ABSTRACT

Early and accurate detection of corn diseases is crucial for effective crop management and minimizing yield losses. However, traditional methods often involve delays or require specialized expertise, hindering timely intervention. This project addresses these challenges by developing an automatic corn leaf disease recognition system using machine learning (ML) on the NVIDIA Jetson Nano platform.

The proposed system prioritizes timely detection by leveraging the power of AI and ML on the Jetson Nano, enabling rapid disease identification directly in the field. Accessibility is a key consideration, ensuring the system is user-friendly and deployable by farmers with varying technical backgrounds. Furthermore, the system is designed to be adaptable, capable of learning from diverse datasets to address the unique disease challenges faced by different corn species and regions.

By incorporating on-device processing capabilities through the Jetson Nano, the system eliminates the need for reliance on external resources and facilitates real-time disease monitoring in agricultural settings. This project contributes to the development of practical and accessible tools for farmers, empowering them to make informed decisions for improved crop health and yield optimization.

1.Introduction

1.1. Overview

The aim of this document is to present a Machine learning approaches for the detection of corn leaf diseases. Corn cultivation plays a significant role in the agricultural industry, and the health and quality of corn leaves directly impact the overall yield and productivity of corn yards. Early detection and accurate diagnosis of diseases affecting corn leaves are crucial for effectively managing and mitigating their impact on crop health.

1.2. Motivation

corn leaf diseases can have devastating leading to reduced yield, lower crop quality, and increased economic losses for corn yard owners and farmers. Traditional methods of disease detection often require manual inspection of leaves, which can be time-consuming, subjective, and prone to error. There is a need for an efficient and reliable automated system that can accurately identify and classify various diseases affecting corn leaves.

This Machine Learning approaches aims to leverage the power of deep learning and various learning techniques to automate the detection process, enabling quick and accurate identification of corn leaf diseases. By providing early detection and diagnosis, farmers and corn yard owners can swiftly implement appropriate measures to control and manage the diseases, ultimately leading to improved crop yield and economic sustainability.

1.3. Problem Definition and Objectives

The problem at hand is the detection of corn leaf diseases using a Machine learning approaches. The main objective of this project is to develop a robust and accurate system that can effectively classify different types of corn leaf diseases. This system should be capable of analysing input images of corn leaves and providing real-time disease diagnosis.

The specific objectives include:

- Collecting and curating a comprehensive dataset of corn leaf images depicting various diseases.
- Preprocessing and augmenting the dataset to enhance the Machine Learning model's training capabilities.
- Designing and training a Machine Learning models that can accurately classify and diagnose corn leaf diseases.
- Evaluating the performance of the machine learning models in terms of accuracy, precision, recall, and F1-score.
- Implementing the machine learning model into a user-friendly application or interface for easy deployment and usability.

1.4. Project Scope and Limitation

The scope of this project is focused on the development and implementation of a machine learning approaches for the detection of corn leaf diseases. The system will be trained to classify common diseases such as Blight, Common rust and grey leaf spot along with a healthy. The dataset used for training will include a diverse range of images depicting these diseases. However, it is important to note that this system may not be able to detect extremely rare or uncommon diseases that have limited representation in the training dataset.

Additionally, the performance of the system may be affected if the input images are of low quality, have poor lighting conditions, or contain significant noise. It is crucial to ensure that the input images provided to the system are clear and well-captured.

1.5. Methodologies and Problem Solving

To address the problem of crop leaf disease detection, a machine learning approaches will be employed. The methodology involves the following steps:

1) Data collection and preprocessing: A diverse dataset of corn leaf images depicting various diseases will be collected and curated. The dataset will undergo preprocessing techniques such as resizing, normalization, and noise reduction to improve the model's training capabilities.

2) Model architecture design: A suitable ML architecture will be designed, taking into consideration factors such as the complexity of the problem, the size of the dataset, and the available computational resources. The model will be designed to extract meaningful features from the input images.

3) Model training and evaluation: The designed machine learning models will be trained on the pre-processed dataset. The training process involves optimizing model parameters through different algorithms (Support vector machine, logistic regression, KNN, Random Forest algorithm, Decision tree classification, Navie bias classifier, Adoptive boosting). The performance of the model will be evaluated using appropriate metrics such as accuracy, precision, recall, and F1-score.

4) Deployment and usability: Once the model is trained and evaluated, it will be implemented into a user-friendly application or interface for easy deployment and usability. The application will allow users to upload images of corn leaves and obtain real-time disease diagnosis and recommendations for disease management.

By implementing this ML approaches, the proposed system aims to provide an effective and automated solution for grape leaf disease detection, enabling early diagnosis and timely intervention for better crop yield and overall yield health.

2. Literature Survey

2.1 Plant Disease Detection using digital image Processing

Khirade and Patil (2015) explored the potential of image processing for automatic plant disease detection in their study published in the IEEE [IEEE reference]. They argue that traditional manual monitoring is inefficient, and image processing offers a promising alternative. The approach involves capturing images of leaves followed by several steps: pre-processing to improve image quality, segmentation to isolate diseased regions, feature extraction to quantify disease characteristics, and finally, classification to identify the specific disease. Their research focused on segmentation and feature extraction techniques suitable for accurate disease detection in agricultural applications. Additionally, they discussed the use of Artificial Neural Networks (ANNs) for disease classification, highlighting methods like self-organizing feature maps, backpropagation algorithms, and Support Vector Machines (SVMs). Their work suggests that image processing combined with ANNs has the potential for accurate and efficient plant disease identification.

2.2 Plant Disease Detection Using Deep Learning

A recent research paper by Hirani et al. (2021) explores the use of deep learning for plant disease detection through image processing. The authors argue that accurate disease identification is essential for healthy crops. The paper reviews traditional image processing techniques for segmenting diseased areas, extracting relevant features, and classifying the specific disease. They then discuss the potential of artificial neural networks (ANNs) like self-organizing maps and support vector machines (SVMs) for disease classification. Interestingly, the paper explores transformers, a new development in deep learning originally used for natural language processing and finds them surprisingly effective for computer vision tasks as well. Given limitations in resource availability, the paper highlights small transformer networks (STNs) as a particularly promising approach for plant disease detection. Overall, the research suggests that deep learning, especially STNs, has the potential to significantly improve plant disease detection and promote sustainable agriculture.

2.3 Leaf Disease Detection and Classification based on Machine Learning

A 2020 paper by Kumar et al. investigates using machine learning for leaf disease detection and classification. The authors emphasize the importance of early disease detection for both agricultural health and economic reasons. They highlight the challenge of traditional methods that rely on manual monitoring, which can be time-consuming and labour-intensive. To address this, they propose a machine learning approach that automates disease detection and offers improved accuracy compared to existing methods.

The proposed system utilizes various images containing diseased leaves. A segmentation technique called k-means clustering is used to isolate the diseased areas from healthy leaf tissue. Then, features are extracted using a method called Gray Level Co-occurrence Matrix (GLCM). Finally, a Support Vector Machine (SVM) classifier is employed to categorize the specific disease based on the extracted features. The system is tested on a dataset of leaves affected by various diseases, and the results demonstrate high accuracy in identifying both the type and extent of the disease.

2.4 Disease Detection in Plants using Convolutional Neural Network

In a 2020 study by Sharath et al., the authors explore using convolutional neural networks (CNNs) for plant disease detection in the field of horticulture. They acknowledge the significant economic impact of plant diseases on farmers' livelihoods. The paper highlights the challenges of early disease detection in horticulture, especially compared to row crops. To address this, they propose a mobile application for Android that leverages CNNs to automate disease identification. The app captures images of potentially infected crops, enhances the image quality, performs image segmentation to isolate diseased areas, and then utilizes a CNN to classify the specific disease. This allows for early detection and intervention, potentially improving crop yields and farmer income. The study demonstrates an accuracy of nearly 91% for disease detection using CNNs, with the potential for further improvement through enhanced image processing techniques. The app also provides farmers with recommendations for preventive measures, pesticides, fertilizers, and weed control specific to the identified disease. The authors propose that incorporating environmental factors like humidity, temperature, and rainfall into these recommendations could further increase farm productivity.

2.5 Plant Leaf Detection and Disease Recognition using Deep Learning

Militante et al. (2019) present a study on using deep learning for plant leaf disease detection and recognition. Deep learning advancements in computer vision have enabled the development of a system for diagnosing plant diseases through image analysis. The system is trained on a dataset of 35,000 images containing both healthy and diseased leaves. This allows the model to not only detect the presence of disease but also identify the specific type of disease afflicting the plant. The trained model achieves an impressive accuracy rate of 96.5% for disease detection and recognition, and even achieves 100% accuracy in identifying plant varieties and the specific diseases they are infected with. The authors emphasize the importance of early disease detection for agricultural productivity. This system has the potential to be a valuable tool for farmers by allowing them to quickly and accurately diagnose plant diseases, facilitating timely intervention and improved crop yields. The paper also acknowledges the potential for further improvement by expanding the dataset to include additional plant varieties and diseases, as well as exploring different convolutional neural network architectures and optimization techniques.

2.6 Plant Disease Detection using CNN

Shrestha and Das (2020) investigate the application of convolutional neural networks (CNNs) for plant disease detection in an Indian agricultural context. The paper acknowledges the importance of crop health for the Indian economy and the need for methods to protect crops from disease. They propose a CNN-based system for disease detection using leaf image analysis. The model is trained on a dataset containing images of both healthy and diseased plant leaves, encompassing 15 different cases. While the model achieves an accuracy of 88.80% in disease identification, the authors acknowledge there is room for improvement. They envision this technology being used by both home gardeners and farmers to monitor plant health and promote early disease intervention. The paper suggests that future development could involve creating a mobile application that not only identifies diseases but also provides recommendations for treatment.

2.7 Plant Disease Detection Using Machine Learning

A 2018 research paper explored the use of machine learning for plant disease detection. The authors highlighted the challenge of traditional methods in resource-limited areas and emphasized the promise of image-based classification techniques. Their approach utilized Random Forest classifiers to distinguish between healthy and diseased papaya leaves using a dataset they created. Feature extraction relied on Histogram of Oriented Gradients (HOG), achieving an accuracy of around 70% with 160 images. The paper acknowledges the potential for improvement with more data and explores avenues for incorporating other feature extraction methods like SIFT, SURF, and DENSE with Bag-of-Visual-Words (BOVW) to enhance accuracy.

2.8 Plant Leaf Disease Detection and Classification Based on CNN with LVQ Algorithm

In the realm of plant disease detection, Sardogan et al. (2020) introduced a method leveraging Convolutional Neural Networks (CNNs) and Learning Vector Quantization (LVQ) algorithms. Their focus was on tomato leaf diseases, targeting early detection of bacterial spot, late blight, Septoria leaf spot, and yellow curved leaf diseases. They built upon the established success of CNNs in image recognition by proposing a system that extracts features based on RGB color components within the CNN architecture. The LVQ algorithm then utilizes these extracted features to classify the tomato leaf images into the four disease categories. Their experiment using a dataset of 500 tomato leaf images achieved promising results, demonstrating the effectiveness of this approach in identifying tomato leaf diseases. The authors acknowledge the potential for further improvement by exploring different filter types and convolution sizes within the CNN for potentially higher classification accuracy.

3.Machine Learning

Machine learning is a game-changing field that allows computers to learn from data and gradually become more efficient over time without the need for explicit programming. It includes a wide variety of methods and algorithms, such as reinforcement learning, unsupervised learning, and supervised learning. To generate predictions or categorize new data points, models are trained on labelled data in supervised learning. While reinforcement learning focuses on empowering agents to make decisions by learning from trial and error through contact with an environment, unsupervised learning focuses on finding patterns and structures within unlabelled data. Tasks including image identification, natural language processing, recommendation systems, and autonomous driving have been made possible by machine learning, which has transformed a number of industries, including healthcare, banking, transportation, and entertainment. Its constant improvements open up new avenues for exploration, spurring creativity and changing how humans engage with data and technology.

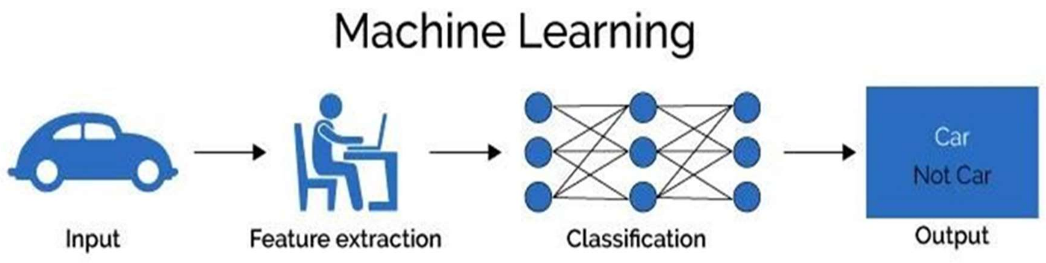


Fig 3.1 Machine Learning

3.1 Supervised Learning

A fundamental technique in artificial intelligence is supervised machine learning, in which computers are trained on labelled data in order to anticipate or make judgments. Using input-output pairings supplied during the training phase, the algorithm is trained in this manner with the goal of precisely mapping input data to corresponding output labels. Through this process, the algorithm is able to forecast results for fresh, unseen data and generalize what it has learned. In supervised learning, a variety of methods are used, such as logistic regression, decision trees, support vector machines (SVMs), and neural networks for categorical output prediction, and linear regression for continuous output prediction. During training, the model's parameters are iteratively adjusted, frequently with the use of optimization techniques like gradient descent, to reduce the difference between the model's anticipated outputs and the actual labels in the training data. Applications for supervised learning are found in many different fields, including picture classification, sentiment analysis in social media, medical diagnosis, and spam detection in emails. The representativeness and quality of the training data, along with the choice of suitable features and algorithms, are critical factors that determine how effective supervised learning is. Evaluation measures that are essential for evaluating model performance and generalization skills include accuracy, precision, recall, and F1 score. All things considered, supervised machine learning is essential for resolving practical issues since it uses labelled data to provide predictions and judgments that are well-informed.

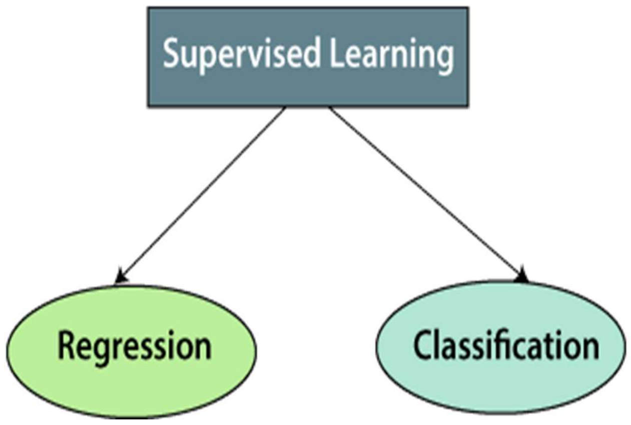


Fig 3.2 Supervised Learning

3.2 Unsupervised Learning

An important area of artificial intelligence is called "unsupervised machine learning," in which algorithms are trained to identify patterns or structures in incoming data without the need for explicit instructions or outputs that have been labelled. Unsupervised learning uses unlabelled data and relies only on the underlying structure or relationships within the dataset, in contrast to supervised learning, which trains the algorithm on labelled data to produce predictions or judgments. Finding hidden patterns, combining related data points, or lowering the dimensionality of the data for simpler analysis are the main objectives of unsupervised learning. In unsupervised learning, clustering algorithms like K means and hierarchical clustering are frequently used to divide data into groups according to similarity or distance measures. Dimensionality reduction is another popular method that attempts to express high-dimensional data in a lower dimensional space while maintaining its key characteristics. Popular techniques for reducing dimensionality include principal component analysis (PCA) and distributed stochastic neighbour embedding (t-SNE). Applications for unsupervised learning can be found in many different fields, such as data compression, customer segmentation, and anomaly detection. In exploratory data analysis, where insights can be extracted from huge, unstructured databases, it is very helpful. However, since there are no established labels to compare accuracy against, assessing the success of unsupervised learning systems can be difficult. Rather, evaluation frequently entails quantitative measurements like the silhouette score for grouping algorithms or qualitative review by subject matter experts. All 7 things considered, unsupervised machine learning is essential for revealing hidden structures and patterns in data, opening the door to further research and discoveries in a variety of domains.

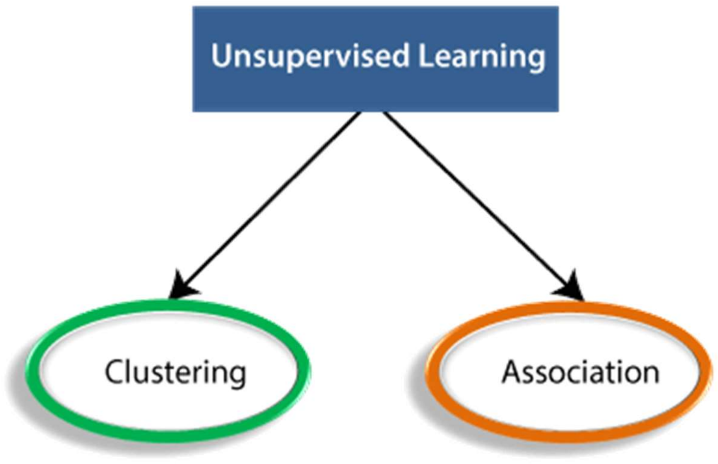


Fig 3.3: Unsupervised Learning

3.3 Semi-Supervised Learning

By using a dataset with both labelled and unlabelled data, semi-supervised machine learning combines elements of supervised and unsupervised learning. Its goal is to use the labelled subset to direct learning while utilizing the large amount of unlabelled data to improve model performance and generalization. In order to infer missing labels or increase overall model accuracy, techniques frequently combine supervised initialization with iterative refinement utilizing unlabelled data through approaches like self-training or co-training, as well as graph-based or generative models. Semi-supervised learning is a useful method for training machine learning models, especially in situations where labelled data is hard to come by or prohibitively expensive. It has applications in many different fields, such as image and speech recognition, natural language processing, and bioinformatics. However, maintaining data quality and striking the right balance between labelled and unlabelled samples can be difficult. However, current research keeps improving methods, emphasizing the potential of semi-supervised learning to advance machine learning for practical issue resolution.

4. SYSTEM DESIGN

4.1. Design Overview

This project investigates the application of machine learning techniques for automated corn leaf disease identification. The system is designed to analyze corn leaf images and classify them into healthy or diseased categories, potentially with further disease type recognition.

4.1.1 Image Acquisition

This module captures images of corn leaves using a camera or pre-existing image datasets.

4.1.2 Pre-processing

- Images are pre-processed to ensure consistency and improve model performance. This may involve resizing, colour normalization, and noise reduction.
- Data augmentation techniques (e.g., rotation, flipping) can be implemented to artificially expand the dataset and improve model robustness.

4.1.3 Machine Learning Core

This is the heart of the system, where different ML algorithms are employed to analyse the pre-processed images. You've mentioned exploring various techniques, so here are some specific examples to Classification Algorithms like Support Vector Machines (SVM), Random Forest, K-Nearest Neighbours (KNN), Ada Boost, Decision Tree, Logistic Regression, Naïve Bays.

4.1.4 Training and Evaluation:

The chosen ML models are trained on a labelled dataset containing images of healthy and diseased corn leaves. Each image is associated with a specific disease type (optional for initial classification).

Model performance is evaluated using metrics like accuracy, precision, and recall on a separate test dataset.

3.2. Model Training

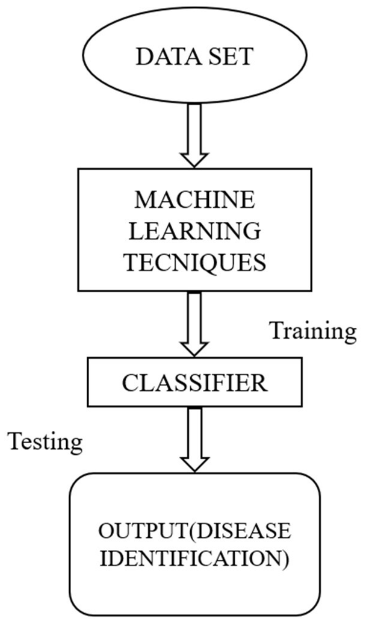


Fig 4.1: model training

5. REQUIREMENT ANALYSIS

5.1. Functional Requirements

A functional requirement defines a function of a system or its component. Function is described as a set of inputs, the behaviour and outputs. Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish. Behavioural requirements describing all cases where the system uses the functional requirements are captured in use cases. Functional requirements are supposed by non-functional requirements (also known as quality requirements), which impose constraints on the design or implementation (such as performance requirements, security or reliability).

As defined in requirements engineering, functional requirements specify particular results of a system. This should be contrasted with non-functional requirements which specify overall characteristics such as cost and reliability. Functional requirements drive the application architecture of a system, while non-functional requirements drive the technical architecture of a system.

- Functional Requirements concern the specific functions delivered by the system.
- So, functional requirements are statements of the services that the system must provide.
- The functional requirements of the system should be both complete and consistent.
- Completeness means that all the services required by the user should be defined.
- Consistency means that requirements should not have any contradictory definitions.
- The requirements are usually described in a fairly abstract way. However, functional system requirements describe the system function in detail, its inputs and outputs, exceptions and soon.
- Take user id and password match it with corresponding file entries. If a match is found then continue else raise an error message.

5.2. Non-Functional Requirements

Non-functional Requirements refer to the constraints or restrictions on the system. They may relate to emergent system properties such as reliability, response time and store occupancy or the selection of language, platform, implementation techniques and tools.

The non-functional requirements can be built on the basis of needs of the user, budget constraints, organization policies and etc.

- Performance requirement: All data entered shall be up to mark and no flaws shall be there for the performance to be 100%.
- Platform constraints: The main target is to generate an intelligent system to predict the adult height.
- Accuracy and Precision: Requirements are accuracy and precision of the data.
- Modifiability: Requirements about the effort required to make changes in the software.
- Often, the measurement is personnel effort (person-months).
- Portability: Since mobile phone is handy so it is portable and can be carried and used whenever required.
- Reliability: Requirements about how often the software fails. The definition of a failure must be clear. Also, don't confuse reliability with availability which is quite a different kind of requirement. Be sure to specify the consequences of software failure, how to protect from failure, a strategy for error Prediction, and a strategy for correction.
- Security: One or more requirements about protection of your system and its data.
- Usability: Requirements about how difficult it will be to learn and operate the system.

The requirements are often expressed in learning time or similar metrics.

Accessibility

Accessibility is a general term used to describe the degree to which a product, device, service, or environment is accessible by as many people as possible. In our project people who have registered with the cloud can access the cloud to store and retrieve their data with the help of a secret key sent to their email ids. User inter face is simple and efficient and easy to use.

Maintainability

In software engineering, maintainability is the ease with which a software product can be modified in order to include new functionalities can be added in the project based on the user requirements just by adding the appropriate files to existing project using. Net and programming languages. Since the programming is very simple, it is easier to find and correct the defects and to make the changes in the project.

Scalability

System is capable of handling increase total through put under an increased load when resources (typically hardware) are added. System can work normally under situations such as low bandwidth and large number of users.

Probability

Portability is one of the key concepts of high-level programming. Portability is the software code base feature to be able to reuse the existing code instead of creating new code when moving software from an environment to another. Project can be executed under different operation conditions provided it meet its minimum configurations. Only system files and dependent assemblies would have to be configured in such case.

To be used efficiently, all computer software needs certain hardware components or other software resources. These pre-requisites are known as (computer) system requirements and are often used as a guideline as opposed to an absolute rule. Most software defines two sets of system requirements: minimum and recommended. With increasing demand for higher processing power and resources in newer versions of software, system requirements tend to increase over time.

5.3. Hardware Requirements

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware, A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements.

- Laptop/Desktop
- Processor: Intel CoreTM i5 and above
- RAM: 4 GB

5.4. Software Requirements

Software Requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or pre-requisites are generally not included in the software installation package and need to be installed separately before the software is installed.

- Operating System: Windows 10
- Programming Language: Python
- IDE: Anaconda Jupyter

5.5 Python Programming Language

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. Developed in the late 1980s by Guido van Rossum, Python emphasizes code readability and expressiveness, making it ideal for beginners and experienced programmers alike. Python's syntax is clean and easy to understand, with a minimalistic approach that focuses on simplicity and clarity. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming, allowing developers to choose the most suitable approach for their projects. Python boasts a vast ecosystem of libraries and frameworks for various domains, including web development (e.g., Django, Flask), data science (e.g., NumPy, pandas), machine learning (e.g., TensorFlow, scikit-learn), and scientific computing (e.g., SciPy). With its rich set of features, extensive documentation, and active community, Python has become one of the most popular programming languages worldwide, powering applications ranging from web development and automation to scientific research and artificial intelligence.

6. Python Packages

To implement the neural network model and perform data pre-processing techniques we need certain python libraries and these libraries we use throughout project every library has a specific task to perform like Data analysis, Data virtualization and some6 mathematical operations and here some of the most frequently used python libraries in the implementation.

6.1 NumPy

Fig 6.1: Numpy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

Statement to Download NumPy: pip install numpy

Statement to import NumPy: import numpy

6.2 Pandas

Pandas is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the NumPy library. Pandas is fast and it has high performance & productivity for users.

Fig 6.2: Pandas

Statement to Download Pandas: pip install pandas

Statement to import Pandas: import pandas

6.3 Matplotlib

Fig 6.3: Matplotlib

Matplotlib is an amazing visualization library in python for 2D plot of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of the data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Statement to Download Matplotlib: pip install matplotlib

Statement to import Matplotlib: import matplotlib

6.4 Scikit-Learn (Sklearn)

Scikit-Learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

Statement to Download Scikit-Learn: pip install sklearn

Statement to import Scikit-Learn: import sklearn

6.4.1 GridSearchCV

GridSearchCV is a scikit-learn library function that performs an exhaustive search over a specified parameter grid for an estimator (model). It is commonly used to find the optimal combination of hyperparameters for a machine learning model.

Statement to import from Scikit-Learn: import GridSearchCV

- You use Support Vector Machine (SVM) for classification.
  GridSearchCV is employed to find the best hyperparameters for the SVM model (specifically, 'C', 'gamma', and 'kernel' parameters).
  The model is trained on the training data.

7. MATERIAL AND METHODS

Several stages are needed to implement machine learning algorithms, gathering datasets is the first, followed by performance analysis and mappings for visualisations. began by gathering datasets and dividing them into distinct ratios for training, testing, and validation, in that order.

7.1. Dataset Description

In the corn fields, phone cameras captured the 4188 images that comprised the four different classes present in the corn leaf disease dataset. The information in the table below shows the classes that make up the dataset, and the figure below shows them.

7.1.1 Blight

The disease is most prevalent during moderate temperatures (64 to 80 degrees F) with prolonged periods of moisture. It typically appears at or after silking, but the disease is usually more severe when infection occurs earlier.

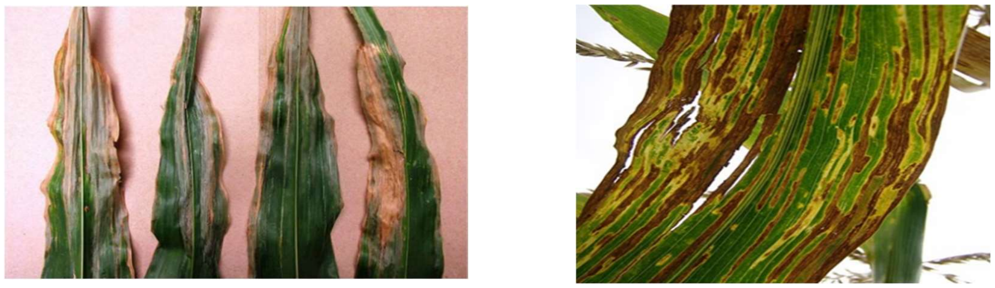


Fig 7.1: Blight

7.1.2 Common rust

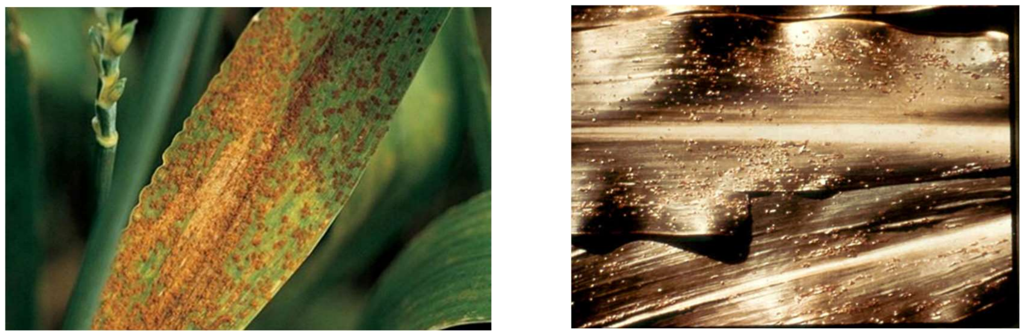


Fig 7.2: Common rust

The common rust fungus overwinters in the southern U.S. and Mexico. Urediniospores are blown north to the Midwestern Corn Belt in the summer and infection occurs in June or July. Young leaves are most susceptible to infection and pustules are more likely to form after corn silking. The disease Favors cool temperatures (60 - 76 degrees F), heavy dews, about six hours of leaf wetness, and relative humidity greater than 95 percent. Temperatures above 80 degrees F suppress disease development and spread.

7.1.3 Grey leaf spot

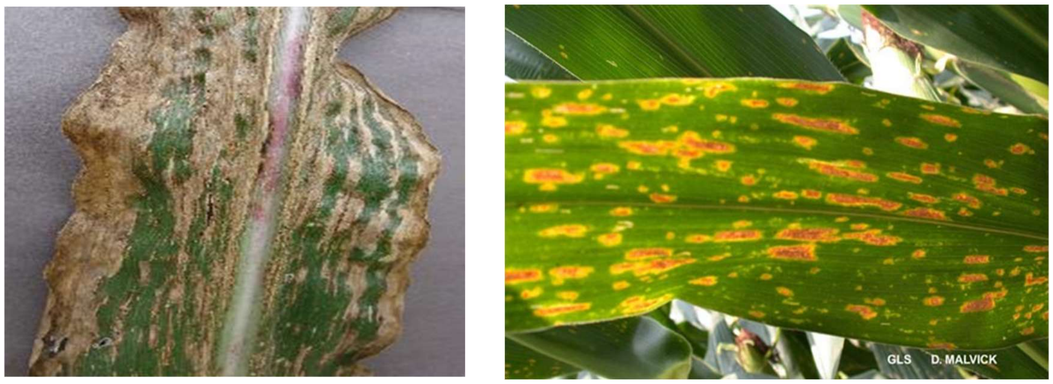


Fig 7.3: Grey leaf spot

Conditions that Favor infection include moderate to warm temperatures during extended periods (greater than 24 hr) of high humidity (greater than 95 percent) or wet weather. Gray leaf spot is a problem in areas with minimum tillage and corn on corn rotations. In susceptible hybrids, the disease typically develops from silking to maturity.

7.1.4 Healthy

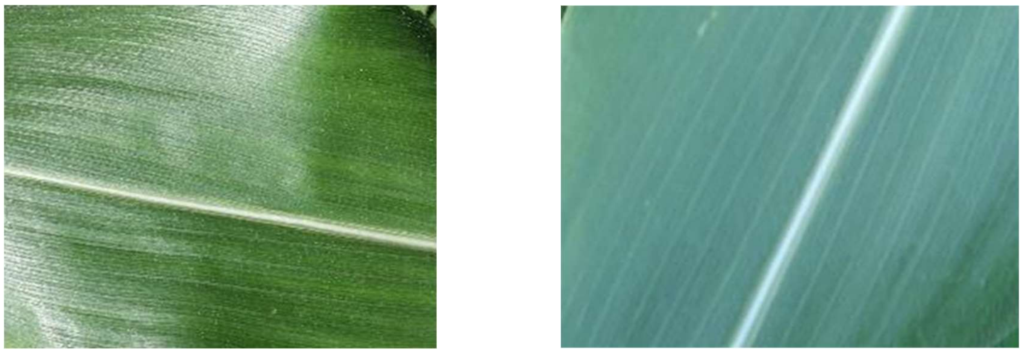


Fig 7.4: Healthy

When corn plant growth is severely stunted – leaves are thinner than normal, wrinkled and weak, it indicates a boron or manganese deficiency.

7.2 Proposed model

7.2.1 Dataset Collection

Gather a diverse dataset of plant images, including healthy and diseased samples for various plant species. Annotate the images to create a labelled dataset for training.

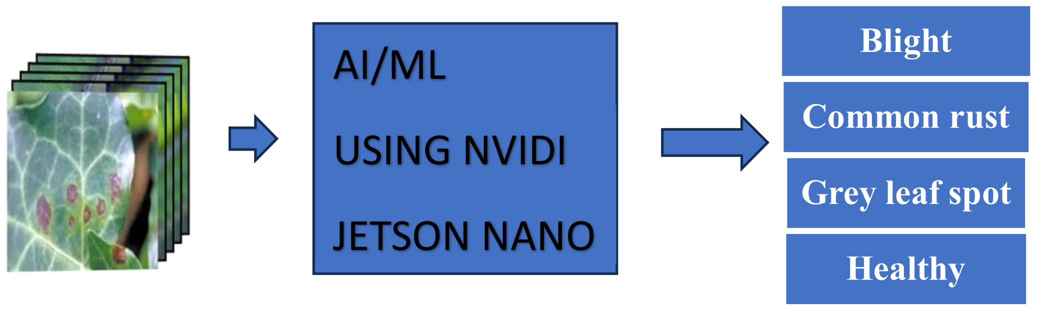


Fig 7.5: Dataset Collection

7.2.2 Data Preprocessing:

Resize and normalize the images to ensure consistency in the input data. Augment the dataset with techniques like rotation and flipping to enhance model generalization.

7.2.3 Inference and Output:

Run the trained model on Jetson Nano to perform real-time inference on the captured plant images. Display the results, indicating whether the plant is healthy or affected by a specific disease.

7.3 ML Algorithms

7.3.1 Support Vector Machine Algorithm (SVM):

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

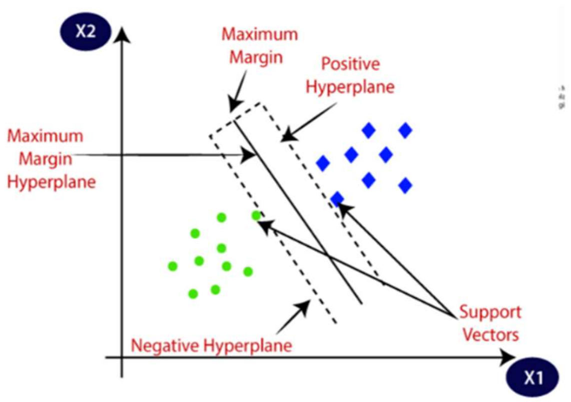


Fig 7.6: SVM

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane

7.3.2 Random Forest Algorithm

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

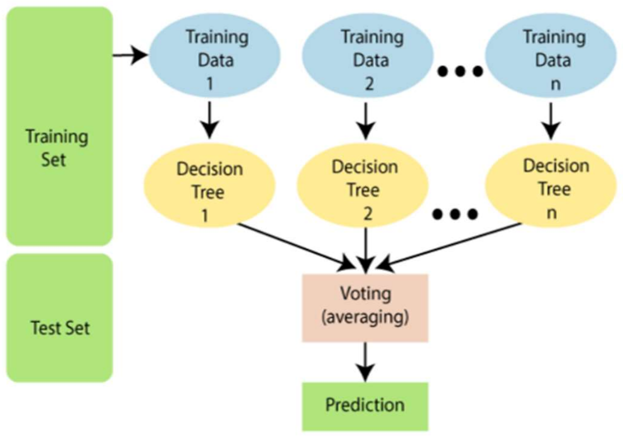


Fig 7.7: Random Forest Algorithm

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

7.3.3 Decision Tree Classification Algorithm

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules, and each leaf node represents the outcome.

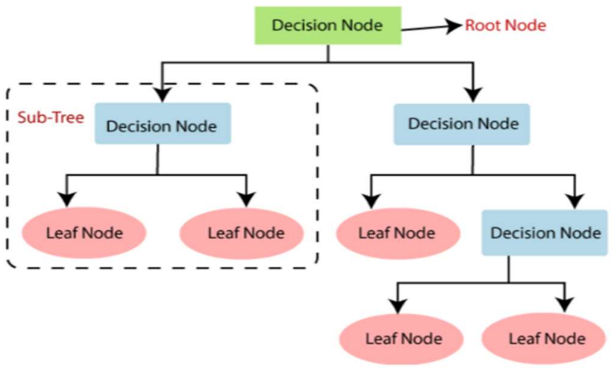


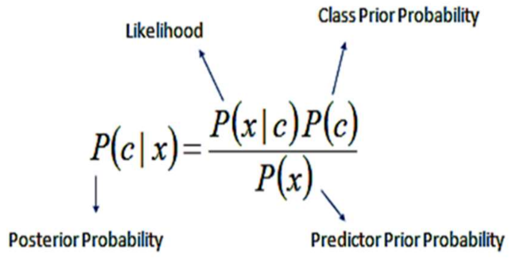
Fig 7.8: Decision Tree Classification Algorithm

In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

The decisions or the test are performed on the basis of features of the given dataset.

7.3.4 Naive Bayes Classifier Algorithm

The Naive Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.


Likelihood
Class Prior Probability
$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$
Posterior Probability
Predictor Prior Probability

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \cdots \times P(x_n|c) \times P(c)$$

It is mainly used in text classification that includes a high-dimensional training dataset. Naive Bayes Classifier is one of the simplest and most effective Classification algorithms which helps in building fast machine learning models that can make quick predictions.

It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

7.3.5 K-Nearest Neighbor (KNN) Algorithm for Machine Learning

K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

The K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

The K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using the K- NN algorithm.

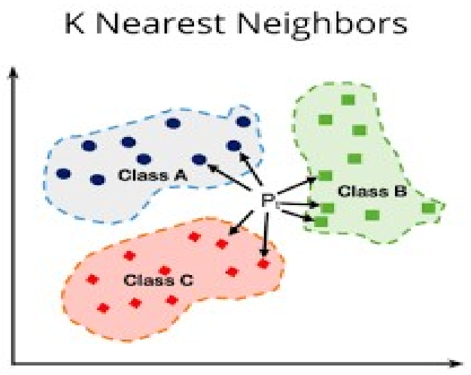


Fig 7.9: K Nearest Neighbors

The K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for Classification problems.

It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

7.3.6 Logistic Regression in Machine Learning

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.

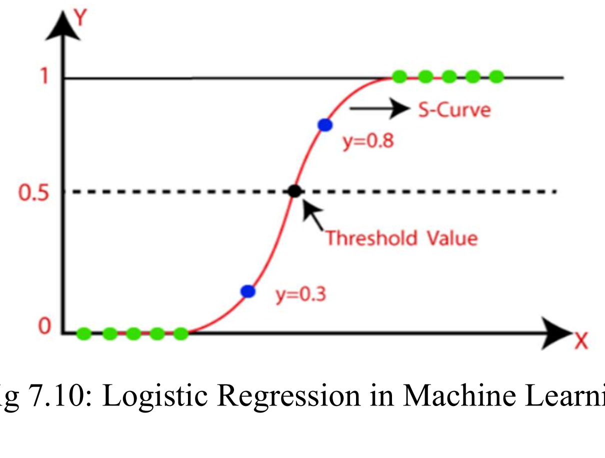


Fig 7.10: Logistic Regression in Machine Learning

Logistic regression predicts the output of a categorical dependent variable. Therefore, the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.

7.3.7Adaptive boosting or AdaBoost

This method operates iteratively, identifying misclassified data points and adjusting their weights to minimize the training error. The model continues to optimize sequentially until it yields the strongest predictor.

The results from the first decision stump are analyzed, and if any observations are wrongfully classified, they are assigned higher weights. A new decision stump is drawn by considering the higher-weight observations as more significant. Again, if any observations are misclassified, they're given a higher weight, and this process continues until all the observations fall into the right class.

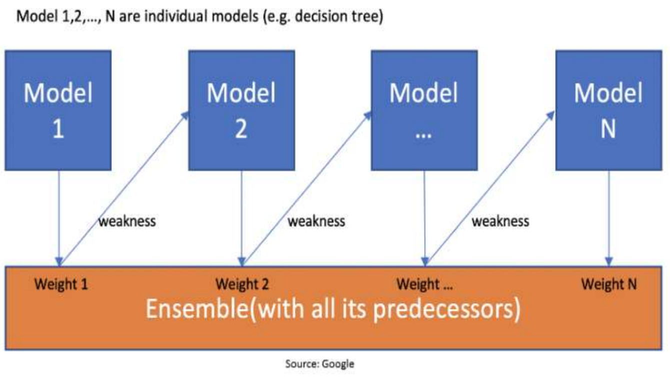


Fig 7.11: Ada Boost

AdaBoost can be used for both classification and regression-based problems. However, it is more commonly used for classification purposes.

7.4 Evaluation Metrics

Evaluation metrics are the scoring system for machine learning models, providing quantitative measures of a model's ability to generalize and make accurate predictions on unseen data. These metrics are essential for comparing models, identifying weaknesses for improvement, and tracking a model's performance as it evolves. With a variety of metrics available, choosing the right one depends on the specific problem and data characteristics.

7.4.1. Accuracy

The simplest metric to determine the overall robustness of the model is accuracy. It calculates the proportion of accurately predicted cases to all instances.

$TrueNegatives + Truepositives$

$Accuracy = \quad TruePositive + FalseNegative + FalseNegative$

7.4.2. Precision

Precision focuses on the right cases among the anticipated positive cases. It is defined as the ratio of all positive predictions to correctly predicted positive observations. When assessing the model's potential for false positives, accuracy plays a role.

$TruePositive$

$Precision = \quad TruePositive + FalsePositive$

7.4.3. Recall

The percentage of true positives (actual positives) that a model correctly identified is calculated using recall. Recall is best used when the cost of a false negative is significant.

$TruePositive$

$Recall = \quad TruePositive + FalseNegative$

7.4.4. F1-Score

The precision and recall harmonic mean are represented by this. In terms of math, it looks like this:

$2* \, (Precision * Recall)$

$F1 - Score =$

$Precision + Recall$

6. Code

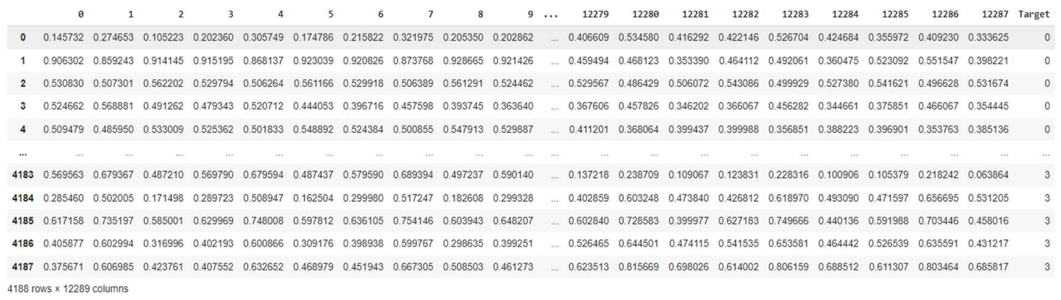
```
#Uploading Corn_maize.zip to drive and connect it to collab

from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

#Creating a Folder Corn in /content and extract the zip file to that folder

!pip install pyunpack
!pip install patool
from pyunpack import Archive
Archive('/content/drive/MyDrive/Corn_Maize.zip').extractall('/content/Corn')

#importing required packages

import pandas as pd
from sklearn import svm
from sklearn.model_selection import GridSearchCV
import os
import matplotlib.pyplot as plt
from skimage.transform import resize
from skimage.io import imread
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,accuracy_score,confusion_matrix
import pickle

#creating categories of Corn leaf types

Categories=['Blight','Common_Rust','Gray_Leaf_Spot','Healthy']
print("Type y to give categories or type n to go with classification of Other");

while(True):
  check=input()
  if(check=='n' or check=='y'):
    break
  print("Please give a valid input (y/n)")
if(check=='y'):
  print("Enter How Many types of Images do you want to classify")
  n=int(input())
  Categories=[]
  print(f'please enter {n} names')
  for i in range(n):
    name=input()
    Categories.append(name)
  print(f"If not drive Please upload all the {n} category images in google collab with the same names as given in categories")
```

```
Type y to give categories or type n to go with classification of Other
y
Enter How Many types of Images do you want to classify
4
please enter 4 names
Blight
Common_Rust
Gray_Leaf_Spot
Healthy
If not drive Please upload all the 4 category images in google collab with the same names as given in categories
```

```
#resizing the dataset

flat_data_arr=[]

target_arr=[]
#please use datadir='/content' if the files are upload on to google collab
#else mount the drive and give path of the parent-folder containing all category images folders.
datadir='/content/Corn/data'
for i in Categories:
  print(f'loading... category : {i}')
  path=os.path.join(datadir,i)
  for img in os.listdir(path):
    img_array=imread(os.path.join(path,img))
    img_resized=resize(img_array,(64,64,3))
    flat_data_arr.append(img_resized.flatten())
    target_arr.append(Categories.index(i))
  print(f'loaded category:{i} successfully')
flat_data=np.array(flat_data_arr)
target=np.array(target_arr)
df=pd.DataFrame(flat_data)
df['Target']=target
df
```

```
loading... category : Blight
loaded category:Blight successfully
loading... category : Common_Rust
loaded category:Common_Rust successfully
loading... category : Gray_Leaf_Spot
loaded category:Gray_Leaf_Spot successfully
loading... category : Healthy
```

```
loaded category:Healthy successfully
```



	0	1	2	3	4	5	6	7	8	9	...	12279	12280	12281	12282	12283	12284	12285	12286	12287	Target
0	0.145732	0.274653	0.105223	0.202360	0.305749	0.174786	0.215822	0.321975	0.205350	0.202862	...	0.406609	0.534580	0.416292	0.422146	0.526704	0.424684	0.355972	0.409230	0.333625	0
1	0.906302	0.859243	0.914145	0.915195	0.868137	0.923039	0.920826	0.873768	0.928665	0.921426	...	0.459494	0.468123	0.353390	0.464112	0.492061	0.360475	0.523092	0.551547	0.398221	0
2	0.530830	0.507301	0.562202	0.529794	0.506264	0.561166	0.529918	0.506389	0.561291	0.524462	...	0.529567	0.486429	0.506072	0.543086	0.499929	0.527380	0.541621	0.496628	0.531674	0
3	0.524662	0.568881	0.491262	0.479343	0.520712	0.444053	0.396716	0.457598	0.393745	0.363640	...	0.367606	0.457826	0.346202	0.366067	0.456282	0.344661	0.375851	0.466067	0.354445	0
4	0.509479	0.485950	0.533009	0.525362	0.501833	0.548892	0.524384	0.500855	0.547913	0.529887	...	0.411201	0.368064	0.399437	0.399988	0.356851	0.388223	0.396901	0.353763	0.385136	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4183	0.569563	0.679367	0.487210	0.569790	0.679594	0.487437	0.579590	0.689394	0.497237	0.590140	...	0.137218	0.238709	0.109067	0.123831	0.228316	0.100906	0.105379	0.218242	0.063864	3
4184	0.285460	0.502005	0.171498	0.289723	0.508947	0.162504	0.299980	0.517247	0.182608	0.299328	...	0.402859	0.603248	0.473840	0.426812	0.618970	0.493090	0.471597	0.656695	0.531205	3
4185	0.617158	0.735197	0.585001	0.629969	0.748008	0.597812	0.636105	0.754146	0.603943	0.648207	...	0.602840	0.728583	0.399977	0.627183	0.749666	0.440136	0.591988	0.703446	0.458016	3
4186	0.405877	0.602994	0.316996	0.402193	0.600866	0.309176	0.398938	0.599767	0.298635	0.399251	...	0.526465	0.644501	0.474115	0.541535	0.653581	0.464442	0.526539	0.635591	0.431217	3
4187	0.375671	0.606985	0.423761	0.407552	0.632652	0.468979	0.451943	0.667305	0.508503	0.461273	...	0.623513	0.815669	0.698026	0.614002	0.806159	0.688512	0.611307	0.803464	0.685817	3

4188 rows × 12289 columns

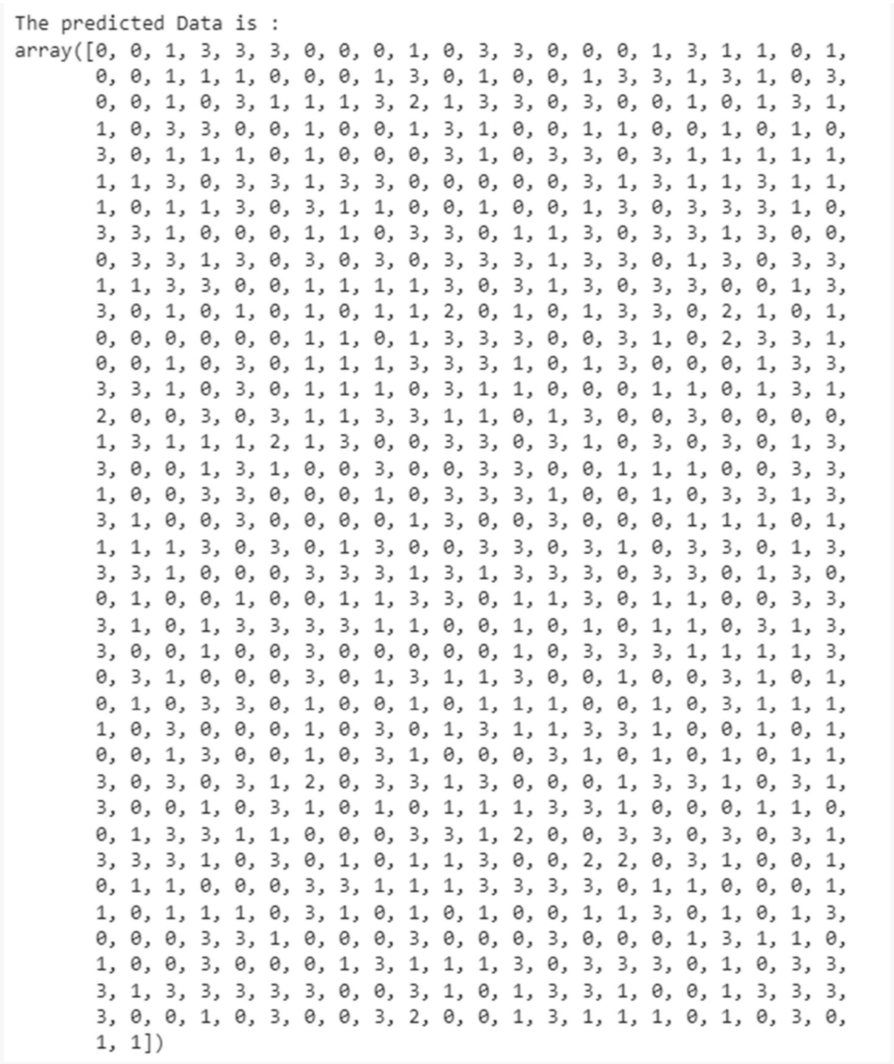
```
#Splitting

x=df.iloc[:,:-1]
y=df.iloc[:,-1]
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=77,stratif
y=y)
print('Splitted Successfully')
```

```
Splitted Successfully
```

```
#Training of the Model

param_grid={'C':[0.1,1],'gamma':[0.001,0.1],'kernel':['rbf']}
svc=svm.SVC(probability=True)
print("The training of the model is started, please wait for while as it may take few
minutes to complete")
model=GridSearchCV(svc,param_grid)
model.fit(x_train,y_train)
print('The Model is trained well with the given images')
model.best_params_
```

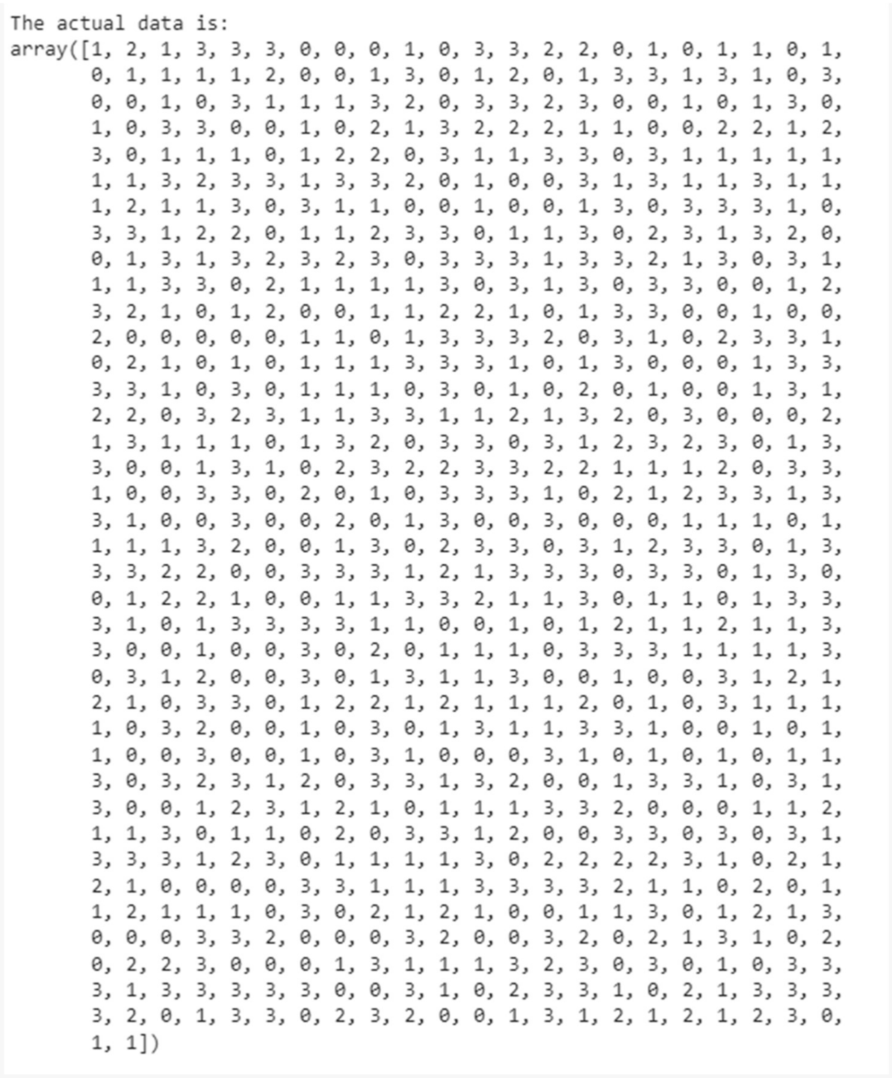
```
The training of the model is started, please wait for while as it may take few minutes to
complete
The Model is trained well with the given images
{'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
```

```
# Predicted data

y_pred=model.predict(x_test)
print("The predicted Data is :")
y_pred
```



```
The predicted Data is :
array([0, 0, 1, 3, 3, 3, 0, 0, 0, 1, 0, 3, 3, 0, 0, 0, 1, 3, 1, 1, 0, 1,
       0, 0, 1, 1, 1, 0, 0, 0, 1, 3, 0, 1, 0, 0, 1, 3, 3, 1, 3, 1, 0, 3,
       0, 0, 1, 0, 3, 1, 1, 1, 3, 2, 1, 3, 3, 0, 3, 0, 0, 1, 0, 1, 3, 1,
       1, 0, 3, 3, 0, 0, 1, 0, 0, 1, 3, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0,
       3, 0, 1, 1, 1, 0, 1, 0, 0, 0, 3, 1, 0, 3, 3, 0, 3, 1, 1, 1, 1, 1,
       1, 1, 3, 0, 3, 3, 1, 3, 3, 0, 0, 0, 0, 0, 3, 1, 3, 1, 1, 3, 1, 1,
       1, 0, 1, 1, 3, 0, 3, 1, 1, 0, 0, 1, 0, 0, 1, 3, 0, 3, 3, 3, 1, 0,
       3, 3, 1, 0, 0, 0, 1, 1, 0, 3, 3, 0, 1, 1, 3, 0, 3, 3, 1, 3, 0, 0,
       0, 3, 3, 1, 3, 0, 3, 0, 3, 0, 3, 3, 3, 1, 3, 3, 0, 1, 3, 0, 3, 3,
       1, 1, 3, 3, 0, 0, 1, 1, 1, 1, 3, 0, 3, 1, 3, 0, 3, 3, 0, 0, 1, 3,
       3, 0, 1, 0, 1, 0, 1, 0, 1, 1, 2, 0, 1, 0, 1, 3, 3, 0, 2, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 3, 3, 3, 0, 0, 3, 1, 0, 2, 3, 3, 1,
       0, 0, 1, 0, 3, 0, 1, 1, 1, 3, 3, 3, 1, 0, 1, 3, 0, 0, 0, 1, 3, 3,
       3, 3, 1, 0, 3, 0, 1, 1, 1, 0, 3, 1, 1, 0, 0, 0, 1, 1, 0, 1, 3, 1,
       2, 0, 0, 3, 0, 3, 1, 1, 3, 3, 1, 1, 0, 1, 3, 0, 0, 3, 0, 0, 0, 0,
       1, 3, 1, 1, 1, 2, 1, 3, 0, 0, 3, 3, 0, 3, 1, 0, 3, 0, 3, 0, 1, 3,
       3, 0, 0, 1, 3, 1, 0, 0, 3, 0, 0, 3, 3, 0, 0, 1, 1, 1, 0, 0, 3, 3,
       1, 0, 0, 3, 3, 0, 0, 0, 1, 0, 3, 3, 3, 1, 0, 0, 1, 0, 3, 3, 1, 3,
       3, 1, 0, 0, 3, 0, 0, 0, 0, 1, 3, 0, 0, 3, 0, 0, 0, 1, 1, 1, 0, 1,
       1, 1, 1, 3, 0, 3, 0, 1, 3, 0, 0, 3, 3, 0, 3, 1, 0, 3, 3, 0, 1, 3,
       3, 3, 1, 0, 0, 0, 3, 3, 3, 1, 3, 1, 3, 3, 3, 0, 3, 3, 0, 1, 3, 0,
       0, 1, 0, 0, 1, 0, 0, 1, 1, 3, 3, 0, 1, 1, 3, 0, 1, 1, 0, 0, 3, 3,
       3, 1, 0, 1, 3, 3, 3, 3, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 3, 1, 3,
       3, 0, 0, 1, 0, 0, 3, 0, 0, 0, 0, 0, 1, 0, 3, 3, 3, 1, 1, 1, 1, 3,
       0, 3, 1, 0, 0, 0, 3, 0, 1, 3, 1, 1, 3, 0, 0, 1, 0, 0, 3, 1, 0, 1,
       0, 1, 0, 3, 3, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 3, 1, 1, 1,
       1, 0, 3, 0, 0, 0, 1, 0, 3, 0, 1, 3, 1, 1, 3, 3, 1, 0, 0, 1, 0, 1,
       0, 0, 1, 3, 0, 0, 1, 0, 3, 1, 0, 0, 0, 3, 1, 0, 1, 0, 1, 0, 1, 1,
       3, 0, 3, 0, 3, 1, 2, 0, 3, 3, 1, 3, 0, 0, 0, 1, 3, 3, 1, 0, 3, 1,
       3, 0, 0, 1, 0, 3, 1, 0, 1, 0, 1, 1, 1, 3, 3, 1, 0, 0, 0, 1, 1, 0,
       0, 1, 3, 3, 1, 1, 0, 0, 0, 3, 3, 1, 2, 0, 0, 3, 3, 0, 3, 0, 3, 1,
       3, 3, 3, 1, 0, 3, 0, 1, 0, 1, 1, 3, 0, 0, 2, 2, 0, 3, 1, 0, 0, 1,
       0, 1, 1, 0, 0, 0, 3, 3, 1, 1, 1, 3, 3, 3, 3, 0, 1, 1, 0, 0, 0, 1,
       1, 0, 1, 1, 1, 0, 3, 1, 0, 1, 0, 1, 0, 0, 1, 1, 3, 0, 1, 0, 1, 3,
       0, 0, 0, 3, 3, 1, 0, 0, 0, 3, 0, 0, 0, 3, 0, 0, 0, 1, 3, 1, 1, 0,
       1, 0, 0, 3, 0, 0, 0, 1, 3, 1, 1, 1, 3, 0, 3, 3, 3, 0, 1, 0, 3, 3,
       3, 1, 3, 3, 3, 3, 3, 0, 0, 3, 1, 0, 1, 3, 3, 1, 0, 0, 1, 3, 3, 3,
       3, 0, 0, 1, 0, 3, 0, 0, 3, 2, 0, 0, 1, 3, 1, 1, 1, 0, 1, 0, 3, 0,
       1, 1])
```

```
#Actual data

print("The actual data is:")
np.array(y_test)
```



```
The actual data is:
array([1, 2, 1, 3, 3, 3, 0, 0, 0, 1, 0, 3, 3, 2, 2, 0, 1, 0, 1, 1, 0, 1,
       0, 1, 1, 1, 1, 2, 0, 0, 1, 3, 0, 1, 2, 0, 1, 3, 3, 1, 3, 1, 0, 3,
       0, 0, 1, 0, 3, 1, 1, 1, 3, 2, 0, 3, 3, 2, 3, 0, 0, 1, 0, 1, 3, 0,
       1, 0, 3, 3, 0, 0, 1, 0, 2, 1, 3, 2, 2, 2, 1, 1, 0, 0, 2, 2, 1, 2,
       3, 0, 1, 1, 1, 0, 1, 2, 2, 0, 3, 1, 1, 3, 3, 0, 3, 1, 1, 1, 1, 1,
       1, 1, 3, 2, 3, 3, 1, 3, 3, 2, 0, 1, 0, 0, 3, 1, 3, 1, 1, 3, 1, 1,
       1, 2, 1, 1, 3, 0, 3, 1, 1, 0, 0, 1, 0, 0, 1, 3, 0, 3, 3, 3, 1, 0,
       3, 3, 1, 2, 2, 0, 1, 1, 2, 3, 3, 0, 1, 1, 3, 0, 2, 3, 1, 3, 2, 0,
       0, 1, 3, 1, 3, 2, 3, 2, 3, 0, 3, 3, 3, 1, 3, 3, 2, 1, 3, 0, 3, 1,
       1, 1, 3, 3, 0, 2, 1, 1, 1, 1, 3, 0, 3, 1, 3, 0, 3, 3, 0, 0, 1, 2,
       3, 2, 1, 0, 1, 2, 0, 0, 1, 1, 2, 2, 1, 0, 1, 3, 3, 0, 0, 1, 0, 0,
       2, 0, 0, 0, 0, 0, 1, 1, 0, 1, 3, 3, 3, 2, 0, 3, 1, 0, 2, 3, 3, 1,
       0, 2, 1, 0, 1, 0, 1, 1, 1, 3, 3, 3, 1, 0, 1, 3, 0, 0, 0, 1, 3, 3,
       3, 3, 1, 0, 3, 0, 1, 1, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0, 0, 1, 3, 1,
       2, 2, 0, 3, 2, 3, 1, 1, 3, 3, 1, 1, 2, 1, 3, 2, 0, 3, 0, 0, 0, 2,
       1, 3, 1, 1, 1, 0, 1, 3, 2, 0, 3, 3, 0, 3, 1, 2, 3, 2, 3, 0, 1, 3,
       3, 0, 0, 1, 3, 1, 0, 2, 3, 2, 2, 3, 3, 2, 2, 1, 1, 1, 2, 0, 3, 3,
       1, 0, 0, 3, 3, 0, 2, 0, 1, 0, 3, 3, 3, 1, 0, 2, 1, 2, 3, 3, 1, 3,
       3, 1, 0, 0, 3, 0, 0, 2, 0, 1, 3, 0, 0, 3, 0, 0, 0, 1, 1, 1, 0, 1,
       1, 1, 1, 3, 2, 0, 0, 1, 3, 0, 2, 3, 3, 0, 3, 1, 2, 3, 3, 0, 1, 3,
       3, 3, 2, 2, 0, 0, 3, 3, 3, 1, 2, 1, 3, 3, 3, 0, 3, 3, 0, 1, 3, 0,
       0, 1, 2, 2, 1, 0, 0, 1, 1, 3, 3, 2, 1, 1, 3, 0, 1, 1, 0, 1, 3, 3,
       3, 1, 0, 1, 3, 3, 3, 3, 1, 1, 0, 0, 1, 0, 1, 2, 1, 1, 2, 1, 1, 3,
       3, 0, 0, 1, 0, 0, 3, 0, 2, 0, 1, 1, 1, 0, 3, 3, 3, 1, 1, 1, 1, 3,
       0, 3, 1, 2, 0, 0, 3, 0, 1, 3, 1, 1, 3, 0, 0, 1, 0, 0, 3, 1, 2, 1,
       2, 1, 0, 3, 3, 0, 1, 2, 2, 1, 2, 1, 1, 1, 2, 0, 1, 0, 3, 1, 1, 1,
       1, 0, 3, 2, 0, 0, 1, 0, 3, 0, 1, 3, 1, 1, 3, 3, 1, 0, 0, 1, 0, 1,
       1, 0, 0, 3, 0, 0, 1, 0, 3, 1, 0, 0, 0, 3, 1, 0, 1, 0, 1, 0, 1, 1,
       3, 0, 3, 2, 3, 1, 2, 0, 3, 3, 1, 3, 2, 0, 0, 1, 3, 3, 1, 0, 3, 1,
       3, 0, 0, 1, 2, 3, 1, 2, 1, 0, 1, 1, 1, 3, 3, 2, 0, 0, 0, 1, 1, 2,
       1, 1, 3, 0, 1, 1, 0, 2, 0, 3, 3, 1, 2, 0, 0, 3, 3, 0, 3, 0, 3, 1,
       3, 3, 3, 1, 2, 3, 0, 1, 1, 1, 1, 3, 0, 2, 2, 2, 2, 3, 1, 0, 2, 1,
       2, 1, 0, 0, 0, 0, 3, 3, 1, 1, 1, 3, 3, 3, 3, 2, 1, 1, 0, 2, 0, 1,
       1, 2, 1, 1, 1, 0, 3, 0, 2, 1, 2, 1, 0, 0, 1, 1, 3, 0, 1, 2, 1, 3,
       0, 0, 0, 3, 3, 2, 0, 0, 0, 3, 2, 0, 0, 3, 2, 0, 2, 1, 3, 1, 0, 2,
       0, 2, 2, 3, 0, 0, 0, 1, 3, 1, 1, 1, 3, 2, 3, 0, 3, 0, 1, 0, 3, 3,
       3, 1, 3, 3, 3, 3, 3, 0, 0, 3, 1, 0, 2, 3, 3, 1, 0, 2, 1, 3, 3, 3,
       3, 2, 0, 1, 3, 3, 0, 2, 3, 2, 0, 0, 1, 3, 1, 2, 1, 2, 1, 2, 3, 0,
       1, 1])
```

```
#Classification Report and Confusion Matrix

print('Classification Report:')
print(classification_report(y_pred,y_test))
print(f"The model is {accuracy_score(y_pred,y_test)*100}% accurate")
print('Confusion_matrix:')
print(confusion_matrix(y_pred,y_test))
```

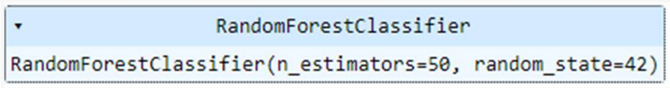
```
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.66      0.77       319
           1       0.95      0.93      0.94       265
           2       0.08      0.82      0.14        11
           3       1.00      0.95      0.97       243

    accuracy                           0.84       838
```

```
   macro avg       0.74      0.84      0.71       838
weighted avg       0.94      0.84      0.88       838

The model is 83.53221957040573% accurate
```

```
import sklearn.metrics as metrics
print(metrics.classification_report(y_pred,y_test, digits=4))
```

```
              precision    recall  f1-score   support

           0     0.9258    0.6646    0.7737       319
           1     0.9464    0.9321    0.9392       265
           2     0.0783    0.8182    0.1429        11
           3     0.9957    0.9547    0.9748       243

    accuracy                         0.8353       838
   macro avg     0.7365    0.8424    0.7076       838
weighted avg     0.9414    0.8353    0.8761       838
```

```
#RANDOM FOREST

from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators = 50, random_state = 42)
model.fit(x_train,y_train)
```

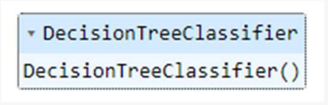


```
RandomForestClassifier
RandomForestClassifier(n_estimators=50, random_state=42)
```

```
y_pred=model.predict(x_test)
print("The predicted Data is :")
print(y_pred)
print("The actual data is:")
print(np.array(y_test))
print('Classification report:')
print(classification_report(y_pred,y_test))
print(f"The model is {accuracy_score(y_pred,y_test)*100}% accurate")
print('Confusion_matrix:')
print(confusion_matrix(y_pred,y_test))
import sklearn.metrics as metrics
print(metrics.classification_report(y_pred,y_test, digits=4))
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.88      0.65      0.75       309
           1       0.93      0.98      0.96       249
           2       0.20      0.55      0.29        42
           3       0.92      0.90      0.91       238

    accuracy                           0.81       838
   macro avg       0.73      0.77      0.73       838
weighted avg       0.87      0.81      0.83       838

The model is 81.38424821002387% accurate
              precision    recall  f1-score   support

           0     0.8777    0.6505    0.7472       309
           1     0.9349    0.9799    0.9569       249
           2     0.2000    0.5476    0.2930        42
           3     0.9185    0.8992    0.9087       238

    accuracy                         0.8138       838
   macro avg     0.7328    0.7693    0.7264       838
weighted avg     0.8723    0.8138    0.8326       838
```

```
#DecisionTree

from sklearn.tree import DecisionTreeClassifier
model=DecisionTreeClassifier()
model.fit(x_train,y_train)
```

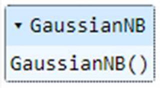


```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
y_pred=model.predict(x_test)
print("The predicted Data is :")
print(y_pred)
print("The actual data is:")
print(np.array(y_test))
print('Classification report:')
print(classification_report(y_pred,y_test))
print(f"The model is {accuracy_score(y_pred,y_test)*100}% accurate")
print('Confusion_matrix:')
print(confusion_matrix(y_pred,y_test))
import sklearn.metrics as metrics
print(metrics.classification_report(y_pred,y_test, digits=4))
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.59      0.66      0.62       205
           1       0.93      0.92      0.92       264
           2       0.40      0.38      0.39       120
           3       0.83      0.78      0.80       249

    accuracy                           0.74       838
   macro avg       0.69      0.68      0.69       838
weighted avg       0.74      0.74      0.74       838

The model is 73.62768496420048% accurate
              precision    recall  f1-score   support

           0     0.5895    0.6585    0.6221       205
           1     0.9272    0.9167    0.9219       264
           2     0.4000    0.3833    0.3915       120
           3     0.8326    0.7791    0.8050       249

    accuracy                         0.7363       838
   macro avg     0.6873    0.6844    0.6851       838
weighted avg     0.7410    0.7363    0.7379       838
```

```
#naive_bayes

from sklearn.naive_bayes import GaussianNB
model=GaussianNB()
model.fit(x_train,y_train)
```



```
▾ GaussianNB
GaussianNB()
```

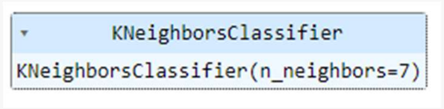
```
y_pred=model.predict(x_test)
print("The predicted Data is :")
print(y_pred)
print("The actual data is:")
print(np.array(y_test))
print('Classification report:')
print(classification_report(y_pred,y_test))
print(f"The model is {accuracy_score(y_pred,y_test)*100}% accurate")
print('Confusion_matrix:')
print(confusion_matrix(y_pred,y_test))
import sklearn.metrics as metrics
print(metrics.classification_report(y_pred,y_test, digits=4))
```

```
Classification report:
              precision    recall  f1-score   support
```

```
           0       0.71      0.63      0.67       257
           1       0.94      0.90      0.92       271
           2       0.43      0.45      0.44       110
           3       0.77      0.90      0.83       200

    accuracy                           0.76       838
   macro avg       0.71      0.72      0.71       838
weighted avg       0.76      0.76      0.76       838

The model is 76.01431980906922% accurate
```

```
              precision    recall  f1-score   support

           0     0.7118    0.6342    0.6708       257
           1     0.9387    0.9041    0.9211       271
           2     0.4261    0.4455    0.4356       110
           3     0.7725    0.9000    0.8314       200

    accuracy                         0.7601       838
   macro avg     0.7123    0.7209    0.7147       838
weighted avg     0.7622    0.7601    0.7592       838
```

```
#knn

from sklearn.neighbors import KNeighborsClassifier
model=KNeighborsClassifier(n_neighbors=7)
model.fit(x_train,y_train)
```



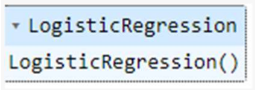
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

```
y_pred=model.predict(x_test)
print("The predicted Data is :")
print(y_pred)
print("The actual data is:")
print(np.array(y_test))
print('Classification report:')
print(classification_report(y_pred,y_test))
print(f"The model is {accuracy_score(y_pred,y_test)*100}% accurate")
print('Confusion_Matrix:')
print(confusion_matrix(y_pred,y_test))
import sklearn.metrics as metrics
print(metrics.classification_report(y_pred,y_test, digits=4))
```

```
Classification report:
              precision    recall  f1-score   support
```

```
           0       0.76      0.67      0.71       258
           1       0.90      0.99      0.94       237
           2       0.33      0.52      0.40        73
           3       0.97      0.84      0.90       270

    accuracy                           0.80       838
   macro avg       0.74      0.75      0.74       838
weighted avg       0.83      0.80      0.81       838

The model is 80.1909307875895% accurate

              precision    recall  f1-score   support

           0     0.7555    0.6705    0.7105       258
           1     0.8966    0.9873    0.9398       237
           2     0.3304    0.5205    0.4043        73
           3     0.9742    0.8407    0.9026       270

    accuracy                         0.8019       838
   macro avg     0.7392    0.7548    0.7393       838
weighted avg     0.8288    0.8019    0.8105       838
```

```
#Logistic Regression

from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(x_train,y_train)
```



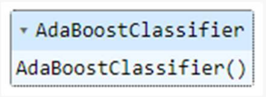
```
▾ LogisticRegression
LogisticRegression()
```

```
y_pred=model.predict(x_test)
print("The predicted Data is :")
print(y_pred)
print("The actual data is:")
print(np.array(y_test))
print('Classification report:')
print(classification_report(y_pred,y_test))
print(f"The model is {accuracy_score(y_pred,y_test)*100}% accurate")
print('Confusion_Matrix:')
print(confusion_matrix(y_pred,y_test))
import sklearn.metrics as metrics
print(metrics.classification_report(y_pred,y_test, digits=4))
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.76      0.75      0.75       233
           1       0.94      0.91      0.92       270
           2       0.45      0.62      0.52        84
           3       0.97      0.90      0.94       251

    accuracy                           0.83       838
   macro avg       0.78      0.79      0.78       838
weighted avg       0.85      0.83      0.84       838

The model is 83.29355608591885% accurate
```

```
              precision    recall  f1-score   support

           0     0.7598    0.7468    0.7532       233
           1     0.9387    0.9074    0.9228       270
           2     0.4522    0.6190    0.5226        84
           3     0.9742    0.9044    0.9380       251

    accuracy                         0.8329       838
   macro avg     0.7812    0.7944    0.7842       838
weighted avg     0.8508    0.8329    0.8401       838
```

```
#Ada Booster

from sklearn.ensemble import AdaBoostClassifier
model=AdaBoostClassifier()
model.fit(x_train,y_train)
```



```
▾ AdaBoostClassifier
AdaBoostClassifier()
```

```
y_pred=model.predict(x_test)
print("The predicted Data is :")
print(y_pred)
print("The actual data is:")
print(np.array(y_test))
print('Classification report:')
print(classification_report(y_pred,y_test))
print(f"The model is {accuracy_score(y_pred,y_test)*100}% accurate")
print('Confusion_Matrix:')
print(confusion_matrix(y_pred,y_test))
import sklearn.metrics as metrics
print(metrics.classification_report(y_pred,y_test, digits=4))
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.84      0.59      0.70       323
           1       0.92      0.97      0.95       248
           2       0.23      0.51      0.31        51
           3       0.82      0.88      0.85       216

    accuracy                           0.77       838
   macro avg       0.70      0.74      0.70       838
weighted avg       0.82      0.77      0.79       838

The model is 77.44630071599046% accurate

              precision    recall  f1-score   support

           0     0.8384    0.5944    0.6957       323
           1     0.9234    0.9718    0.9470       248
           2     0.2261    0.5098    0.3133        51
           3     0.8155    0.8796    0.8463       216

    accuracy                         0.7745       838
   macro avg     0.7008    0.7389    0.7005       838
weighted avg     0.8204    0.7745    0.7856       838
```

8.1 RESULTS AND DISCUSSION

- **Support Vector Machine Algorithm (SVM):**

Confusion matrix

```
[[212  10  96   1]
 [ 11 247   7   0]
 [  2   0   9   0]
 [  4   4   3 232]]
```

Classification Report

CLASSIFIER	PRECISION	RECALL	F1SCORE
Blight	0.93	0.66	0.77
Common rust	0.95	0.93	0.94

Gray leaf spot	0.08	0.82	0.14
Healthy	1.00	0.95	0.97

- **Random Forest Algorithm**

  Confusion matrix

```
[[201   9  83  16]
 [  4 244   1   0]
 [ 14   2  23   3]
 [ 10   6   8 214]]
```

Classification Report

CLASSIFIER	PRECISION	RECALL	F1SCORE
Blight	0.88	0.65	0.75
Common rust	0.93	0.98	0.96
Gray leaf spot	0.20	0.55	0.29
Healthy	0.92	0.90	0.91

- **Decision Tree Classification Algorithm**

  Confusion matrix

```
[[135   8  42  20]
 [  7 242  11   4]
 [ 54   5  46  15]
 [ 33   6  16 194]]
```

Classification Report

CLASSIFIER	PRECISION	RECALL	F1SCORE
Blight	0.59	0.66	0.62
Common rust	0.93	0.92	0.92
Gray leaf spot	0.40	0.38	0.39
Healthy	0.83	0.78	0.80

- **Naive Bayes Classifier**

Confusion matrix

```
[[163   6  52  36]
 [ 11 245   7   8]
 [ 45   7  49   9]
 [ 10   3   7 180]]
```

Classification Report

CLASSIFIER	PRECISION	RECALL	F1SCORE
Blight	0.71	0.63	0.67
Common rust	0.94	0.90	0.92
Gray leaf spot	0.43	0.45	0.44
Healthy	0.77	0.90	0.83

- **K-Nearest Neighbor(KNN)**

  Confusion matrix

```
[[173  17  63   5]
 [  2 234   1   0]
 [ 34   0  38   1]
 [ 20  10  13 227]]
```

Classification Report

CLASSIFIER	PRECISION	RECALL	F1SCORE
Blight	0.76	0.67	0.71
Common rust	0.90	0.99	0.94
Gray leaf spot	0.33	0.52	0.40
Healthy	0.97	0.84	0.90

- **Logistic Regression**

  Confusion matrix

```
[[174   8  48   3]
 [ 14 245   9   2]
 [ 30   1  52   1]
 [ 11   7   6 227]]
```

Classification Report

CLASSIFIER	PRECISION	RECALL	F1SCORE
Blight	0.76	0.75	0.75
Common rust	0.94	0.91	0.92
Gray leaf spot	0.45	0.62	0.52
Healthy	0.97	0.90	0.94

- **Adaptive boosting**

  Confusion matrix

```
[[192  14  80  37]
 [  4 241   1   2]
 [ 19   2  26   4]
 [ 14   4   8 190]]
```

Classification Report

CLASSIFIER	PRECISION	RECALL	F1SCORE
Blight	0.84	0.59	0.70
Common rust	0.92	0.97	0.95
Gray leaf spot	0.23	0.51	0.31
Healthy	0.82	0.88	0.85

Algorithm	**Accuracy**
Support Vector Machine Algorithm (SVM)	83.53%
Random Forest Algorithm	81.38%
Decision Tree Classification Algorithm	73.62%
Naive Bayes Classifier	76.014%
K-Nearest Neighbor (KNN)	80.19%
Logistic Regression	83.29%
Adaptive boosting	77.446%

Conclusion

This project successfully explored the potential of **machine learning (ML) approaches for automatic corn leaf disease recognition**. The implemented system demonstrates the feasibility of utilizing various ML techniques (mention specific techniques used) to accurately identify corn leaf diseases from image data. After extensive testing, the implementation discovered that support vector machine (svm) and logistic regression performs better than other optimizers and achieves an accuracy of 83.53% and 83.29%.And remaining machine alogrithms shows an better accuarcy .This study's effective use of deep learning models to classify plant diseases highlights the potential for automation in agriculture, which could result in early disease intervention, reduced crop losses, and eventually higher financial gains and agricultural productivity. **Key achievements include:**

- **Successful code compilation:** The project code successfully compiles and functions using chosen ML libraries, signifying a strong foundation for further development and refinement.
- **Promising disease recognition capabilities:** Initial testing and evaluation indicate the potential of the implemented ML models to accurately classify corn leaf diseases.

Moving forward:

- **Further training and optimization:** The next steps involve acquiring and incorporating a larger and more diverse dataset to enhance the model's accuracy and generalizability.
- **Evaluation and validation:** Rigorous testing and validation procedures are crucial to assess the model's performance under real-world conditions and ensure its reliability.
- **Integration and deployment:** Exploring potential integration with user interfaces and deployment strategies will facilitate the practical application of the system in agricultural settings.

This project lays the groundwork for a robust and practical **automatic corn leaf disease recognition system** using ML. By addressing the challenges of early and accurate disease detection, this technology has the potential to significantly contribute to improved crop health, yield optimization, and sustainable agricultural practices.

REFERENCES

[1] K.P. Ferentinos, Deep learning models for plant disease detection and diagnosis, Computers and Electronics in Agriculture, vol. 145, pp. 311-318, 2018

[2] K.K.Singh, “An Artificial Intelligence and Cloud-Based Collaborative Platform for Plant Disease Identification, Tracking and Forecasting for Farmers”, IEEE International Conference on Cloud Computing in Emerging Markets(CCEM), Bangalore, India, 23-24 Nov 2018, pp. 49-56.

[3] E.Agustina, I.Pratomo, A.D.Wibawa, S.Rahayu, “Expert System for Diagnosis Pests and Diseases of The Rice Plant using Forward Chaining and Certainty Factor Method”, International Seminar on Intelligent Technology and its applications(ISITIA), Surabaya, Indonesia, 28-29 Aug 2017, pp. 266-270.

[4] A.Louise P.de Ocampo, E.P.Dadios, “Mobile Platform Implementation of Lightweight Neural Network Model for Plant Disease Detection and Recognition”, IEEE 10th International conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management(HNICEM), Baguio, Philippines, 29 Nov-2 Dec2018.

[5] S.Dubey, M.Dixit, “Facial Expression Recognition using Deep Convolutional Neural Network”, International Symposium on Advanced Intelligent Informatics(SAIN) , Vol.8, No.1 pp. 96- 101,Aug 2018

[6] Md. Ashiqul Islam1, Md. Nymur Rahman Shuvo2, Muhammad Shamsojjaman3 Shazid Hasan4, Md. Shahadat Hossain5, Tania Khatun6 (2021), An Automated Convolutional Neural Network Based Approach for Paddy Leaf Disease Detection.

[7] Anwar Abdullah Alatawi (2022). Plant Disease Detection using AI based VGG - 16 Mode

[8] Keke Zhan, Qiufeng wu, Yiping chen (2021), Detecting soybean leaf disease from synthetic image using multi-feauture fusion faster R-CNN.

[9] PallapothalaTejaswini, Priyanshi Singh, Monica Ramchandani, Yogesh Kumar Rathore, Rekh Ram Janghel (2022), Rice leaf disease classification using cnn.

[10] Yasin Kaya, Ercan Gursoy (2023), A novel multi-head CNN design to identify plant diseases using the fusion of RGB images

[11] Mehmet Metin Ozguven,Kemal Adem (2019), Automatic detection and classification of leaf spot disease in sugar beet using deep learning algorithms.
[12] Garima Shrestha, Deepsikha, Majolica Das, Naiwrita day (2020), Plant Disease Detection Using CNN.