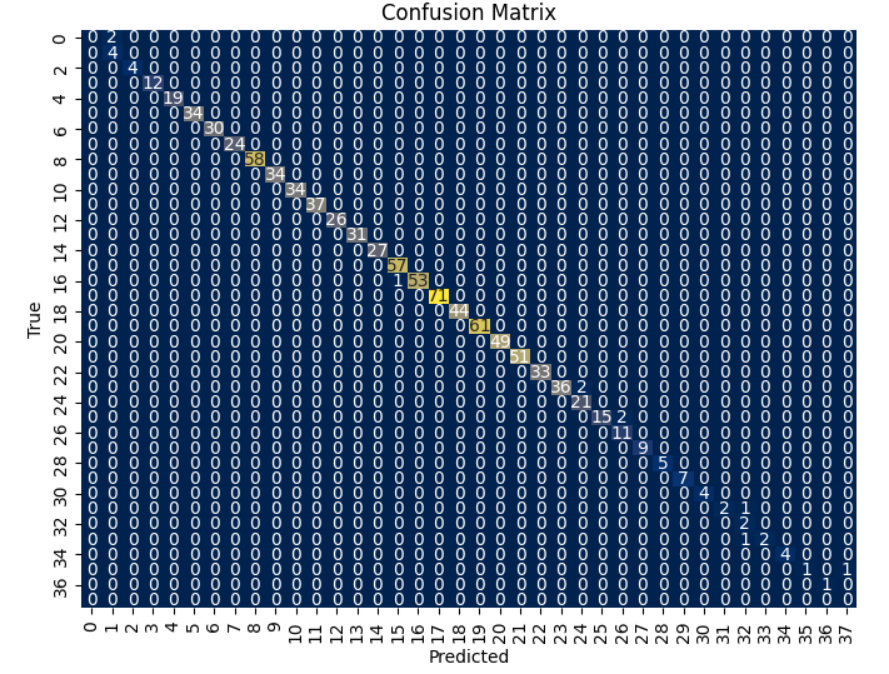


Figure 6: Confusion matrix graph of Decision Tree classifier algorithm

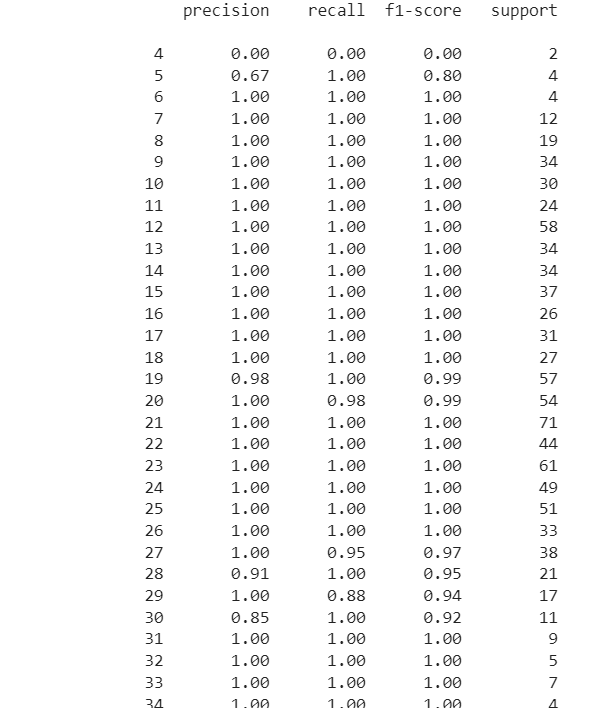


Figure 7: Classification report calculation of Decision Tree classifier algorithm

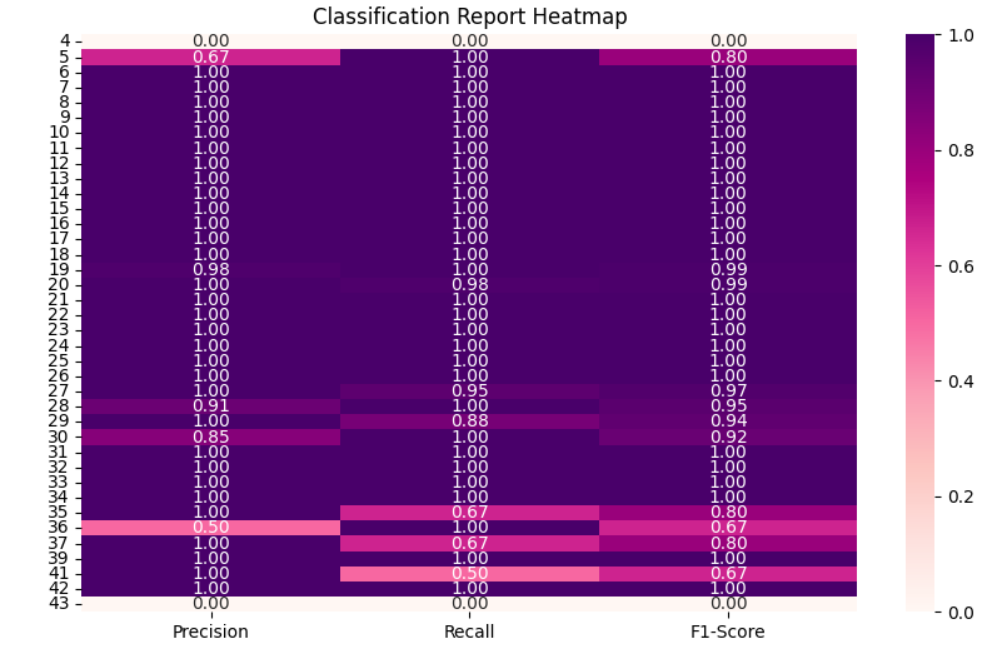


Figure 8: Classification report graph of Random Forest classifier algorithm

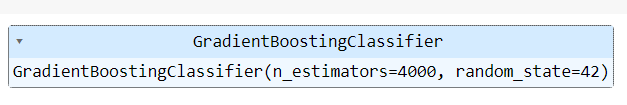


Figure 9: Gradient Boosting classifier algorithm



Figure 10: Accuracy calculation of Gradient Boosting classifier algorithm

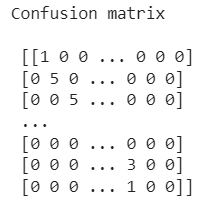
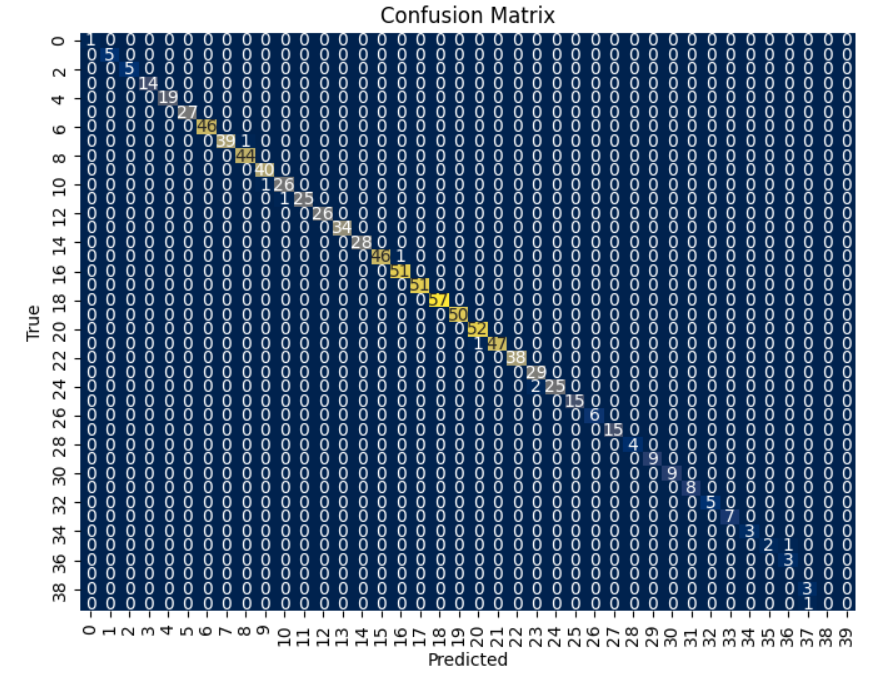


Figure 11: Confusion matrix of Gradient Boosting classifier algorithm

Figure 12: Confusion matrix graph of Gradient Boosting classifier algorithm

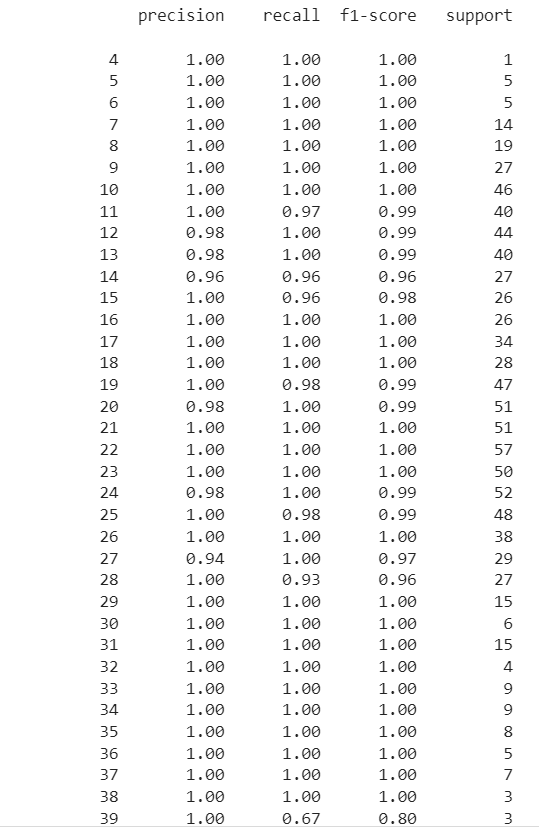


Figure 13: Classification report of Gradient Boosting classifier algorithm

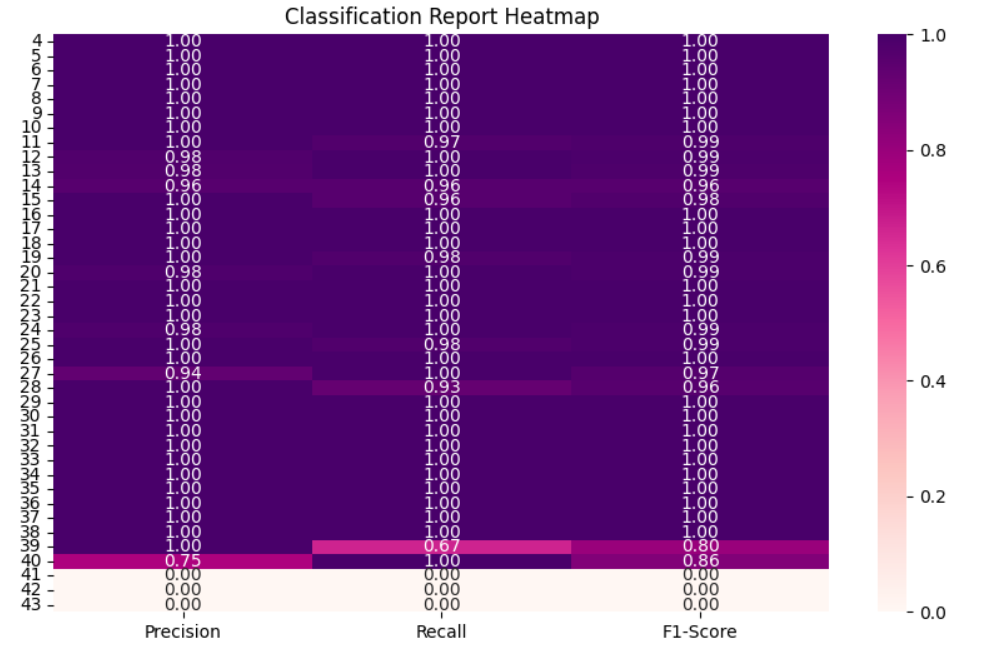


Figure 14: Classification report graph of Gradient Boosting classifier algorithm

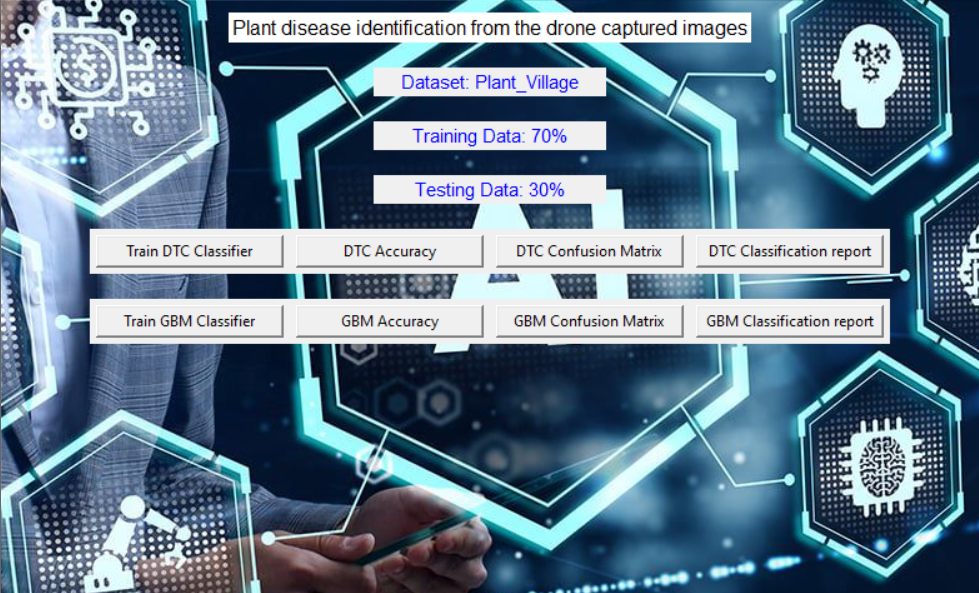


Figure 15: Frame work design





Figure 16: Classifier training

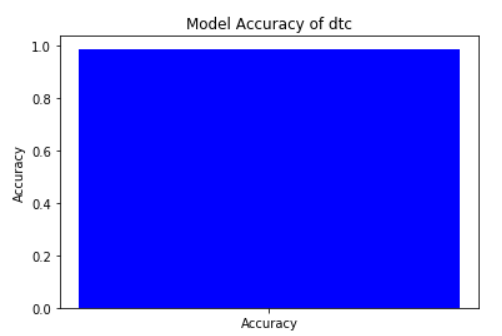


Figure 17: Accuracy graph of Decision Tree classifier algorithm

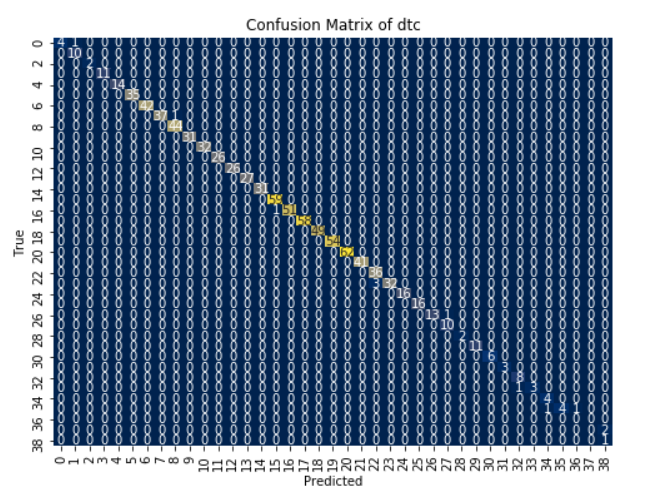


Figure 18: Confusion matrix graph of Decision Tree algorithm

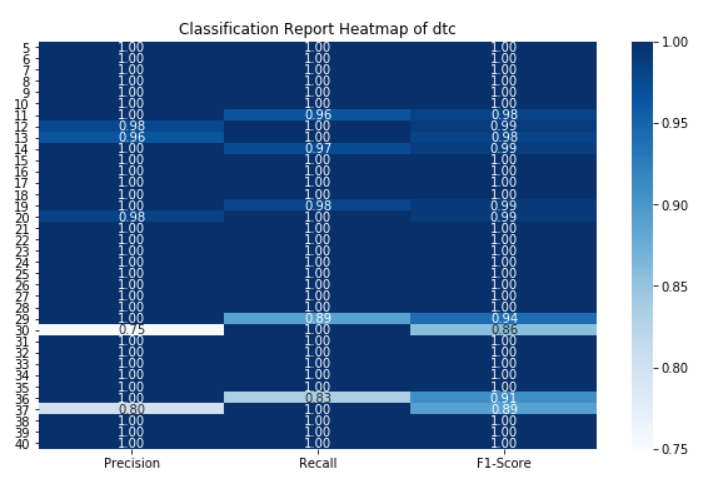


Figure 19: Classification report graph of Decision Tree classifier algorithm

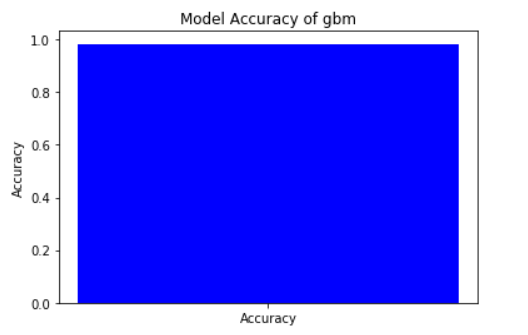


Figure 20: Accuracy graph of Gradient Boosting classifier algorithm

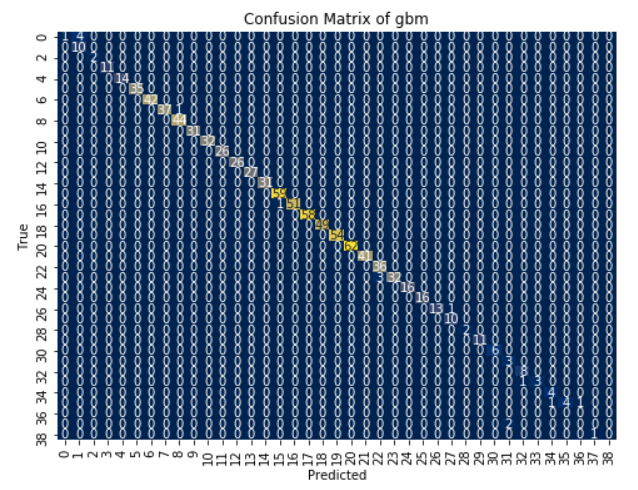


Figure 21: Confusion matrix graph of Gradient Boosting classifier algorithm

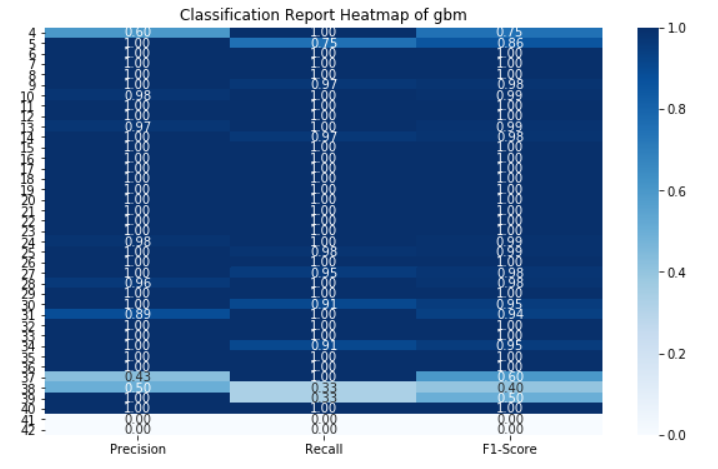


Figure 22: Classification report graph of Gradient Boosting classifier algorithm

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 4 | 1.00 | 0.80 | 0.89 | 5 |
| 5 | 0.80 | 1.00 | 0.89 | 4 |
| 6 | 1.00 | 1.00 | 1.00 | 5 |
| 7 | 1.00 | 1.00 | 1.00 | 8 |
| 8 | 1.00 | 1.00 | 1.00 | 18 |
| 9 | 1.00 | 1.00 | 1.00 | 36 |
| 10 | 0.97 | 1.00 | 0.99 | 39 |
| 11 | 1.00 | 0.97 | 0.99 | 39 |
| 12 | 1.00 | 1.00 | 1.00 | 45 |
| 13 | 0.96 | 1.00 | 0.98 | 27 |
| 14 | 1.00 | 0.96 | 0.98 | 28 |
| 15 | 1.00 | 1.00 | 1.00 | 35 |
| 16 | 1.00 | 1.00 | 1.00 | 19 |
| 17 | 0.97 | 1.00 | 0.98 | 30 |
| 18 | 1.00 | 0.97 | 0.98 | 31 |
| 19 | 1.00 | 0.98 | 0.99 | 47 |
| 20 | 0.98 | 1.00 | 0.99 | 64 |
| 21 | 1.00 | 1.00 | 1.00 | 55 |
| 22 | 1.00 | 1.00 | 1.00 | 50 |
| 23 | 1.00 | 1.00 | 1.00 | 51 |
| 24 | 1.00 | 0.98 | 0.99 | 51 |
| 25 | 0.98 | 1.00 | 0.99 | 49 |
| 26 | 1.00 | 1.00 | 1.00 | 39 |
| 27 | 0.98 | 1.00 | 0.99 | 42 |
| 28 | 1.00 | 0.95 | 0.97 | 19 |
| 29 | 1.00 | 1.00 | 1.00 | 17 |
| 30 | 1.00 | 1.00 | 1.00 | 9 |
| 31 | 0.92 | 1.00 | 0.96 | 11 |
| 32 | 1.00 | 0.83 | 0.91 | 6 |
| 33 | 1.00 | 1.00 | 1.00 | 7 |
| 34 | 1.00 | 1.00 | 1.00 | 4 |
| 35 | 1.00 | 1.00 | 1.00 | 9 |
| 36 | 1.00 | 1.00 | 1.00 | 6 |
| 37 | 1.00 | 1.00 | 1.00 | 4 |
| 38 | 1.00 | 0.50 | 0.67 | 4 |
| 39 | 0.60 | 0.75 | 0.67 | 4 |
| 40 | 0.50 | 1.00 | 0.67 | 3 |
| 41 | 0.00 | 0.00 | 0.00 | 1 |
| 42 | 1.00 | 0.50 | 0.67 | 2 |
| accuracy |  |  | 0.99 | 923 |
| macro avg | 0.94 | 0.93 | 0.93 | 923 |
| weighted avg | 0.99 | 0.99 | 0.99 | 923 |

Table 1: classification report of DTC

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support | |
| 4 | 1.00 | 1.00 | 1.00 | | 1 |
| 5 | 1.00 | 1.00 | 1.00 | | 5 |
| 6 | 1.00 | 1.00 | 1.00 | | 5 |
| 7 | 1.00 | 1.00 | 1.00 | | 14 |
| 8 | 1.00 | 1.00 | 1.00 | | 19 |
| 9 | 1.00 | 1.00 | 1.00 | | 27 |
| 10 | 1.00 | 1.00 | 1.00 | | 46 |
| 11 | 1.00 | 0.97 | 0.99 | | 40 |
| 12 | 0.98 | 1.00 | 0.99 | | 44 |
| 13 | 0.98 | 1.00 | 0.99 | | 40 |
| 14 | 0.96 | 0.96 | 0.96 | | 27 |
| 15 | 1.00 | 0.96 | 0.98 | | 26 |
| 16 | 1.00 | 1.00 | 1.00 | | 26 |
| 17 | 1.00 | 1.00 | 1.00 | | 34 |
| 18 | 1.00 | 1.00 | 1.00 | | 28 |
| 19 | 1.00 | 0.98 | 0.99 | | 47 |
| 20 | 0.98 | 1.00 | 0.99 | | 51 |
| 21 | 1.00 | 1.00 | 1.00 | | 51 |
| 22 | 1.00 | 1.00 | 1.00 | | 57 |
| 23 | 1.00 | 1.00 | 1.00 | | 50 |
| 24 | 0.98 | 1.00 | 0.99 | | 52 |
| 25 | 1.00 | 0.98 | 0.99 | | 48 |
| 26 | 1.00 | 1.00 | 1.00 | | 38 |
| 27 | 0.94 | 1.00 | 0.97 | | 29 |
| 28 | 1.00 | 0.93 | 0.96 | | 27 |
| 29 | 1.00 | 1.00 | 1.00 | | 15 |
| 30 | 1.00 | 1.00 | 1.00 | | 6 |
| 31 | 1.00 | 1.00 | 1.00 | | 15 |
| 32 | 1.00 | 1.00 | 1.00 | | 4 |
| 33 | 1.00 | 1.00 | 1.00 | | 9 |
| 34 | 1.00 | 1.00 | 1.00 | | 9 |
| 35 | 1.00 | 1.00 | 1.00 | | 8 |
| 36 | 1.00 | 1.00 | 1.00 | | 5 |
| 37 | 1.00 | 1.00 | 1.00 | | 7 |
| 38 | 1.00 | 1.00 | 1.00 | | 3 |
| 39 | 1.00 | 0.67 | 0.80 | | 3 |
| 40 | 0.75 | 1.00 | 0.86 | | 3 |
| 41 | 0.00 | 0.00 | 0.00 | | 0 |
| 42 | 0.00 | 0.00 | 0.00 | | 3 |
| 43 | 0.00 | 0.00 | 0.00 | | 1 |
| accuracy |  |  | 0.99 | | 923 |
| macro avg | 0.91 | 0.91 | 0.91 | | 923 |
| weighted avg | 0.99 | 0.99 | 0.99 | | 923 |

Table 2: classification report of GBC

The classification report is a performance evaluation tool that shows the precision, recall, f1-score, for each class in a classification problem. In training images using the deep learning model, the classification report would provide information about how well the model performed in classifying images into different categories. The precision represents the percentage of correctly classified images among all the images classified as belonging to a specific class. The recall represents the percentage of correctly classified images among all the images that actually belong to a specific class. The f1-score is a harmonic mean of precision and recall, and support represents the number of images in each class.

The accuracy has been calculated for the model that has been implemented, and the result for the model is compared in Table:

|  |  |
| --- | --- |
| Algorithms | Accuracy |
| DTC | 98 |
| GBM | 99 |

Table 3: Accuracy comparison of algorithm.

|  |  |  |
| --- | --- | --- |
| Dataset Count | Training Value | Testing Value |
| 3706 | 70 | 30 |

Table 4: Consist of dataset count, Training and Testing percentage.

**CONCLUSION**

In conclusion, employing a combination of image processing techniques, such as the Gray-Level Co-occurrence Matrix (GLCM) for feature extraction, and machine learning algorithms like Decision Trees and Gradient Boosting, offers a robust approach to plant disease identification from drone-captured images. Preprocessing steps like Min-Max scaling ensure that features are appropriately scaled for analysis, enhancing the performance of the models. By leveraging these methodologies, accurate and efficient identification of plant diseases can be achieved, enabling timely interventions to mitigate crop damage and optimize agricultural productivity. The integration of advanced technologies in agriculture holds promise for revolutionizing crop management practices and addressing global food security challenges.

**REFERNCES**

Barbedo, Jayme Garcia Arnal. "Plant disease identification from individual lesions and spots using deep learning." Biosystems Engineering 191 (2020): 96-107.

Sladojevic, Sanja, et al. "Deep learning for plant disease detection and diagnosis." Computers in Industry 100 (2018): 103-113.

Mohanty, Sharada Prasad, David P. Hughes, and Marcel Salathe. "Using deep learning for image-based plant disease detection." Frontiers in plant science 7 (2016): 1419.

Sa, Inkyu, et al. "Deep learning classification of citrus greening using airborne hyperspectral imagery." Remote Sensing 11.14 (2019): 1672.

Ferentinos, Konstantinos P. "Deep learning models for plant disease detection and diagnosis." Computers and Electronics in Agriculture 145 (2018): 311-318.

Ghosal, Subhankar, et al. "An explainable deep machine vision framework for plant stress phenotyping." Proceedings of the National Academy of Sciences 115.18 (2018): 4613-4618.

Sharma, Vikas, et al. "A comprehensive review on application of deep learning with its interpretability for agricultural pest and disease detection." Computers and Electronics in Agriculture 180 (2021): 105862.