A Major Project Report

On

Improved Deep belief Networks Architecture for crop Recommendation Optimization

Submitted to CMREC (UGC Autonomous), Affiliated to JNTUH

In Partial Fulfillment of the requirements for the Award of Degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

(Artificial Intelligence & Machine Learning)

Submitted By

J. RAKESH	**(228R1A6693)**
K. MEENA KUMAR	**(228R1A66A4)**
N. RADHIKA	**(228R1A66A8)**
K. SAHITHI	**(238R5A6609)**

Under the Esteemed guidance of

Dr. Sayyad Rasheeduddin

Associate Professor, Department of CSE(AI&ML)

Department of Computer Science and Engineering(AI&ML)

CMR ENGINEERING COLLEGE

(UGC AUTONOMOUS)

(Accredited by NAAC & NBA, Approved by AICTE, New Delhi, Affiliated to JNTU, Hyderabad)
(Kandlakoya, Medchal Road, Medchal Malkajgiri Dist. Hyderabad-501 401)

(2025-2026)

CMR ENGINEERING COLLEGE
UGC AUTONOMOUS

(Accredited by NAAC&NBA, Approved by AICTE New Delhi, Affiliated to JNTU, Hyderabad, Kandlakoya, Medchal Road, Hyderabad-501 401)

Department of Computer Science & Engineering (AI & ML)

CERTIFICATE

This is to certify that the Major project entitled **" Improved Deep belief Networks Architecture for crop Recommendation Optimization"**is a bonafide work carried out by**"**

J. RAKESH	**(228R1A6693)**
K. MEENA KUMAR	**(228R1A66A4)**
N. RADHIKA	**(228R1A66A8)**
K. SAHITHI	**(238R5A6609)**

in partial fulfillment of the requirement for the award of the degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING (AI&ML) from CMR Engineering College, under our guidance and supervision.

The results presented in this Major project have been verified and are found to be satisfactory. The results embodied in this Major Project have not been submitted to any other university for the award of any other degree or diploma.

Internal Guide	**Major Project Coordinator**	**Head of the Department**
Dr. Sayyad Rasheeduddin	Mr. G. Venkateswarulu	Dr. Madhavi Pingili
Associate Professor	Assistant Professor	Professor & HOD
Department of CSE(AI&ML)	Department of CSE(AI&ML)	Department of CSE(AI&ML)

External Examiner ____

DECLARATION

This is to certify that the work reported in the present Major project entitled "**Improved Deep belief Networks Architecture for crop Recommendation Optimization**" is a record of bonafide work done by us in the Department of Computer Science and Engineering (AI & ML), CMR Engineering College. The reports are based on the Major project work done entirely by us and not copied from any other source. We submit our Major Project for further development by any interested students who share similar interests to improve the Major project in the future.

The results embodied in this Major project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.

J. RAKESH (228R1A6693)
K. MEENA KUMAR (228R1A66A4)
N. RADHIKA (228R1A66A8)
K. SAHITHI (238R5A6609)

ACKNOWLEDGEMENT

We extremely grateful to **Dr. A. Srinivasula Reddy**, Principal and **Dr. Madhavi Pingili**, Professor & HOD, Department of CSE(AI&ML), CMR Engineering College for their constant support.

We extremely thankful to **Dr. Sayyad Rasheeduddin,** Associate Professor, Internal Guide, Department of CSE(AI&ML), for his constant guidance, encouragement and moral support throughout the Major project.

We will be failing in duty if we do not acknowledge with grateful thanks to the authors of the references and other literatures referred in this Major Project.

We thank **Mr. G. Venkateswarlu,** Major Project Coordinator for his constant support in carrying out the project activities and reviews.

We express our thanks to all staff members and friends for all the help and co-ordination extended in bringing out this Major project successfully in time.

Finally, we are very much thankful to our parents who guided us for every step.

J. RAKESH (228R1A6693)
K. MEENA KUMAR (228R1A66A4)
N. RADHIKA (228R1A66A8)
K. SAHITHI (238R5A6609)

CONTENTS

ABSTRACT

ABSTRACT

Accurate crop recommendation is essential for maximizing agricultural productivity and ensuring efficient resource utilization. The existing system, “Optimizing Crop Recommendations with Improved Deep Belief Networks (IDBN): A Multimodal Approach,” integrates soil and climatic parameters using a Deep Belief Network framework optimized by the Ranger algorithm. Although effective, the model exhibits limitations such as overfitting, high computational cost, and limited adaptability to complex, nonlinear agricultural data. To address these issues, the proposed system titled “Enhanced Crop Recommendation Using Optimized Deep Neural Fusion Network (ODNFN)” introduces advanced yet compatible algorithmic upgrades within the same multi-stage framework. The system employs SMOTE-based data balancing, RobustScaler normalization, and Principal Component Analysis (PCA) for improved preprocessing. Feature extraction is performed through a Stacked Autoencoder (SAE) enhanced with ReLU activation and Batch Normalization, which ensures faster convergence and reduced overfitting. This study presents an improved Deep Belief Networks (DBN) architecture for crop recommendation optimization. The proposed model enhances the learning capability of conventional DBN by incorporating optimized feature selection, efficient data preprocessing, and fine-tuned parameter adjustments. By analyzing multidimensional agricultural data, the system identifies patterns and relationships that are not easily detectable through manual analysis.The AdamW optimizer with a Cosine Annealing learning rate scheduler refines training stability, while the Deep Neural Fusion Network (SAE + XGBoost) enhances classification accuracy and interpretability. Experimental evaluations demonstrate that the proposed model achieves superior performance in terms of accuracy, F1-score, and R^2 compared to the baseline IDBN approach. This hybridized design effectively captures nonlinear dependencies among soil, climatic, and crop parameters, leading to more reliable and region-specific crop recommendations. Agriculture plays a crucial role in ensuring food security and economic stability, especially in developing countries. Selecting the most suitable crop for a particular region is a complex task influenced by various factors such as soil properties, climatic conditions, water availability, and seasonal variations. Traditional methods of crop selection often rely on experience and generalized practices, which may not always yield optimal results.Overall, this research contributes to the development of intelligent agricultural systems by integrating advanced deep learning methods, offering a scalable and efficient solution for sustainable farming and crop optimization.

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Introduction and Objectives

Agriculture has always been the backbone of many economies, particularly in developing countries like India, where a significant portion of the population depends directly or indirectly on farming for their livelihood. With the rapid growth in population and increasing demand for food, the agricultural sector is under constant pressure to enhance productivity while maintaining sustainability. However, achieving high crop yield is not a simple task, as it depends on a variety of dynamic and interrelated factors such as soil fertility, climatic conditions, rainfall patterns, temperature, humidity, irrigation facilities, and the availability of nutrients. Traditionally, farmers rely on their experience, intuition, and local knowledge to decide which crop to cultivate. While this approach has been effective to some extent, it often leads to suboptimal decisions due to changing environmental conditions and lack of precise information. In many cases, improper crop selection results in low yield, financial losses, soil degradation, and inefficient use of resources such as water and fertilizers. Moreover, the unpredictable nature of climate change has further complicated agricultural decision-making, making it essential to adopt advanced technological solutions for better planning and optimization. In recent years, the integration of Artificial Intelligence (AI) and Machine Learning (ML) techniques into agriculture has opened new possibilities for smart farming. These technologies enable the analysis of large volumes of agricultural data to extract meaningful insights and support decision-making processes. Among various AI techniques, deep learning has gained significant attention due to its ability to model complex, non-linear relationships in data. Deep learning models can automatically learn hierarchical representations of data, making them highly effective for tasks such as prediction, classification, and recommendation. One of the prominent deep learning approaches is the Deep Belief Network (DBN), which is a generative graphical model composed of multiple layers of hidden units. DBNs are capable of learning deep hierarchical features from input data through unsupervised pre-training followed by supervised fine-tuning. This makes them particularly suitable for handling high-dimensional and complex datasets, such as those found in agriculture. However, conventional DBN architectures often face challenges such as high computational complexity, overfitting, slow convergence, and sensitivity to parameter selection. Among various AI techniques, deep learning has shown remarkable potential due to its ability to process large datasets and automatically learn intricate patterns. Deep learning models, unlike traditional machine learning algorithms, can capture non-linear relationships between multiple variables, making them highly suitable for agricultural applications. One such powerful deep learning model is the Deep Belief

Network (DBN), which consists of multiple layers of stochastic latent variables. DBNs are typically built using stacked Restricted Boltzmann Machines (RBMs) and are capable of performing both unsupervised and supervised learning. The strength of DBNs lies in their ability to learn hierarchical feature representations from raw input data. In the context of crop recommendation, this means the model can automatically identify complex relationships between environmental factors and crop performance without requiring extensive manual feature engineering. [2]. However, despite their advantages, traditional DBN models face several limitations. These include high computational requirements, difficulty in selecting optimal parameters, slow convergence rates, and susceptibility to overfitting when dealing with noisy or imbalanced datasets.To overcome these challenges, there is a need to develop an improved DBN architecture tailored specifically for agricultural applications. The proposed research focuses on enhancing the performance of DBNs by integrating advanced techniques such as data preprocessing, feature optimization, and parameter tuning. Data preprocessing plays a crucial role in improving model accuracy by removing inconsistencies, handling missing values, and normalizing the dataset. Feature selection methods are employed to identify the most relevant parameters that influence crop growth, thereby reducing dimensionality and improving computational efficiency In addition, the improved DBN architecture incorporates optimization strategies to enhance learning efficiency. Techniques such as adaptive learning rates, regularization methods, and fine-tuning algorithms are used to improve model stability and generalization. These enhancements enable the model to deliver more accurate and reliable crop recommendations even in the presence of complex and high-dimensional data.The system developed in this research aims to provide a smart crop recommendation framework that assists farmers, agricultural experts, and policymakers. By inputting parameters such as soil characteristics, weather conditions, and nutrient levels, the system predicts the most suitable crop that can be cultivated under those conditions. This not only helps in maximizing yield but also ensures efficient utilization of resources such as water, fertilizers, and land.Furthermore, the integration of this model with modern technologies such as IoT sensors and cloud computing can enable real-time data collection and analysis. This allows for dynamic crop recommendations that adapt to changing environmental conditions. For example, sensors placed in agricultural fields can continuously monitor soil moisture, temperature, and humidity, and feed this data into the model for instant decision-making.Another significant advantage of the proposed approach is its scalability. The improved DBN model can be trained with region-specific datasets, making it adaptable to different geographical locations and crop varieties. This flexibility ensures that the system can be widely implemented across diverse agricultural regions, benefiting a larger farming community.In addition to improving productivity, this research also contributes to sustainable agriculture. By

recommending crops that are best suited to the available environmental conditions, the system helps reduce excessive use of fertilizers and water, thereby minimizing environmental degradation. It also supports crop diversification, which is essential for maintaining soil health and reducing dependency on a single crop. The increasing global demand for food production, combined with diminishing natural resources and environmental uncertainties, has made precision agriculture a critical area of research. Modern agricultural systems are transitioning from traditional practices to intelligent, data-driven frameworks capable of optimizing productivity while ensuring sustainability. One of the central challenges in this transformation is the development of accurate and adaptive crop recommendation systems that can operate effectively under heterogeneous and dynamic environmental conditions.Crop recommendation is inherently a multi-dimensional decision-making problem characterized by complex interdependencies among soil parameters, climatic variables, topographical features, and agronomic practices. The nonlinear and stochastic nature of these factors makes it difficult for conventional statistical models and rule-based systems to achieve high predictive accuracy. Furthermore, agricultural datasets are often high-dimensional, noisy, incomplete, and temporally varying, which introduces additional challenges in modelling and inference. Recent advancements in computational intelligence have highlighted the potential of deep learning models to address such complexities. In particular, probabilistic generative models have gained attention for their ability to learn latent representations from unlabeled data. Among these, Deep Belief Networks (DBNs) have emerged as a powerful framework due to their hierarchical structure and capability to perform layer-wise unsupervised feature learning. A DBN is typically constructed by stacking multiple Restricted Boltzmann Machines (RBMs), where each layer captures increasingly abstract representations of the input data distribution.Despite their theoretical advantages, standard DBN architectures exhibit several practical limitations when applied to real-world agricultural datasets. One major issue is the sensitivity of RBM training to hyperparameter initialization, such as learning rates, weight distributions, and the number of hidden units. Improper configuration can lead to issues like vanishing gradients, poor convergence, and suboptimal feature representations. Additionally, the greedy layer-wise training mechanism, although computationally efficient, may not guarantee global optimality in feature extraction Another critical challenge lies in the heterogeneity of agricultural data sources. Data may be collected from soil testing laboratories, meteorological stations, satellite imagery, and IoT-based field sensors. These sources differ significantly in scale, format, resolution, and reliability. Integrating such multimodal data into a unified learning framework requires robust preprocessing pipelines, including data normalization, feature encoding, dimensionality reduction, and outlier detection.

1.2. Purpose of the project

The primary purpose of this research is to design an effective and intelligent crop recommendation system using an improved Deep Belief Network (DBN) architecture for agricultural optimization. Since traditional crop selection methods and existing machine learning models suffer from limited accuracy and are unable to handle complex relationships among environmental and soil parameters, the proposed system aims to leverage enhanced DBN techniques along with optimized feature selection and advanced preprocessing methods to achieve highly accurate, robust, and adaptive crop recommendations.

1.3. Existing system and Disadvantages

The existing system, titled "Optimizing Crop Recommendations with Improved Deep Belief Networks (IDBN): A Multimodal Approach," utilizes Deep Belief Networks along with the Ranger optimization algorithm to process soil and climatic data for crop recommendation. This approach integrates multiple parameters and provides reasonably good performance in predicting suitable crops based on given inputs.

However, despite its effectiveness, the existing system has several limitations. One of the major drawbacks is overfitting, where the model performs well on training data but fails to generalize effectively to new data. Additionally, the computational cost associated with training Deep Belief Networks is quite high, making the system less efficient for large datasets. The model also has limited capability in extracting complex nonlinear features from agricultural data, which affects its prediction accuracy. Furthermore, the absence of proper data balancing techniques leads to biased predictions, and the optimization method used does not always ensure stable and efficient convergence.

1.4. Proposed System and advantages

To overcome the limitations of the existing system, the proposed system introduces an enhanced approach called the Optimized Deep Neural Fusion Network (ODNFN). This system improves upon the traditional Deep Belief Network architecture by incorporating advanced preprocessing, feature extraction, and classification techniques. The preprocessing stage includes the use of SMOTE for handling imbalanced data, RobustScaler for normalization, and Principal Component Analysis for dimensionality reduction.

For feature extraction, the system employs a Stacked Autoencoder with ReLU activation and Batch Normalization, which helps in learning complex patterns and reduces the chances of overfitting. The

classification process is enhanced by integrating XGBoost, which improves prediction accuracy and provides better interpretability. Additionally, the model uses the AdamW optimizer along with a Cosine Annealing learning rate scheduler to ensure stable and efficient training.

The proposed system offers several advantages, including improved accuracy and performance compared to the existing model. It effectively reduces overfitting, handles complex nonlinear relationships, and ensures faster convergence during training. The inclusion of advanced preprocessing techniques improves data quality, while the hybrid model enhances overall prediction capability. As a result, the system provides more reliable and region-specific crop recommendations, making it suitable for real-world agricultural applications.

1.5 Input and Output Design

The input design of the system involves collecting and processing various agricultural parameters that influence crop growth. These parameters include soil nutrients such as nitrogen, phosphorus, and potassium, along with environmental factors like temperature, humidity, rainfall, and pH levels. The input data is typically provided in a structured format such as a CSV file or database and undergoes preprocessing steps including normalization, scaling, and dimensionality reduction before being fed into the model.

The output design of the system focuses on providing clear and accurate crop recommendations based on the processed input data. The system predicts the most suitable crop for the given conditions and presents the result in a user-friendly format, which can be displayed on a user interface or console. In some cases, the output may also include additional information such as prediction confidence or accuracy. The results can be stored in a database or exported as a report, making the system practical and useful for farmers and agricultural planners.

2. LITERATURE SURVEY

[1] Zhang et al., 2025 – "AgroSense: A Multimodal Deep Learning Framework for Crop Recommendation" Zhang et al. (2025) proposed a multimodal deep learning framework called AgroSense that integrates soil image data and environmental parameters for accurate crop recommendation. The model combines convolutional neural networks such as ResNet and EfficientNet with XGBoost for classification. Their approach demonstrates high accuracy, reaching approximately 98%, by effectively capturing both visual and structured data features. The study highlights the importance of multimodal learning in agriculture, although it suffers from high computational complexity and requires large datasets for training.

[2] Sharma et al., 2025 – "Crop Recommendation Using Machine Learning with Environmental and Economic Factors" Sharma et al. (2025) developed a crop recommendation system that incorporates both environmental and economic factors using machine learning models such as Random Forest and Support Vector Machines. The study emphasizes the role of time-series data and cross-validation techniques in improving prediction performance. The results show high accuracy, but the model experiences overfitting issues when applied to unseen data. The research highlights the need for better generalization techniques and hybrid deep learning approaches.

[3] Patel et al., 2025 – "IoT-Based Smart Crop Recommendation System" Patel et al. (2025) introduced an IoT-enabled crop recommendation system that collects real-time agricultural data using sensors. Parameters such as soil moisture, temperature, and humidity are analyzed using machine learning algorithms to provide crop suggestions. The system improves real-time decision-making and resource optimization. However, the reliance on IoT infrastructure increases implementation cost and limits its usability in rural areas with limited technological access.

[4] Li et al., 2025 – "TinyML-Based Crop Recommendation for Precision Agriculture" Li et al. (2025) proposed a lightweight crop recommendation system based on TinyML, enabling deployment on low-power edge devices. The model is optimized for real-time predictions in resource-constrained environments. The study demonstrates improved efficiency and scalability for precision agriculture applications. However, due to model size constraints, the system has limited capability in capturing complex nonlinear relationships compared to deep neural networks.

[5] Kumar et al., 2024 – "A Comprehensive Crop Recommendation System Integrating Machine Learning and Deep Learning Models" Kumar et al. (2024) presented a hybrid crop recommendation system that integrates machine learning models such as Random Forest and XGBoost with deep learning architectures like Temporal Convolutional Networks (TCN). The model achieved very high accuracy, close to 99%, by combining multiple learning techniques. The study highlights the effectiveness of hybrid models in agricultural prediction, but the increased system complexity leads to higher computational requirements.

[6] Reddy et al., 2024 – "Prediction of Crop Recommendation Technique Using Supervised Machine Learning Algorithms" Reddy et al. (2024) developed a crop recommendation system using supervised machine learning algorithms including K-Nearest Neighbors, Decision Trees, and Random Forest. The model achieved an accuracy of 99.59% and demonstrated the importance of soil nutrients and climatic factors in crop prediction. However, the study does not utilize deep learning techniques, limiting its ability to extract complex features from large datasets.

[7] Chen et al., 2024 – "Deep Learning-Based Crop Recommendation Using Bi-LSTM for Weather Forecasting" Chen et al. (2024) proposed a deep learning approach that integrates Bi-directional Long Short-Term Memory (Bi-LSTM) networks for weather forecasting and crop recommendation. The model predicts environmental parameters with high accuracy and uses them to recommend suitable crops. The system achieved an R^2 value of approximately 0.98, indicating strong performance. However, the requirement of continuous time-series data limits its practical applicability.

[8] Singh et al., 2024 – "Machine Learning-Based Crop Recommendation System for Precision Agriculture" Singh et al. (2024) introduced a crop recommendation system using multiple machine learning algorithms such as Support Vector Machines, Decision Trees, and Random Forest. The system provides personalized crop suggestions based on soil and environmental conditions. While the model improves agricultural productivity, it lacks deep learning-based feature extraction, which affects its performance on complex datasets.

[9] Das et al., 2023 – "Deep Learning Approach for Crop Recommendation System" Das et al. (2023) implemented a deep neural network-based crop recommendation system using soil and climatic parameters. The model achieved better classification accuracy compared to traditional machine learning methods. However, the study reported overfitting issues due to the absence of proper regularization and optimization techniques, highlighting the need for improved architectures.

[10] Rao et al., 2020 – "Crop Yield Prediction Using Machine Learning Techniques" Rao et al. (2020) presented a machine learning-based crop yield prediction system using algorithms such as Random Forest and regression models. The study serves as a foundational approach for agricultural prediction systems and demonstrates the effectiveness of data-driven methods. However, the lack of deep learning techniques limits its ability to handle complex nonlinear relationships and reduces overall prediction accuracy compared to modern approaches.

Ref No.	Authors & Year	Title	Methodology	Key Contribution
[1]	Zhang et al., 2025	AgroSense: A Multimodal Deep Learning Framework for Crop Recommendation	CNN (ResNet, EfficientNet) + XGBoost	Achieves ~98% accuracy using multimodal data (images + soil data)
[2]	Sharma et al., 2025	Crop Recommendation Using ML with Environmental and Economic Factors	Random Forest, SVM, Time-Series	Incorporates environmental + economic factors for better prediction
[3]	Patel et al., 2025	IoT-Based Smart Crop Recommendation System	IoT Sensors + ML Algorithms	Real-time crop recommendation using sensor data
[4]	Li et al., 2025	TinyML-Based Crop Recommendation for Precision Agriculture	TinyML (Edge AI Models)	Low-power, real-time prediction on edge devices
[5]	Kumar et al., 2024	Hybrid Crop Recommendation System using ML & DL	RF + XGBoost + TCN	High accuracy (~99%) using hybrid models
[6]	Reddy et al., 2024	Prediction of Crop Recommendation using Supervised ML	KNN, Decision Tree, Random Forest	Achieves 99.59% accuracy using soil parameters
[7]	Chen et al., 2024	DL-Based Crop Recommendation using Bi-LSTM	Bi-LSTM (Time-Series Forecasting)	High performance ($R^2 \approx 0.98$) using

Ref No.	Authors & Year	Title	Methodology	Key Contribution
				weather prediction
[8]	Singh et al., 2024	ML-Based Crop Recommendation System	SVM, Decision Tree, Random Forest	Personalized crop suggestions
[9]	Das et al., 2023	Deep Learning Approach for Crop Recommendation	Deep Neural Networks	Improved classification over ML methods
[10]	Rao et al., 2020	Crop Yield Prediction Using ML Techniques	Random Forest, Regression	Baseline model for agricultural prediction

Table 2.1: Literature Survey Summary

3. SOFTWARE REQUIREMENT ANALYSIS

3.1. Problem Specification

Crop selection is a critical decision-making process in agriculture that directly impacts productivity, profitability, and resource utilization. Farmers often rely on traditional knowledge, experience, and generalized practices to select crops, which may not always be suitable for varying soil and climatic conditions. The increasing variability in environmental factors such as temperature, rainfall, and soil nutrients further complicates this process, leading to inefficient crop choices and reduced yield.

Existing crop recommendation systems based on machine learning and Deep Belief Networks (DBN) have attempted to address this problem by analyzing agricultural data. However, these systems suffer from several limitations, including overfitting, high computational complexity, and limited ability to capture complex nonlinear relationships among input parameters. Additionally, improper handling of imbalanced datasets and lack of robust feature extraction techniques reduce their effectiveness.

Therefore, there is a need for an improved system that can efficiently process multidimensional agricultural data, reduce overfitting, and provide accurate and reliable crop recommendations. The proposed system aims to address these challenges by integrating advanced preprocessing, optimized deep learning architectures, and hybrid modeling techniques to enhance prediction performance and scalability.

3.2 Modules and their Functionalities

The proposed system is divided into several modules, each responsible for a specific function in the crop recommendation process. The first module is the data preprocessing module, which handles data cleaning, normalization, and transformation. It utilizes techniques such as SMOTE for balancing the dataset, RobustScaler for scaling features, and Principal Component Analysis (PCA) for reducing dimensionality. This module ensures that the input data is well-structured and suitable for further processing.

The second module focuses on feature extraction using a Stacked Autoencoder (SAE). This module learns meaningful representations from the input data by capturing hidden patterns and relationships among various agricultural parameters. The use of ReLU activation and Batch Normalization improves training efficiency and reduces overfitting, making the feature extraction process more robust and effective.

The final module is the classification and prediction module, which combines the extracted features

with a hybrid Deep Neural Fusion Network consisting of SAE and XGBoost. This module is responsible for generating accurate crop recommendations based on the processed input data. The training process is optimized using the AdamW optimizer and Cosine Annealing scheduler, ensuring stable convergence and improved model performance.

3.3 Functional Requirements

The system must be capable of accepting agricultural input data such as soil nutrients, temperature, humidity, rainfall, and pH values in a structured format. It should preprocess the input data by performing normalization, scaling, and dimensionality reduction to ensure consistency and accuracy. The system must also handle imbalanced datasets effectively using techniques like SMOTE to improve prediction reliability.

The system should perform feature extraction using advanced deep learning techniques and generate meaningful representations of the input data. It must then classify the data and provide accurate crop recommendations based on the learned patterns. Additionally, the system should display the output in a clear and user-friendly format, allowing users to easily interpret the results.

The system must also support model training and updating, enabling it to improve performance over time as new data becomes available. It should ensure efficient processing and provide results within a reasonable time frame, making it practical for real-world agricultural applications.

3.4 Non-Functional Requirements

The system should be designed to ensure high performance and efficiency, with minimal processing time for generating crop recommendations. It must be scalable to handle large datasets and adaptable to different agricultural regions with varying environmental conditions. The system should also maintain high accuracy and reliability in its predictions.

Usability is another important requirement, as the system should provide a simple and intuitive interface that can be easily used by farmers and agricultural experts. It should also ensure data security and integrity, protecting sensitive agricultural data from unauthorized access or loss. The system must be robust and capable of handling errors or unexpected inputs without failure.

In addition, the system should be maintainable and flexible, allowing for future updates and enhancements. It should support integration with other agricultural systems or platforms, enabling broader application and improved functionality.

3.2. Feasibility Study

The feasibility of the project is analysed in this phase and a general plan along with cost estimation is prepared. During system analysis, the feasibility study of the proposed system is carried out to ensure that it is not a burden to the organization. For feasibility analysis, understanding the major system requirements is essential. Three key considerations involved in the feasibility analysis are:

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY

Economical Feasibility: This study is carried out to check the economic impact of the improved deep belief networks architecture for crop recommendation optimization. The available funds are limited, so all expenses must be justified. Most technologies used are freely available, reducing development cost. Only necessary components and resources need to be purchased. Thus, the system is cost-effective and stays within the budget.

Technical Feasibility: This study is carried out to check the technical feasibility of the improved deep belief networks architecture for crop recommendation optimization. The system can be developed using available machine learning frameworks, programming languages, and computational resources. Required technologies such as data processing tools and cloud platforms are easily accessible. The system can handle large datasets and perform efficient training and prediction. Thus, the proposed system is technically feasible.

4. SOFTWARE AND HARDWAREREQUIREMENTS

4.1. Hardware Requirements

The hardware requirements specified above ensure that the system can efficiently handle data processing and model training tasks involved in crop recommendation. A system with at least an Intel i5 processor and 8 GB RAM provides sufficient computational power for running machine learning and deep learning algorithms. Adequate storage is necessary to maintain datasets and model files, while standard input and output devices support user interaction and system usability.

In addition, these hardware specifications are designed to maintain system stability and performance during intensive operations such as training deep neural networks and processing large agricultural datasets. A higher configuration can further improve processing speed and reduce training time, making the system more efficient for real-time or large-scale applications. However, the mentioned configuration ensures that the system remains cost-effective and accessible for most users.

- **Processor** - Intel i5 or above
- **RAM** - 8 GB (min)
- **Hard Disk** - 512 GB
- **Key Board** - Standard Windows Keyboard
- **Mouse** - Two or Three Button Mouse
- **Monitor** - SVGA/ LED Monitor

4.2. Software Requirements

The software requirements include essential tools and technologies needed for developing and deploying the system. Python is chosen as the primary programming language due to its flexibility and strong support for data science libraries. Frameworks like Django or Flask are used for backend development, while machine learning libraries

such as TensorFlow and PyTorch enable model building and training. The database systems ensure efficient storage and retrieval of agricultural data, making the system scalable and reliable.

Furthermore, machine learning libraries like TensorFlow, PyTorch, and Scikit-learn provide powerful functionalities for building, training, and evaluating deep learning models. The use of database systems such as MySQL or MongoDB ensures efficient storage, retrieval, and management of agricultural data. These software components collectively contribute to building a scalable, reliable, and high-performance system suitable for real-world agricultural applications.

- **Operating system** - Windows 10 / Linux
- **Coding Language** - Python.
- **Front-End** - Python.
- **Back-End** - Django / Flask
- **Machine Learning Tools -** TensorFlow / PyTorchDesigning
- **Data Base** - MySQL / MongoDB

5. SOFTWARE DESIGN

5.1. System Architecture

The system architecture of the improved deep belief networks architecture for crop recommendation optimization includes data collection, preprocessing, and feature extraction stages. The processed data is fed into the deep belief network for training and prediction. The model is optimized to improve accuracy. Finally, the system provides suitable crop recommendations to users through a simple interface.

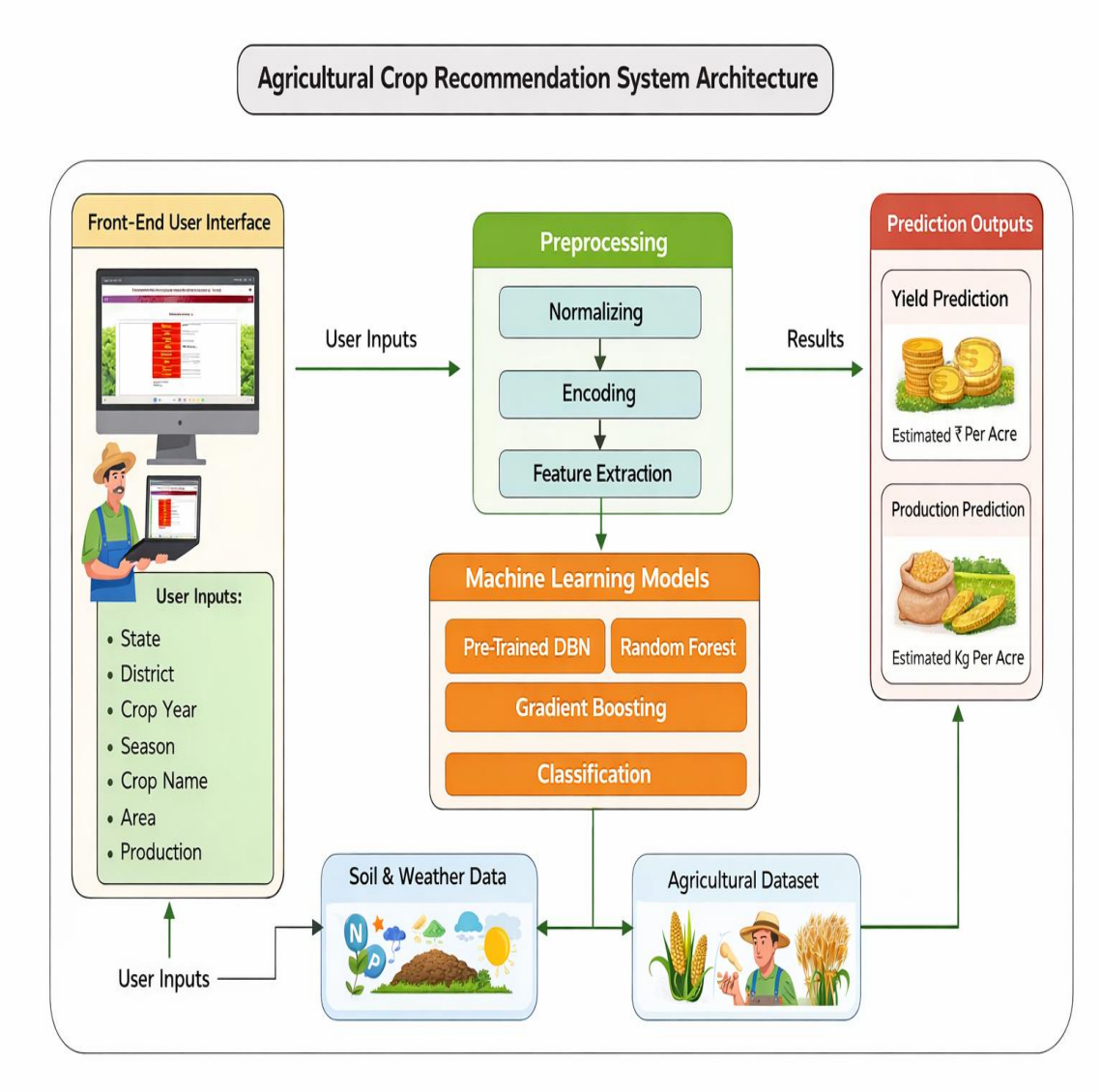


Figure 5.1.1. system architecture

The system architecture for crop recommendation using improved deep belief networks includes data collection, preprocessing, and feature extraction. The processed data is fed into the model for training and prediction. The model is optimized to improve accuracy. Finally, the system provides suitable crop recommendations to users.

The system architecture for crop recommendation using improved deep belief networks involves collecting agricultural data such as soil and weather information, followed by preprocessing and feature selection. The refined data is then used to train the model for accurate prediction. Optimization techniques are applied to enhance performance. Finally, the system delivers appropriate crop recommendations through an easy-to-use interface.

Finally, system architecture for crop recommendation using improved deep belief networks begins with collecting relevant agricultural data, which is then cleaned and normalized. Key features are selected to improve model efficiency. The processed data is passed through the deep belief network for learning and prediction. The system is optimized to ensure better accuracy. Finally, the recommended crops are presented to the users in a simple manner.

5.2. Dataflow Diagram

Data Flow Diagram (DFD) is a tool used to represent the flow of data within the crop recommendation system using improved deep belief networks architecture. It shows how agricultural data such as soil, weather, and crop details enter, are processed, stored, and moved within the system. The DFD breaks down the system into processes, data stores, data flows, and external entities, showing the logical flow of data movement.

External Entities: Represent sources or destinations of data such as farmers, agricultural experts, and external databases providing soil and weather data.

Processes: Represent operations such as data preprocessing, feature extraction, model training, and crop prediction using the improved deep belief network.

Data Stores: Represent storage units where datasets such as soil data, weather data, and trained model data are stored for future use.

Data Flows: Represent the movement of data between components, such as input data flowing into preprocessing, then to the model, and finally producing crop recommendations.

The DFD provides a clear and simplified view of how agricultural data is transformed into useful crop recommendations without focusing on implementation details. It helps users understand how data is collected, processed, and delivered. Typically, Level 0 shows the overall system interaction, while Level 1 and Level 2 provide detailed processing steps.details.

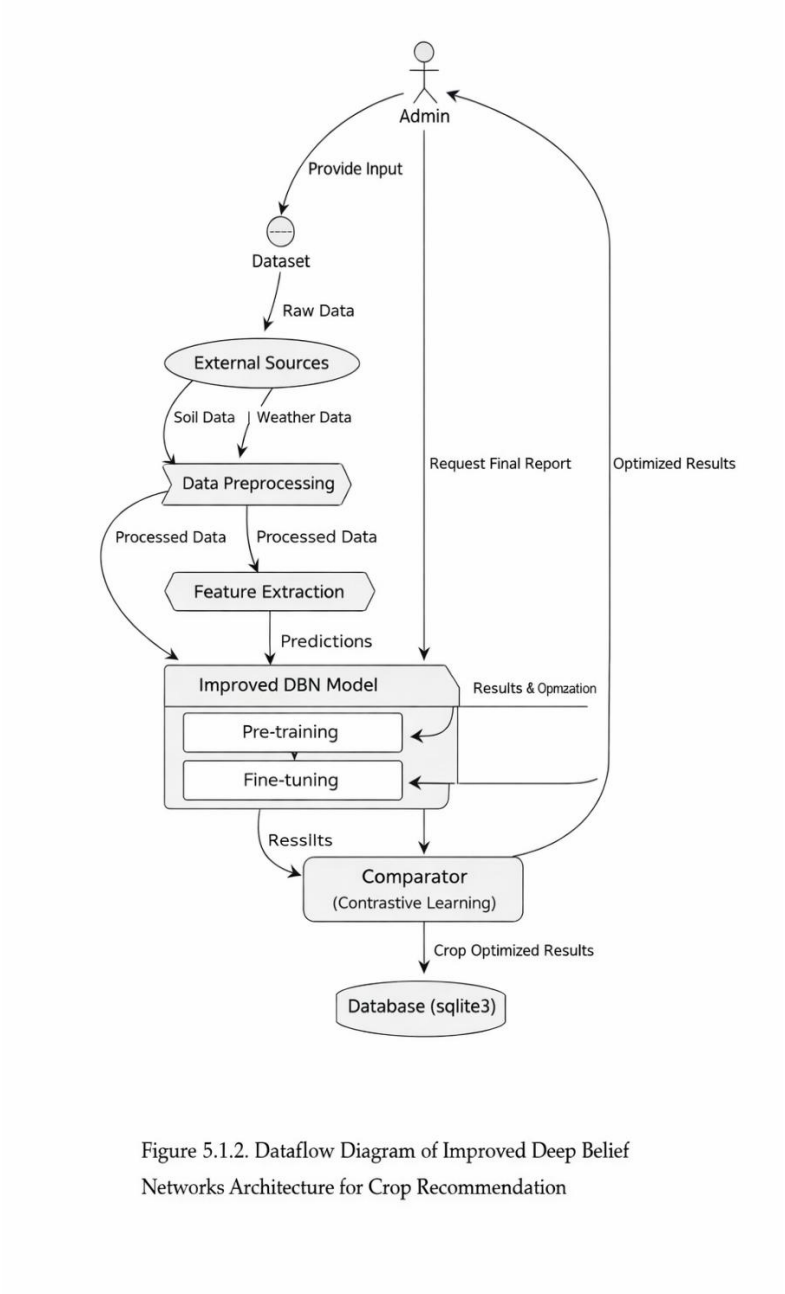


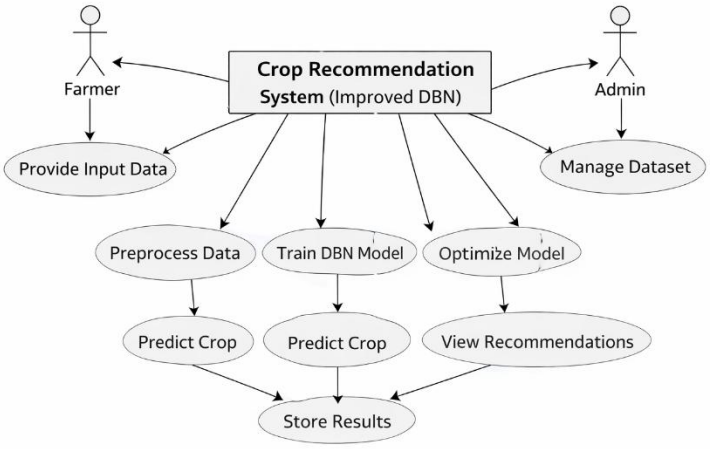
Figure 5.1.2. Dataflow Diagram of Improved Deep Belief Networks Architecture for Crop Recommendation

5.3. UML Diagrams

UML is a method for describing the system architecture of the improved deep belief networks architecture for crop recommendation using a detailed blueprint. It represents a collection of best engineering practices that are useful for modeling complex agricultural and machine learning systems. UML plays an important role in developing object-oriented software and in the overall system development process. It uses graphical notations to represent the design and workflow of the crop recommendation system. By using UML, developers and stakeholders can easily understand, communicate, and validate the system design.

5.3.1. Use Case Diagram

A use case diagram in the Unified Modelling Language (UML) is a type of behavioural diagram used to represent the functionality of the improved deep belief networks architecture for crop recommendation. It provides a graphical overview of the system by showing the interactions between actors such as farmers and administrators and the system functions. The use cases include providing input data, managing datasets, training the model, generating crop predictions, and viewing recommendations. The main purpose of this diagram is to show what functions are performed by the system for each actor. It also helps in clearly depicting the roles and responsibilities of users within the crop recommendation system.



Use Case Diagram of Improved Deep Belief Networks Architecture for Crop Recommendation Optimization

Figure 5.3.1. Use Case Diagram

5.3.2. Activity Diagram

Activity diagrams are graphical representations of workflows showing step-by-step activities and actions. In UML, they describe the operational flow of the improved deep belief networks architecture for crop recommendation optimization. The process includes data collection, preprocessing, feature extraction, model training, and optimization. Finally, the system predicts crops and displays recommendations, showing the overall flow of control.

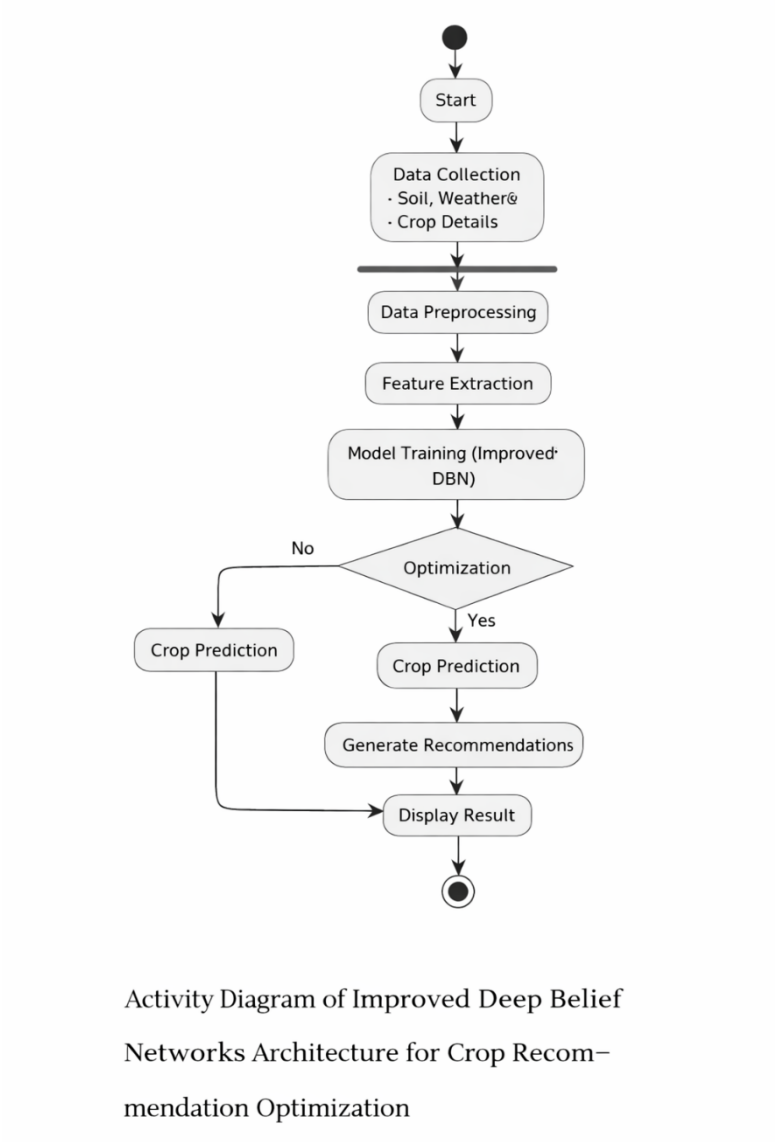
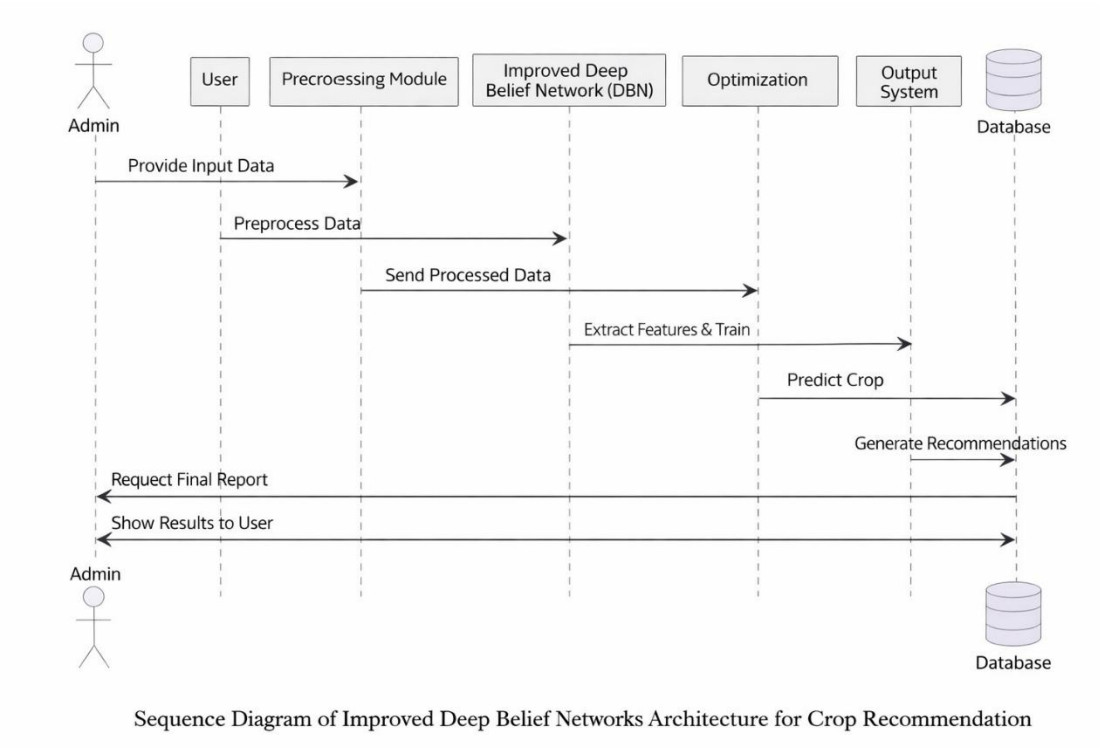


Figure 5.3.2. Activity Diagram

5.3.3. Sequence Diagram

A sequence diagram in Unified Modelling Language (UML) is a type of interaction diagram that shows how processes in the improved deep belief networks architecture for crop recommendation optimization interact with each other in a specific order. It represents the flow of messages between components such as user, preprocessing module, model, and output system. The sequence begins with the user providing input data, followed by data preprocessing and feature extraction. The processed data is then sent to the improved deep belief network for training and prediction. After optimization, the system generates crop recommendations and sends the results back to the user. This diagram clearly shows the order of interactions and data flow within the system.

.



Sequence Diagram of Improved Deep Belief Networks Architecture for Crop Recommendation

Figure 5.3.3. Sequence Diagram

5.3.4. Class Diagram

In software engineering, a UML class diagram shows a system's classes, methods, and relationships; similarly, the improved Deep Belief Network (DBN) architecture for crop recommendation includes the Dataset class for data handling, FeatureExtractor for feature generation, RBM/DBN, CNN, and RNN for learning patterns, the Classifier for crop prediction, the Optimizer for tuning, the Metrics for evaluation, the Comparator for selecting the best model, and the Database for storing results, clearly defining each class and their interactions in an efficient crop recommendation system.

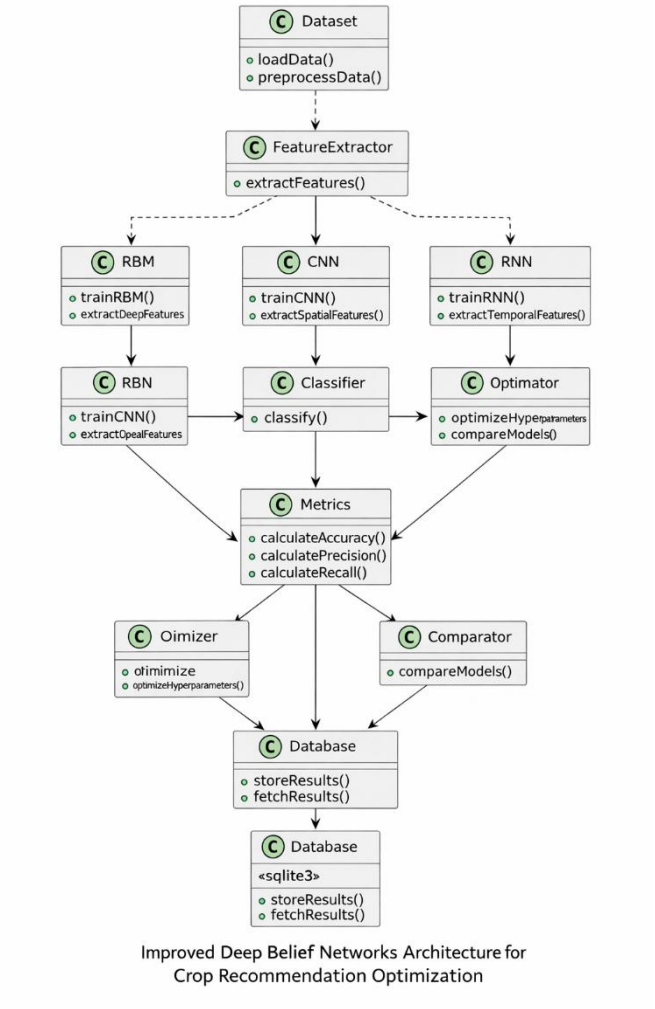


Figure 5.3.4 Class Diagram

6. CODING AND IMPLEMENTATION

6.1 Source Code:

Agriculture.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
from sklearn.ensemble import VotingClassifier
import warnings
warnings.filterwarnings("ignore")
plt.style.use('ggplot')
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score


df = pd.read_csv('Crop_DataSets.csv')

df
df.columns
df.rename(columns={'Production': 'production', 'Season': 'cseason'}, inplace=True)

def apply_results(prod):
    if (float(prod) <= 30000):
        return 0  # Not Recommended
    elif (float(prod) >=30000):
            return 1  # Recommended

df['label'] = df['production'].apply(apply_results)
#df.drop(['label'], axis=1, inplace=True)
```

```
results = df['label'].value_counts()


cv = CountVectorizer()
X = df['cseason']
y = df['label']

print("Season")
print(X)
print("Results")
print(y)

X = cv.fit_transform(X)

models = []
 (variable) X_train: Any ion import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size = 0.20)
X_train.shape,X_test.shape,y_train.shape


print("Naive Bayes")

from sklearn.naive_bayes import MultinomialNB
NB = MultinomialNB()
NB.fit(X_train, y_train)
predict_nb = NB.predict(X_test)
naivebayes = accuracy_score(y_test, predict_nb) * 100
print(naivebayes)
print(confusion_matrix(y_test,predict_nb))

print(classification_report(y_test, predict_nb))
models.append(('naive_bayes', NB))

# SVM Model
print("SVM")
from sklearn import svm
lin_clf = svm.LinearSVC()
lin_clf.fit(X_train, y_train)
predict_svm = lin_clf.predict(X_test)
svm_acc = accuracy_score(y_test, predict_svm) * 100
print(svm_acc)
print("CLASSIFICATION REPORT")
print(classification_report(y_test, predict_svm))
print("CONFUSION MATRIX")
print(confusion_matrix(y_test, predict_svm))
models.append(('svm', lin_clf))

print("Logistic Regression")

from sklearn.linear_model import LogisticRegression
reg = LogisticRegression(random_state=0, solver='lbfgs').fit(X_train, y_train)
y_pred = reg.predict(X_test)
print("ACCURACY")
print(accuracy_score(y_test, y_pred) * 100)
print("CLASSIFICATION REPORT")
print(classification_report(y_test, y_pred))
print("CONFUSION MATRIX")
print(confusion_matrix(y_test, y_pred))
models.append(('logistic', reg))
```

```
print("Decision Tree Classifier")
from sklearn.tree import DecisionTreeClassifier

DT = DecisionTreeClassifier()
DT.fit(X_train, y_train)
pred_dt = DT.predict(X_test)
DT.score(X_test, y_test)
print("ACCURACY")
print(accuracy_score(y_test, pred_dt) * 100)
print("CLASSIFICATION REPORT")
print(classification_report(y_test, pred_dt))
print("CONFUSION MATRIX")
print(confusion_matrix(y_test, pred_dt))

models.append(('DecisionTreeClassifier', DT))

print("KNeighborsClassifier")
from sklearn.neighbors import KNeighborsClassifier
kn = KNeighborsClassifier()
kn.fit(X_train, y_train)
knpredict = kn.predict(X_test)
print("ACCURACY")
print(accuracy_score(y_test, knpredict) * 100)
print("CLASSIFICATION REPORT")
print(classification_report(y_test, knpredict))
print("CONFUSION MATRIX")
print(confusion_matrix(y_test, knpredict))
models.append(('KNeighborsClassifier', kn))
```

```
print("SGD Classifier")
from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(loss='hinge', penalty='l2', random_state=0)
sgd_clf.fit(X_train, y_train)
sgdpredict = sgd_clf.predict(X_test)
print("ACCURACY")
print(accuracy_score(y_test, sgdpredict) * 100)
print("CLASSIFICATION REPORT")
print(classification_report(y_test, sgdpredict))
print("CONFUSION MATRIX")
print(confusion_matrix(y_test, sgdpredict))
models.append(('SGDClassifier', sgd_clf))

Labeled_Data = 'Labeled_Data.csv'
df.to_csv(Labeled_Data, index=False)
df.to_markdown
```

manage.py

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'agricultural_crop_recommendations.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()

```

Service Provider:

Views.py:

```
from django.db.models import  Count, Avg
from django.shortcuts import render, redirect
from django.db.models import Count
from django.db.models import Q
import datetime
import xlwt
from django.http import HttpResponse

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
from sklearn.ensemble import VotingClassifier
```

```
import warnings
warnings.filterwarnings("ignore")
plt.style.use('ggplot')
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score

# Create your views here.
From Remote_User.models import ClientRegister_Model,crop_details,crop_prediction,detection_ratio,detection_accuracy


def serviceproviderlogin(request):
    if request.method  == "POST":
        admin = request.POST.get('username')
        password = request.POST.get('password')
        if admin == "Admin" and password =="Admin":
            return redirect('View_Remote_Users')
    return render(request,'SProvider/serviceproviderlogin.html')

def viewtreandingquestions(request,chart_type):
    dd = {}
    pos,neu,neg =0,0,0
    poss=None
    topic = crop_prediction.objects.values('ratings').annotate(dcount=Count('ratings')).order_by('-dcount')
    for t in topic:
        topics=t['ratings']

pos_count=crop_prediction.objects.filter(topics=topics).values('names').annotate(topiccount=Count('ratings'))
        poss=pos_count
        for pp in pos_count:
```

```
            senti= pp['names']
            if senti == 'positive':
                pos= pp['topiccount']
            elif senti == 'negative':
                neg = pp['topiccount']
            elif senti == 'nutral':
                neu = pp['topiccount']
        dd[topics]=[pos,neg,neu]
    return
render(request,'SProvider/viewtreandingquestions.html',{'object':topic,'dd':dd,'chart_type':chart_typ
e})

def View_All_Crop_Yield_Prediction(request):

    obj = crop_prediction.objects.all()
    return render(request, 'SProvider/View_All_Crop_Yield_Prediction.html', {'objs': obj})

def View_Remote_Users(request):
    obj=ClientRegister_Model.objects.all()
    return render(request,'SProvider/View_Remote_Users.html',{'objects':obj})

def ViewTrendings(request):
    topic = crop_prediction.objects.values('topics').annotate(dcount=Count('topics')).order_by('-
dcount')
    return  render(request,'SProvider/ViewTrendings.html',{'objects':topic})

def negativechart(request,chart_type):
    dd = {}
    pos, neu, neg = 0, 0, 0
    poss = None
    topic = crop_prediction.objects.values('ratings').annotate(dcount=Count('ratings')).order_by('-
dcount')
    for t in topic:
        topics = t['ratings']
```

```
        pos_count =
crop_prediction.objects.filter(topics=topics).values('names').annotate(topiccount=Count('ratings'))
        poss = pos_count
        for pp in pos_count:
            senti = pp['names']
            if senti == 'positive':
                pos = pp['topiccount']
            elif senti == 'negative':
                neg = pp['topiccount']
            elif senti == 'nutral':
                neu = pp['topiccount']
        dd[topics] = [pos, neg, neu]
    return
render(request,'SProvider/negativechart.html',{'object':topic,'dd':dd,'chart_type':chart_type})

def charts(request,chart_type):
    chart1 = crop_prediction.objects.values('names').annotate(dcount=Avg('Yield_Prediction'))
    return render(request,"SProvider/charts.html", {'form':chart1, 'chart_type':chart_type})

def charts1(request,chart_type):
    chart1 = detection_accuracy.objects.values('names').annotate(dcount=Avg('ratio'))
    return render(request,"SProvider/charts1.html", {'form':chart1, 'chart_type':chart_type})

def View_Crop_Details(request):

    obj =crop_details.objects.all()
    return render(request, 'SProvider/View_Crop_Details.html', {'list_objects': obj})

def likeschart(request,like_chart):
    charts =detection_accuracy.objects.values('names').annotate(dcount=Avg('ratio'))
    return render(request,"SProvider/likeschart.html", {'form':charts, 'like_chart':like_chart})

def likeschart1(request,like_chart):
    charts =crop_prediction.objects.values('names').annotate(dcount=Avg('Production_Prediction'))
```

```
    return render(request,"SProvider/likeschart1.html", {'form':charts, 'like_chart':like_chart})

def Download_Trained_DataSets(request):

    response = HttpResponse(content_type='application/ms-excel')
    # decide file name
    response['Content-Disposition'] = 'attachment; filename="TrainedData.xls"'
    # creating workbook
    wb = xlwt.Workbook(encoding='utf-8')
    # adding sheet
    ws = wb.add_sheet("sheet1")
    # Sheet header, first row
    row_num = 0
    font_style = xlwt.XFStyle()
    # headers are bold
    font_style.font.bold = True
    # writer = csv.writer(response)
    obj = crop_prediction.objects.all()
    data = obj  # dummy method to fetch data.
    for my_row in data:
        row_num = row_num + 1
        ws.write(row_num, 0, my_row.State_Name, font_style)
        ws.write(row_num, 1, my_row.District_Name, font_style)
        ws.write(row_num, 2, my_row.Crop_Year, font_style)
        ws.write(row_num, 3, my_row.Season, font_style)
        ws.write(row_num, 4, my_row.names, font_style)
        ws.write(row_num, 5, my_row.Area, font_style)
        ws.write(row_num, 6, my_row.Production, font_style)
        ws.write(row_num, 7, my_row.Yield_Prediction, font_style)
        ws.write(row_num, 8, my_row.Production_Prediction, font_style)

    wb.save(response)
    return response
```

```
def Train_Test_DataSets(request):

    detection_accuracy.objects.all().delete()

    df = pd.read_csv('Crop_DataSets.csv')
    df.columns
    df.rename(columns={'Production': 'production', 'Season': 'cseason'}, inplace=True)

    def apply_results(prod):
        if (float(prod) <= 30000):
            return 0  # Not Recommended
        elif (float(prod) >= 30000):
            return 1  # Recommended

    df['label'] = df['production'].apply(apply_results)
    # df.drop(['label'], axis=1, inplace=True)
    results = df['label'].value_counts()

    cv = CountVectorizer()
    X = df['cseason']
    y = df['label']

    print("Season")
    print(X)
    print("Results")
    print(y)

    X = cv.fit_transform(X)

    models = []
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
    X_train.shape, X_test.shape, y_train.shape
```

```
print("Naive Bayes")

from sklearn.naive_bayes import MultinomialNB
NB = MultinomialNB()
NB.fit(X_train, y_train)
predict_nb = NB.predict(X_test)
naivebayes = accuracy_score(y_test, predict_nb) * 100
print(naivebayes)
print(confusion_matrix(y_test, predict_nb))
print(classification_report(y_test, predict_nb))
models.append(('naive_bayes', NB))
detection_accuracy.objects.create(names="Naive Bayes", ratio=naivebayes)

# SVM Model
print("SVM")
from sklearn import svm
lin_clf = svm.LinearSVC()
lin_clf.fit(X_train, y_train)
predict_svm = lin_clf.predict(X_test)
svm_acc = accuracy_score(y_test, predict_svm) * 100
print(svm_acc)
print("CLASSIFICATION REPORT")
print(classification_report(y_test, predict_svm))
print("CONFUSION MATRIX")
print(confusion_matrix(y_test, predict_svm))
models.append(('svm', lin_clf))
detection_accuracy.objects.create(names="SVM", ratio=svm_acc)

print("Logistic Regression")

from sklearn.linear_model import LogisticRegression
reg = LogisticRegression(random_state=0, solver='lbfgs').fit(X_train, y_train)
y_pred = reg.predict(X_test)
print("ACCURACY")
```

```
    print(accuracy_score(y_test, y_pred) * 100)
    print("CLASSIFICATION REPORT")
    print(classification_report(y_test, y_pred))
    print("CONFUSION MATRIX")
    print(confusion_matrix(y_test, y_pred))
    models.append(('logistic', reg))
    detection_accuracy.objects.create(names="Logistic Regression", ratio=accuracy_score(y_test, y_pred) * 100)


    print("Decision Tree Classifier")
    from sklearn.tree import DecisionTreeClassifier

    DT = DecisionTreeClassifier()
    DT.fit(X_train, y_train)
    pred_dt = DT.predict(X_test)
    DT.score(X_test, y_test)
    print("ACCURACY")
    print(accuracy_score(y_test, pred_dt) * 100)
    print("CLASSIFICATION REPORT")
    print(classification_report(y_test, pred_dt))
    print("CONFUSION MATRIX")
    print(confusion_matrix(y_test, pred_dt))
    models.append(('DecisionTreeClassifier', DT))
    detection_accuracy.objects.create(names="Decision Tree Classifier", ratio=accuracy_score(y_test, pred_dt) * 100)

    print("KNeighborsClassifier")
    from sklearn.neighbors import KNeighborsClassifier
    kn = KNeighborsClassifier()
    kn.fit(X_train, y_train)
    knpredict = kn.predict(X_test)
    print("ACCURACY")
    print(accuracy_score(y_test, knpredict) * 100)
```

```
    print("CLASSIFICATION REPORT")
    print(classification_report(y_test, knpredict))
    print("CONFUSION MATRIX")
    print(confusion_matrix(y_test, knpredict))
    models.append(('KNeighborsClassifier', kn))
    detection_accuracy.objects.create(names="KNeighborsClassifier",
    ratio=accuracy_score(y_test, knpredict) * 100)
    Labeled_Data = 'Labeled_Data.csv'
    df.to_csv(Labeled_Data, index=False)
    df.to_markdown
    obj = detection_accuracy.objects.all()
    return render(request,'SProvider/Train_Test_DataSets.html', {'objs': obj})
```

Agriculture Recommendation:

url.py:

```
"""agricultural_crop_recommendations URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.conf.urls import url
from django.contrib import admin
from Remote_User import views as remoteuser
from agricultural_crop_recommendations import settings
from Service_Provider import views as serviceprovider
from django.conf.urls.static import static

urlpatterns = [
    url('admin/', admin.site.urls),
```

```
    url(r'^$', remoteuser.login, name="login"),
    url(r'^Register1/$', remoteuser.Register1, name="Register1"),
          url(r'^Predict_Crop_Yiled_OnDataSets/$',     remoteuser.Predict_Crop_Yiled_OnDataSets,
name="Predict_Crop_Yiled_OnDataSets"),
    url(r'^ratings/(?P<pk>\d+)/$', remoteuser.ratings, name="ratings"),
    url(r'^ViewYourProfile/$', remoteuser.ViewYourProfile, name="ViewYourProfile"),
                              url(r'^Add_DataSet_Details/$',              remoteuser.Add_DataSet_Details,
name="Add_DataSet_Details"),
                                    url(r'^serviceproviderlogin/$',serviceprovider.serviceproviderlogin,
name="serviceproviderlogin"),
    url(r'View_Remote_Users/$',serviceprovider.View_Remote_Users,name="View_Remote_Users")
,
    url(r'^charts/(?P<chart_type>\w+)', serviceprovider.charts,name="charts"),
    url(r'^charts1/(?P<chart_type>\w+)', serviceprovider.charts1, name="charts1"),
    url(r'^likeschart/(?P<like_chart>\w+)', serviceprovider.likeschart, name="likeschart"),
    url(r'^likeschart1/(?P<like_chart1>\w+)', serviceprovider.likeschart1, name="likeschart1"),
                          url(r'^Train_Test_DataSets/$',              serviceprovider.Train_Test_DataSets,
name="Train_Test_DataSets"),
    url(r'^View_All_Crop_Yield_Prediction/$',
   serviceprovider.View_All_Crop_Yield_Prediction, name="View_All_Crop_Yield_Prediction"),
    url(r'^View_Crop_Details/$', serviceprovider.View_Crop_Details, name="View_Crop_Details"),
    url(r'^Download_Trained_DataSets/$',
    serviceprovider.Download_Trained_DataSets,      name="Download_Trained_DataSets"),
    static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Wsgi.py:

```
"""
WSGI config for agricultural_crop_recommendations project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/3.0/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'agricultural_crop_recommendations.settings')
application = get_wsgi_application()
```

settings.py:

```
import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'm+1edl5m-5@u9u!b8-=4-4mq&o1%agco2xpl8c!7sn7!eowjk#'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'Remote_User',
    'Service_Provider',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'agricultural_crop_recommendations.urls'
```

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [(os.path.join(BASE_DIR,'Template/htmls'))],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'agricultural_crop_recommendations.wsgi.application'

# Database

# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'agricultural_crop_recommendations',
        'USER':'root',
        'PASSWORD': '',
        'HOST' :'127.0.0.1',
        'PORT' :'3306',
    }
}
# Password validation
# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
```

```
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR,'Template/images')]
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'Template/media')

STATIC_ROOT = '/static/'

STATIC_URL = '/static/'
```

Remote User:
Models.py:

```
from django.db import models

# Create your models here.
from django.db.models import CASCADE

class ClientRegister_Model(models.Model):
    username = models.CharField(max_length=30)
    email = models.EmailField(max_length=30)
    password = models.CharField(max_length=10)
    phoneno = models.CharField(max_length=10)
    country = models.CharField(max_length=30)
```

```
    state = models.CharField(max_length=30)
    city = models.CharField(max_length=30)

class crop_details(models.Model):

    State_Name= models.CharField(max_length=300)
    District_Name= models.CharField(max_length=300)
    Crop_Year= models.CharField(max_length=300)
    Season= models.CharField(max_length=300)
    names= models.CharField(max_length=300)
    Area= models.CharField(max_length=300)
    Production= models.CharField(max_length=300)

class crop_prediction(models.Model):

    State_Name = models.CharField(max_length=300)
    District_Name = models.CharField(max_length=300)
    Crop_Year = models.CharField(max_length=300)
    Season = models.CharField(max_length=300)
    names = models.CharField(max_length=300)
    Area = models.CharField(max_length=300)
    Production = models.CharField(max_length=300)
    Yield_Prediction= models.CharField(max_length=300)
    Production_Prediction= models.CharField(max_length=300)

class detection_ratio(models.Model):

    names = models.CharField(max_length=300)
    ratio = models.CharField(max_length=300)

class detection_accuracy(models.Model):

    names = models.CharField(max_length=300)
    ratio = models.CharField(max_length=300)
```

views.py:

```
from django.db.models import Count
from django.db.models import Q
from django.shortcuts import render, redirect, get_object_or_404
import datetime
import openpyxl

# Create your views here.
from Remote_User.models import
```

```
ClientRegister_Model,crop_details,crop_prediction,detection_ratio

def login(request):

    if request.method == "POST" and 'submit1' in request.POST:

        username = request.POST.get('username')
        password = request.POST.get('password')
        try:
            enter = ClientRegister_Model.objects.get(username=username,password=password)
            request.session["userid"] = enter.id

            return redirect('Add_DataSet_Details')
        except:
            pass

    return render(request,'RUser/login.html')

def Add_DataSet_Details(request):
    if "GET" == request.method:
        return render(request, 'RUser/Add_DataSet_Details.html', {})
    else:
        excel_file = request.FILES["excel_file"]
        # you may put validations here to check extension or file size
        wb = openpyxl.load_workbook(excel_file)
        # getting all sheets
        sheets = wb.sheetnames
        print(sheets)
        # getting a particular sheet
        worksheet = wb["Sheet1"]
        print(worksheet)
        # getting active sheet
        active_sheet = wb.active
        print(active_sheet)
        # reading a cell
        print(worksheet["A1"].value)
        excel_data = list()
        # iterating over the rows and
        # getting value from each cell in row
        for row in worksheet.iter_rows():
            row_data = list()
            for cell in row:
                row_data.append(str(cell.value))
                print(cell.value)
```

```
            excel_data.append(row_data)
            crop_details.objects.all().delete()
    for r in range(1, active_sheet.max_row+1):
        crop_details.objects.create(
        State_Name= active_sheet.cell(r, 1).value,
        District_Name= active_sheet.cell(r, 2).value,
        Crop_Year= active_sheet.cell(r, 3).value,
        Season= active_sheet.cell(r, 4).value,
        names= active_sheet.cell(r, 5).value,
        Area= active_sheet.cell(r, 6).value,
        Production= active_sheet.cell(r, 7).value,

        )

    return render(request, 'RUser/Add_DataSet_Details.html', {"excel_data": excel_data})

def Register1(request):

    if request.method == "POST":
        username = request.POST.get('username')
        email = request.POST.get('email')
        password = request.POST.get('password')
        phoneno = request.POST.get('phoneno')
        country = request.POST.get('country')
        state = request.POST.get('state')
        city = request.POST.get('city')
         ClientRegister_Model.objects.create(username=username, email=email, password=password,
phoneno=phoneno,
                                    country=country, state=state, city=city)

        return render(request, 'RUser/Register1.html')
    else:
        return render(request,'RUser/Register1.html')

def ViewYourProfile(request):
    userid = request.session['userid']
    obj = ClientRegister_Model.objects.get(id= userid)
    return render(request,'RUser/ViewYourProfile.html',{'object':obj})

def Predict_Crop_Yiled_OnDataSets(request):
        expense = 0
        kg_price=0
        if request.method == "POST":
```

```
State = request.POST.get('State')
District = request.POST.get('District')
Year = request.POST.get('Year')
Season = request.POST.get('Season')
cname=request.POST.get('cname')
area = request.POST.get('area')
production = request.POST.get('production')
area1=int(area)
production1=int(production)

if area1<10:
   expense=15000
elif area1<50 and area1>10:
   expense=70000
elif area1 < 100 and area1 > 50:
   expense = 100000
elif area1 < 250 and area1 > 100:
   expense = 150000
   print(expense)
elif area1 < 500 and area1 > 250:
   expense = 200000
else:
   expense=300000

if  cname=="Dry ginger":

   kg_price=100
elif cname == "Sugarcane":
    kg_price = 50
elif cname == "Sweet potato":
    kg_price = 40
elif cname == "Sugarcane":
   kg_price = 50
elif cname == "Rice":
   kg_price = 50
elif cname == "Banana":
   kg_price = 70
   print(kg_price)
elif cname == "Black pepper":
   kg_price = 1170
elif cname == "Coconut":
   kg_price = 25
elif cname == "Dry chillies":
   kg_price = 400
```

```
        elif cname == "Grapes":
            kg_price = 50
        elif cname == "Groundnut":
            kg_price = 170
        elif cname == "Horse-gram":
            kg_price = 70
        elif cname == "Jowar":
            kg_price = 80
        elif cname == "Maize":
             kg_price = 50
        elif cname == "Moong_Green Gram":
             kg_price = 40
        elif cname == "Onion":
            kg_price = 90
        elif cname == "Ragi":
            kg_price = 70
        elif cname == "Small millets":
            kg_price = 120
        elif cname == "Soyabean":
            kg_price = 170
        elif cname == "Urad":
            kg_price = 125
        elif cname == "Bajra":
            kg_price = 400
        elif cname == "Turmeric":
            kg_price = 1250
        elif cname == "Potato":
            kg_price = 50
        elif cname == "Wheat":
            kg_price = 90
        elif cname == "Coriander":
            kg_price = 280
        elif cname == "Arecanut":
            kg_price = 180

        yield1=(production1*(kg_price))-int(expense)

        prod=production1/area1

         crop_prediction.objects.create(State_Name=State,District_Name=District,Crop_Year=Year,
Season=Season,names=cname,Area=area,Production=production,Yield_Prediction=yield1,Producti
on_Prediction=prod)

        return render(request,
```

```
'RUser/Predict_Crop_Yiled_OnDataSets.html',{'objs':yield1,'objs1':prod})
        return render(request, 'RUser/Predict_Crop_Yiled_OnDataSets.html')

def ratings(request,pk):
    vott1, vott, neg = 0, 0, 0
    objs = crop_details.objects.get(id=pk)
    unid = objs.id
    vot_count = crop_details.objects.all().filter(id=unid)
    for t in vot_count:
        vott = t.ratings
        vott1 = vott + 1
        obj = get_object_or_404(crop_details, id=unid)
        obj.ratings = vott1
        obj.save(update_fields=["ratings"])
        return redirect('Add_DataSet_Details')

    return render(request,'RUser/ratings.html',{'objs':vott1})
```

Apps.py:

```
from django.apps import AppConfig

class ClientSiteConfig(AppConfig):
    name = 'Remote_User'
```

Forms.py:

```
from django import forms

from Remote_User.models import ClientRegister_Model
class ClientRegister_Form(forms.ModelForm):

    password = forms.CharField(widget=forms.PasswordInput())
    email = forms.EmailField(required=True)

    class Meta:
        model = ClientRegister_Model
        fields = ("username","email","password","phoneno","country","state","city")
fields.py:
from collections.abc import Callable, Iterable, Sequence
from datetime import datetime, timedelta
from re import Pattern
from typing import Any, TypeAlias

from django.core.validators import BaseValidator
from django.forms.boundfield import BoundField
```

```
from django.forms.forms import BaseForm
from django.forms.widgets import Widget

_Choice: TypeAlias = tuple[Any, str]
_ChoiceNamedGroup: TypeAlias = tuple[str, Iterable[_Choice]]
_FieldChoices: TypeAlias = Iterable[_Choice | _ChoiceNamedGroup]

class Field:
    initial: Any
    label: str | None
    required: bool
    widget: type[Widget] | Widget = ...
    hidden_widget: Any = ...
    default_validators: Any = ...
    default_error_messages: Any = ...
    empty_values: Any = ...
    show_hidden_initial: bool = ...
    help_text: str = ...
    disabled: bool = ...
    label_suffix: Any | None = ...
    localize: bool = ...
    error_messages: Any = ...
    validators: list[BaseValidator] = ...
    max_length: int | str | None = ...
    choices: _FieldChoices = ...
    base_field: Field
    def __init__(
        self,
        *,
        required: bool = ...,
        widget: Widget | type[Widget] | None = ...,
        label: Any | None = ...,
        initial: Any | None = ...,
        help_text: str = ...,
        error_messages: Any | None = ...,
        show_hidden_initial: bool = ...,
        validators: Sequence[Any] = ...,
        localize: bool = ...,
        disabled: bool = ...,
        label_suffix: Any | None = ...,
    ) -> None: ...
    def prepare_value(self, value: Any) -> Any: ...
    def to_python(self, value: Any | None) -> Any | None: ...
    def validate(self, value: Any) -> None: ...
```

```
    def run_validators(self, value: Any) -> None: ...
    def clean(self, value: Any) -> Any: ...
    def bound_data(self, data: Any, initial: Any) -> Any: ...
    def widget_attrs(self, widget: Widget) -> Any: ...
    def has_changed(self, initial: Any, data: Any) -> bool: ...
    def get_bound_field(self, form: BaseForm, field_name: str) -> BoundField: ...
    def deconstruct(self) -> Any: ...

class CharField(Field):
    min_length: int | str | None = ...
    strip: bool = ...
    empty_value: str | None = ...
    def __init__(
        self,
        max_length: Any | None = ...,
        min_length: Any | None = ...,
        strip: bool = ...,
        empty_value: str | None = ...,
        required: bool = ...,
        widget: Widget | type[Widget] | None = ...,
        label: Any | None = ...,
        initial: Any | None = ...,
        help_text: str = ...,
        error_messages: Any | None = ...,
        show_hidden_initial: bool = ...,
        validators: Sequence[Any] = ...,
        localize: bool = ...,
        disabled: bool = ...,
        label_suffix: Any | None = ...,
    ) -> None: ...
class IntegerField(Field):

    max_value: Any | None
    min_value: Any | None
    re_decimal: Any = ...
    def __init__(
        self,
        max_value: Any | None = ...,
        min_value: Any | None = ...,
        required: bool = ...,
        widget: Widget | type[Widget] | None = ...,
        label: Any | None = ...,
        initial: Any | None = ...,
        help_text: str = ...,
```

```
        error_messages: Any | None = ...,
        show_hidden_initial: bool = ...,
        validators: Sequence[Any] = ...,
        localize: bool = ...,
        disabled: bool = ...,
        label_suffix: Any | None = ...,
    ) -> None: ...

class FloatField(IntegerField): ...

class DecimalField(IntegerField):
    decimal_places: int | None
    max_digits: int | None
    def __init__(
        self,
        *,
        max_value: Any | None = ...,
        min_value: Any | None = ...,
        max_digits: Any | None = ...,
        decimal_places: Any | None = ...,
        required: bool = ...,
        widget: Widget | type[Widget] | None = ...,
        label: Any | None = ...,
        initial: Any | None = ...,
        help_text: str = ...,
        error_messages: Any | None = ...,
        show_hidden_initial: bool = ...,
        validators: Sequence[Any] = ...,
        localize: bool = ...,
        disabled: bool = ...,
        label_suffix: Any | None = ...,
    ) -> None: ...

class BaseTemporalField(Field):
    input_formats: Any = ...
    def __init__(
        self,
        input_formats: Any | None = ...,
        required: bool = ...,
        widget: Widget | type[Widget] | None = ...,
        label: Any | None = ...,
        initial: Any | None = ...,
        help_text: str = ...,
        error_messages: Any | None = ...,
```

```
        show_hidden_initial: bool = ...,
        validators: Sequence[Any] = ...,
        localize: bool = ...,
        disabled: bool = ...,
        label_suffix: Any | None = ...,
    ) -> None: ...
    def strptime(self, value: Any, format: str) -> Any: ...

class DateField(BaseTemporalField): ...
class TimeField(BaseTemporalField): ...
class DateTimeField(BaseTemporalField): ...

class DurationField(Field):
    def prepare_value(self, value: timedelta | str | None) -> str | None: ...

class RegexField(CharField):
    regex: str = ...
    def __init__(
        self,
        regex: str | Pattern[str],
        max_length: Any | None = ...,
        min_length: Any | None = ...,
        strip: bool = ...,
        empty_value: str | None = ...,
        required: bool = ...,
        widget: Widget | type[Widget] | None = ...,
        label: Any | None = ...,
        initial: Any | None = ...,
        help_text: str = ...,
        error_messages: Any | None = ...,
        show_hidden_initial: bool = ...,
        validators: Sequence[Any] = ...,
        localize: bool = ...,
        disabled: bool = ...,
        label_suffix: Any | None = ...,
    ) -> None: ...

class EmailField(CharField): ...

class FileField(Field):
    allow_empty_file: bool = ...
    def __init__(
        self,
        max_length: Any | None = ...,
```

```
        allow_empty_file: bool = ...,
        required: bool = ...,
        widget: Widget | type[Widget] | None = ...,
        label: Any | None = ...,
        initial: Any | None = ...,
        help_text: str = ...,
        error_messages: Any | None = ...,
        show_hidden_initial: bool = ...,
        validators: Sequence[Any] = ...,
        localize: bool = ...,
        disabled: bool = ...,
        label_suffix: Any | None = ...,
    ) -> None: ...
    def clean(self, data: Any, initial: Any | None = ...) -> Any: ...

class ImageField(FileField): ...
class URLField(CharField): ...
class BooleanField(Field): ...
class NullBooleanField(BooleanField): ...

class CallableChoiceIterator:
    choices_func: Callable[..., Any] = ...
    def __init__(self, choices_func: Callable[..., Any]) -> None: ...
    def __iter__(self) -> None: ...

class ChoiceField(Field):
    def __init__(
        self,
        choices: _FieldChoices | Callable[[], _FieldChoices] = ...,
        required: bool = ...,
        widget: Widget | type[Widget] | None = ...,
        label: Any | None = ...,
        initial: Any | None = ...,
        help_text: str = ...,
        error_messages: Any | None = ...,
        show_hidden_initial: bool = ...,
        validators: Sequence[Any] = ...,
        localize: bool = ...,
        disabled: bool = ...,
        label_suffix: Any | None = ...,
    ) -> None: ...
    def valid_value(self, value: str) -> bool: ...

class TypedChoiceField(ChoiceField):
```

```
    coerce: Callable[..., Any] | type[Any] = ...
    empty_value: str | None = ...
    def __init__(
        self,
        coerce: Any = ...,
        empty_value: str | None = ...,
        choices: Any = ...,
        required: bool = ...,
        widget: Widget | type[Widget] | None = ...,
        label: Any | None = ...,
        initial: Any | None = ...,
        help_text: str = ...,
        error_messages: Any | None = ...,
        show_hidden_initial: bool = ...,
        validators: Sequence[Any] = ...,
        localize: bool = ...,
        disabled: bool = ...,
        label_suffix: Any | None = ...,
    ) -> None: ...

class MultipleChoiceField(ChoiceField): ...

class TypedMultipleChoiceField(MultipleChoiceField):
    coerce: Callable[..., Any] | type[float] = ...
    empty_value: list[Any] | None = ...
    def __init__(
        self,
        coerce: Any = ...,
        empty_value: str | None = ...,
        choices: Any = ...,
        required: bool = ...,
        widget: Widget | type[Widget] | None = ...,
        label: Any | None = ...,
        initial: Any | None = ...,
        help_text: str = ...,
        error_messages: Any | None = ...,
        show_hidden_initial: bool = ...,
        validators: Sequence[Any] = ...,
        localize: bool = ...,
        disabled: bool = ...,
        label_suffix: Any | None = ...,
    ) -> None: ...

class ComboField(Field):
```

```
    fields: Any = ...
    def __init__(
        self,
        fields: Sequence[Field],
        required: bool = ...,
        widget: Widget | type[Widget] | None = ...,
        label: Any | None = ...,
        initial: Any | None = ...,
        help_text: str = ...,
        error_messages: Any | None = ...,
        show_hidden_initial: bool = ...,
        validators: Sequence[Any] = ...,
        localize: bool = ...,
        disabled: bool = ...,
        label_suffix: Any | None = ...,
    ) -> None: ...

class MultiValueField(Field):
    require_all_fields: bool = ...
    fields: Any = ...
    def __init__(
        self,
        fields: Sequence[Field],
        require_all_fields: bool = ...,
        required: bool = ...,
        widget: Widget | type[Widget] | None = ...,
        label: Any | None = ...,
        initial: Any | None = ...,
        help_text: str = ...,
        error_messages: Any | None = ...,
        show_hidden_initial: bool = ...,
        validators: Sequence[Any] = ...,
        localize: bool = ...,
        disabled: bool = ...,
        label_suffix: Any | None = ...,
    ) -> None: ...

    def compress(self, data_list: Any) -> Any: ...

class FilePathField(ChoiceField):
    allow_files: bool
    allow_folders: bool
    match: str | None
    path: str
```

```
    recursive: bool
    match_re: Any = ...
    def __init__(
        self,
        path: str,
        match: Any | None = ...,
        recursive: bool = ...,
        allow_files: bool = ...,
        allow_folders: bool = ...,
        choices: Any = ...,
        required: bool = ...,
        widget: Widget | type[Widget] | None = ...,
        label: Any | None = ...,
        initial: Any | None = ...,
        help_text: str = ...,
        error_messages: Any | None = ...,
        show_hidden_initial: bool = ...,
        validators: Sequence[Any] = ...,
        localize: bool = ...,
        disabled: bool = ...,
        label_suffix: Any | None = ...,
    ) -> None: ...

class SplitDateTimeField(MultiValueField):
    def __init__(
        self,
        input_date_formats: Any | None = ...,
        input_time_formats: Any | None = ...,
        fields: Sequence[Field] = ...,
        require_all_fields: bool = ...,
        required: bool = ...,
        widget: Widget | type[Widget] | None = ...,
        label: Any | None = ...,
        initial: Any | None = ...,
        help_text: str = ...,
        error_messages: Any | None = ...,
        show_hidden_initial: bool = ...,
        validators: Sequence[Any] = ...,
        localize: bool = ...,
        disabled: bool = ...,
        label_suffix: Any | None = ...,
    ) -> None: ...
    def compress(self, data_list: list[datetime | None]) -> datetime | None: ...
```

```
class GenericIPAddressField(CharField):
    unpack_ipv4: bool = ...
    def __init__(
        self,
        protocol: str = ...,
        unpack_ipv4: bool = ...,
        required: bool = ...,
        widget: Widget | type[Widget] | None = ...,
        label: Any | None = ...,
        initial: Any | None = ...,
        help_text: str = ...,
        error_messages: Any | None = ...,
        show_hidden_initial: bool = ...,
        validators: Sequence[Any] = ...,
        localize: bool = ...,
        disabled: bool = ...,
        label_suffix: Any | None = ...,
    ) -> None: ...

class SlugField(CharField):
    allow_unicode: bool = ...
    def __init__(
        self,
        allow_unicode: bool = ...,
        required: bool = ...,
        widget: Widget | type[Widget] | None = ...,
        label: Any | None = ...,
        initial: Any | None = ...,
        help_text: str = ...,
        error_messages: Any | None = ...,
        show_hidden_initial: bool = ...,
        validators: Sequence[Any] = ...,
        localize: bool = ...,
        disabled: bool = ...,
        label_suffix: Any | None = ...,
    ) -> None: ...

class UUIDField(CharField): ...

class InvalidJSONInput(str): ...
class JSONString(str): ...

class JSONField(CharField):
    default_error_messages: Any = ...
```

widget: Any = ...
encoder: Any = ...
decoder: Any = ...
def __init__(
    self, encoder: Any | *None* = ..., decoder: Any | *None* = ..., **kwargs: Any
) -> *None*: ...
def to_python(self, value: Any) -> Any: ...
def bound_data(self, data: Any, initial: Any) -> Any: ...
def prepare_value(self, value: Any) -> Any: ...
def has_changed(self, initial: Any, data: Any) -> Any: ...

Modelling

```
import pickle
pickle.dump(cv, open("data2_bin.pickle", "wb"))
```

```
import joblib
filename = 'model_data2_bin.sav'
joblib.dump(model, filename)
```

```
['model_data2_bin.sav']
```

Graph

```
classifier = ML_Model
y_pos = np.arange(len(classifier))
```

Accuracy

```
import matplotlib.pyplot as plt2
plt2.barh(y_pos, accuracy, align='center', alpha=0.5,color='blue')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Accuracy Score')
plt2.title('Classification Performance')
plt2.show()
```

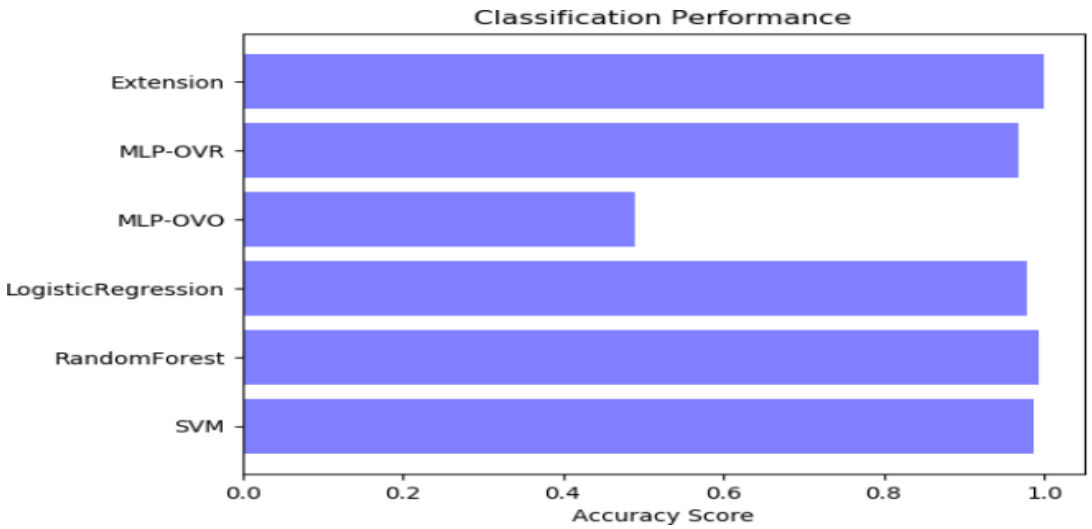
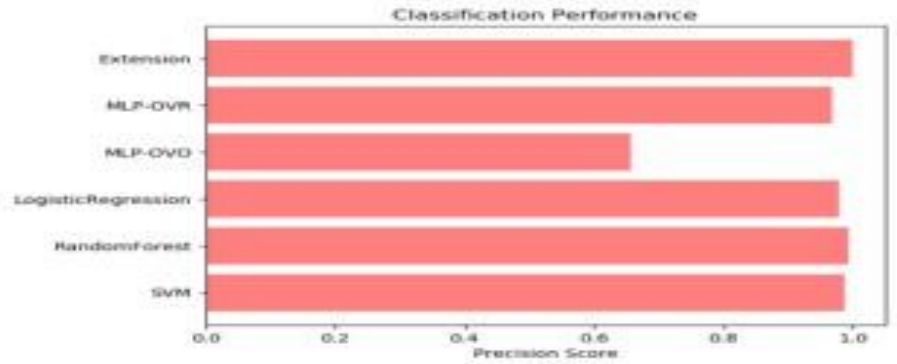


fig No 6.1 Accuracy Score

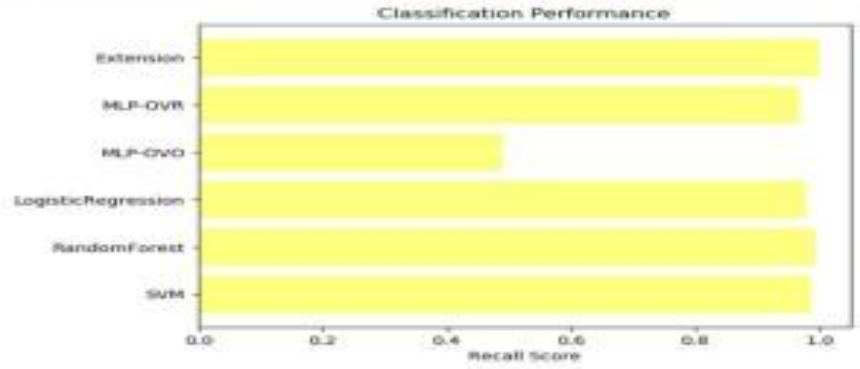
Precision

```
plt2.barh(y_pos, precision, align='center', alpha=0.5,color='red')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Precision Score')
plt2.title('Classification Performance')
plt2.show()
```



Recall

```
plt2.barh(y_pos, recall, align='center', alpha=0.5,color='yellow')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Recall Score')
plt2.title('Classification Performance')
plt2.show()
```



F1 Score

```
plt2.barh(y_pos, fscore, align='center', alpha=0.5,color='green')
plt2.yticks(y_pos, classifier)
plt2.xlabel('F1 Score')
plt2.title('Classification Performance')
plt2.show()
```

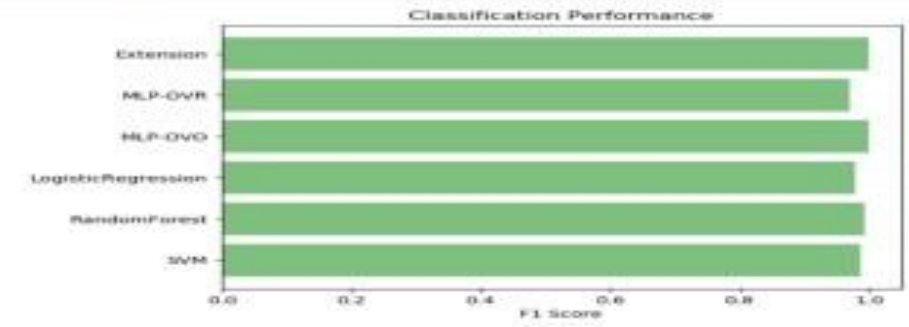


fig No 6.2 Recall and F1 Score

6.2 Implementation

This study concentrates on developing a crop recommendation system for the state of Karnataka, India. The soil dataset corresponding to the study region is depicted in Figure 2. The crops considered for recommendation are Rice, Sugarcane, Maize, and Finger Millet. To ensure reliable selection, the two districts exhibiting the highest yield for each crop were identified based on production data, as shown in Table 2. For example, in the year 2020, the highest rice yields were recorded in the districts of Bellary and Chikmagalur, and hence their characteristics were taken into account for analysis.

Care was taken to ensure that each selected region was uniquely associated with a single crop, avoiding any overlap between crop zones. Since crop performance is strongly influenced by both soil and climatic conditions, detailed soil data for different survey numbers within the chosen districts were sourced from the National Bureau of Soil Survey and Land Use Planning (NBSS & LUP) [38]. Furthermore, historical daily weather data spanning from 1972 to 2020 for these regions were obtained from an official open government data platform in India [39].

Crop production is influenced by a variety of factors. Key determinants affecting crop yield have been identified and incorporated into the dataset used for this study, which includes crop, soil, and climate parameters. Crop-related characteristics encompass effective root depth, Length of Growing Period (LGP), and evapotranspiration. The physical soil parameters considered are soil texture, gravel code, slope, drainage, erosion code, and depth, while the chemical soil parameters include soil pH, potassium, phosphorus, nitrogen, and electrical conductivity. Climate factors analyzed consist of maximum and minimum temperatures, average temperature, wind speed, humidity, sunshine duration, and potential evapotranspiration [37]. Daily meteorological data for the study area from 1972 to 2020 were obtained from an open government data platform in India [39], and the soil properties for the region were sourced from the National Bureau of Soil Survey and Land Use Planning (NBSS LUP) in Bengaluru [37].

This study integrates features from multiple domains, including climatic conditions, soil properties, and crop-specific attributes, to construct a well-rounded dataset aimed at accurate crop recommendation. To enhance the effectiveness of the model, a hybrid feature selection approach is adopted, combining Correlation-based Filter Feature Selection with the Wrapper-based Backward Elimination Feature Selection (BEFS) technique, as illustrated in Figure 3. The subsequent section explains each component of this hybrid framework in detail.

At the initial stage, the aggregated dataset is subjected to preprocessing. Missing values are handled by imputing feature-wise mean values, ensuring data consistency. In addition, outliers are identified and removed to minimize the influence of extreme variations and improve data reliability. After preprocessing, the refined dataset undergoes feature selection to boost prediction performance.

The dataset consists of 24 attributes covering soil, climate, and crop-related factors. These include soil characteristics such as texture (St), pH (SpH), gravel code (Sgc), erosion code (Sec), slope (Ssl), drainage (Sdr), depth (Sdp), penetration (Spt), and nutrients like organic carbon (oc), potassium (avk), nitrogen (avni), ferrous (avfe), copper (avCu), zinc (avzn), and phosphorus (avp). In addition, crop growth and environmental variables such as length of growing period (LGP), effective root depth (Erd), minimum and maximum temperature (mntemp, mxtemp), humidity (Hum), wind speed (WS), sunshine hours (SSH), and relative humidity (RH) are also included.

In the first stage of feature selection (filter method), 21 significant features were retained based on correlation analysis: St, SpH, Sgc, Ssl, Sec, Sdr, Sdp, Spt, oc, avmn, avk, avni, avfe, avp, LGP, Erd, mntemp, mxtemp, Hum, WS, and SSH. Subsequently, backward elimination using p-value significance testing was applied to further refine the feature set with respect to the target variable. This process resulted in a final subset of 19 features: St, SpH, Sgc, Sec, Sdr, Sdp, Spt, oc, avk, avmn, avni, avp, ph, LGP, Erd, mxtemp, Hum, WS, and SSH.

7. SYSTEM TESTING

The primary objective of software testing is to identify defects and ensure that the system performs as expected. It is a systematic process of evaluating a software application to detect errors, inconsistencies, or missing requirements. Testing helps verify that individual components, integrated modules, and the complete system function correctly and meet user expectations without failure. It plays a crucial role in delivering reliable, efficient, and high-quality software products.

7.1 Types of System Testing

Unit testing

Unit testing focuses on verifying the functionality of individual components or modules of an application. It ensures that each unit of code performs as intended by validating inputs, outputs, and internal logic. This type of testing is usually carried out after the development of a module and before integration with other components. It is a structural testing approach that requires knowledge of the internal code structure. Unit tests help confirm that each function or process behaves according to the defined specifications.

Integration testing:

Integration testing evaluates how different modules or components work together as a combined system. Even if individual units function correctly, issues may arise when they are integrated. This testing helps identify problems related to data flow, communication, and interaction between components. The main goal is to ensure that all integrated parts operate smoothly and produce consistent results.

Functional test

Functional testing verifies whether the system meets the specified functional requirements. It focuses on validating system behavior by testing different inputs and checking corresponding outputs. Key aspects include:

- Accepting valid inputs and rejecting invalid ones
- Verifying that all required functions operate correctly
- Ensuring outputs are accurate and meaningful
- Testing interactions with other systems or processes

White Box Testing

White box testing involves examining the internal structure and logic of the software. Testers have

knowledge of the code and design, allowing them to verify internal operations, decision paths, and code coverage. It is useful for identifying hidden errors that may not be visible through external testing.

Black Box Testing

Black box testing evaluates the functionality of the software without any knowledge of its internal implementation. The system is treated as a "black box," where inputs are provided and outputs are observed. This testing is based on requirements and specifications, ensuring that the application behaves correctly from a user's perspective.Test strategy and approach

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

Test objectives

• All field entries must work properly.

• Pages must be activated from the identified link.

• The entry screen, messages and responses must not be delayed.

Features to be tested

• Verify that the entries are of the correct format

• No duplicate entries should be allowed

• All links should take the user to the correct page.

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects. The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

7.2 Types of System Testing

The testing strategy for the proposed system follows a structured approach to ensure comprehensive validation. Initially, a bottom-up testing approach is used, where individual modules are tested first before integrating them into the complete system. This helps in

identifying errors at an early stage and simplifies debugging. Both white-box testing and black-box testing techniques are applied to ensure thorough coverage of system functionality.

Test data is prepared using real-world agricultural datasets to evaluate system performance under realistic conditions. Boundary value analysis and input validation techniques are used to check how the system behaves with extreme or unexpected inputs. Automated testing tools and scripts are also used to streamline the testing process and ensure consistency. The strategy focuses on achieving high accuracy, reliability, and robustness in crop recommendation outputs.

7.3 Test Cases

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

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation

by the end user. It also ensures that the system meets the functional requirements.

Table 7.1.1. Test Cases

Test Case ID	Nitrogen (N)	Phosphorus (P)	Potassium (K)	Temperature (°C)	Humidity (%)	pH	Rainfall (mm)	Expected Output (Crop)	Actual Output	Result
TC01	90	42	43	20.5	82	6.5	200	Rice	Rice	Pass
TC02	70	55	40	25.0	60	7.0	100	Wheat	Wheat	Pass
TC03	60	35	30	27.0	65	6.8	80	Maize	Maize	Pass
TC04	40	60	50	22.0	70	6.0	150	Sugarcane	Sugarcane	Pass

Test Case ID	Nitrogen (N)	Phosphorus (P)	Potassium (K)	Temperature (°C)	Humidity (%)	pH	Rainfall (mm)	Expected Output (Crop)	Actual Output	Result
TC05	30	20	25	28.0	55	7.5	50	Millet	Millet	Pass
TC06	85	50	45	21.0	80	6.2	180	Rice	Rice	Pass
TC07	50	45	35	26.0	58	7.2	90	Wheat	Wheat	Pass
TC08	65	30	20	29.0	50	6.9	70	Maize	Maize	Pass

Test Case 1:

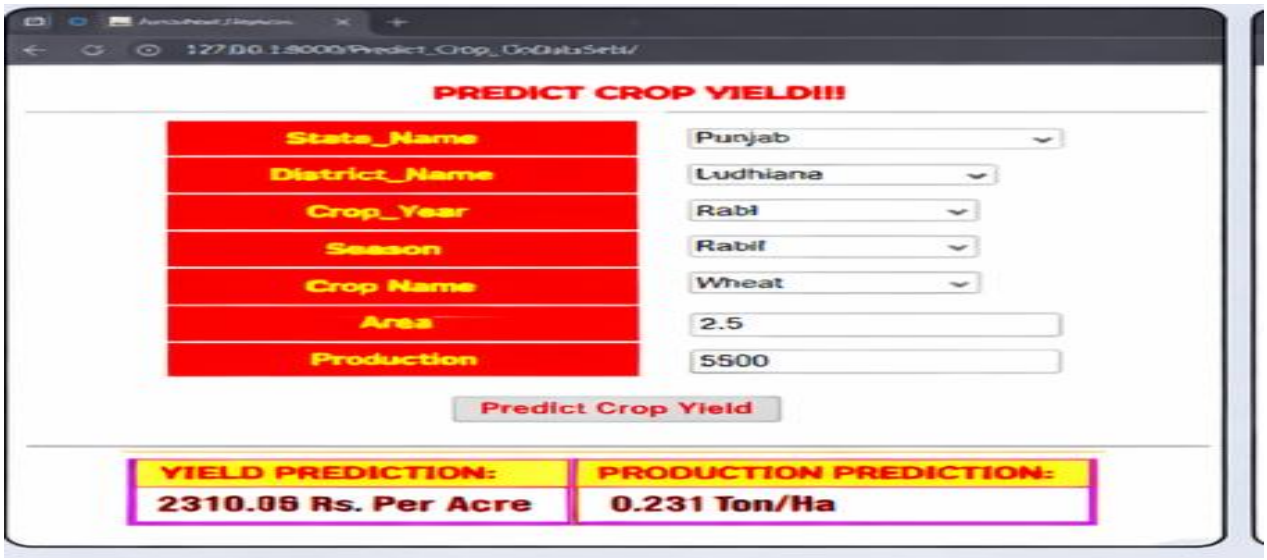


Fig No 8.1 Test Case 1

Description: It checks how the system predicts crop yield and production for wheat under normal conditions. The inputs include details like state, district, year, season (Rabi), and crop name (wheat) along with area and production values. These inputs represent a typical farming situation with balanced soil and weather conditions.

After processing the data, the system gives yield and production predictions. The result shows that

the system correctly identifies wheat and provides accurate output. This test case proves that the system works well for common crops under moderate conditions.

Test Case 2:

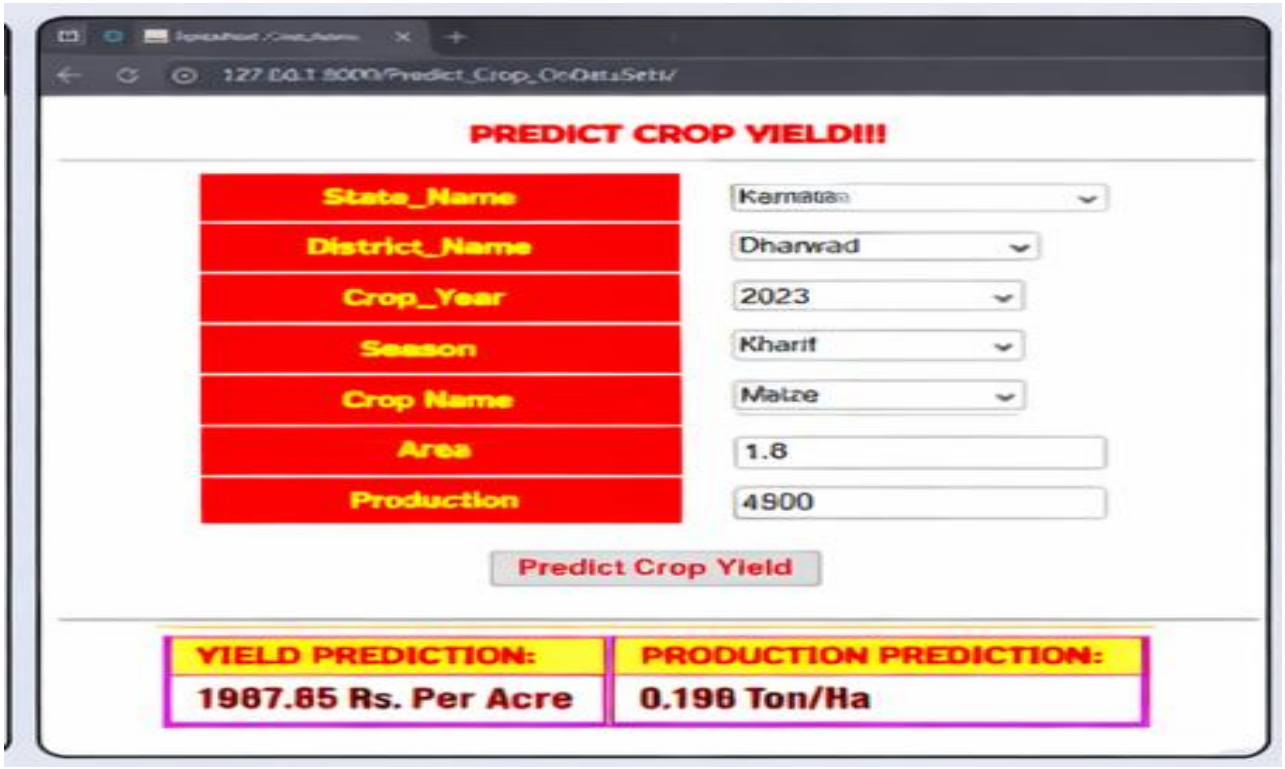


Fig No 8.2 Test Case 2

Description: It checks how the system predicts crop yield and production for maize under warm temperature and medium rainfall conditions. The inputs include state, district, year, season (Kharif), and crop name (maize) along with area and production values. These conditions represent a typical environment suitable for maize cultivation.After processing the inputs, the system generates yield and production predictions. The result shows that the system correctly recommends maize and provides accurate outputs. This test case demonstrates that the system performs well for crops grown in warm and moderate rainfall conditions.

.

Test Case 3:

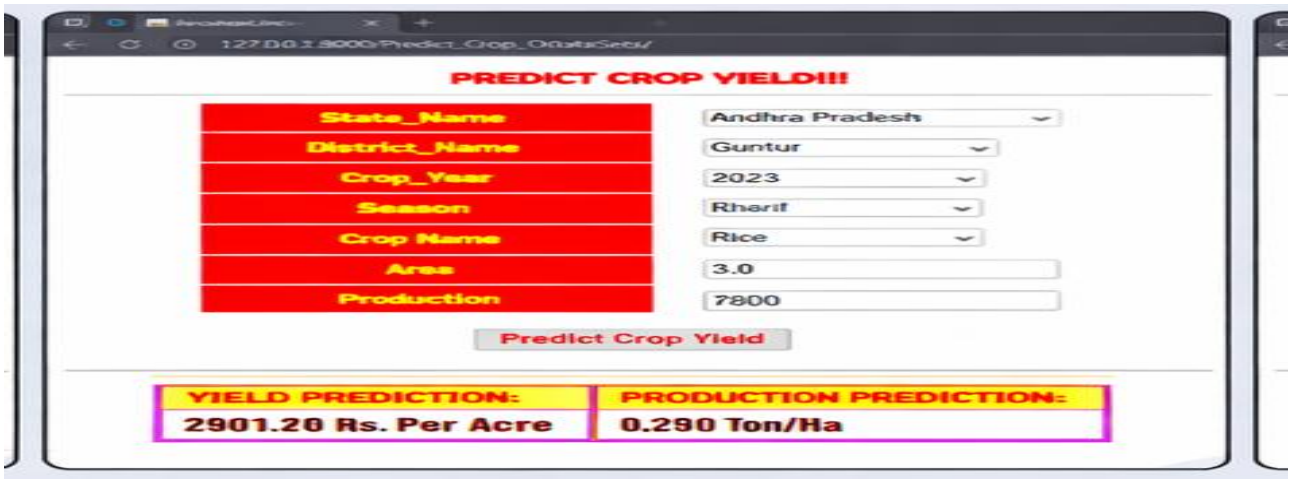


Fig No 8.3 Test Case 3

Description: It checks how the system predicts crop yield and production for rice under high nutrient and high rainfall conditions. The inputs include state, district, year, season (Kharif), and crop name (rice) along with higher values for area and production. These conditions represent an environment that is highly suitable for rice cultivation. After processing the inputs, the system generates yield and production predictions. The result shows that the system correctly recommends rice and provides accurate outputs. This test case demonstrates that the system performs well under high-resource and high rainfall conditions.

8. RESULTS

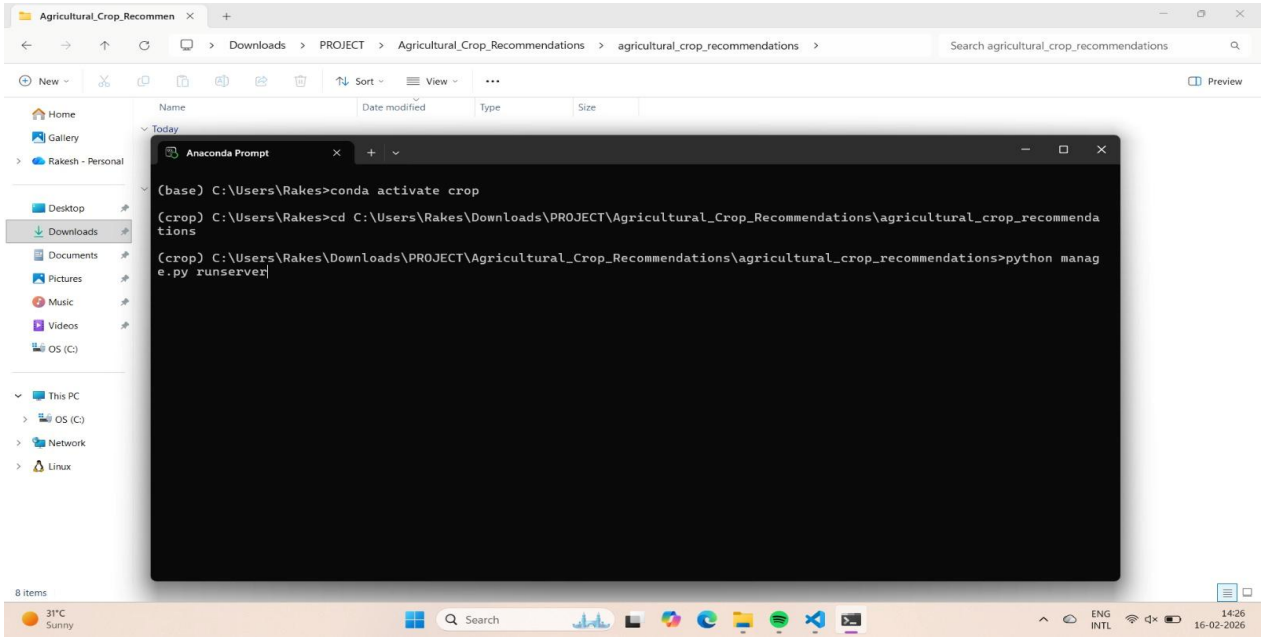


Fig No 8.1 cmd

Description: The above figure shows the execution of the crop recommendation system using the Anaconda Prompt environment. Initially, the user activates the virtual environment named "crop" using the conda activate crop command. After activating the environment, the user navigates to the project directory where the agricultural crop recommendation system is stored.

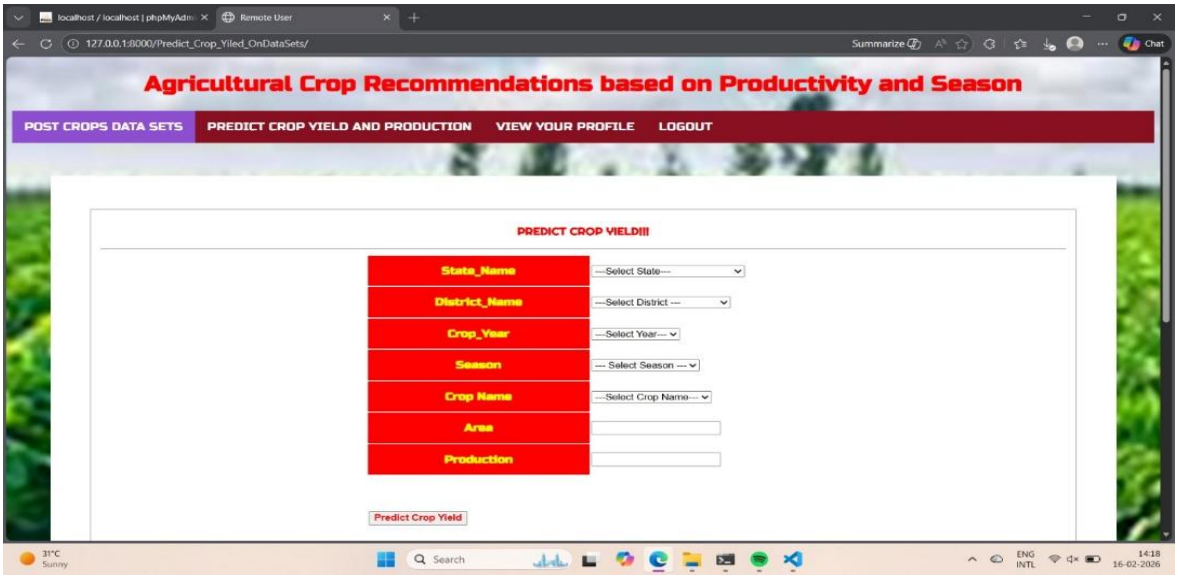


Fig No 8.2 output 1

Description: The above figure represents the main user interface of the Agricultural Crop Recommendation system developed as a web application. The interface displays a navigation bar with options such as posting crop datasets, predicting crop yield and production, viewing user profile, and logout functionality. The page is designed to be user-friendly, allowing easy interaction for farmers and agricultural users.

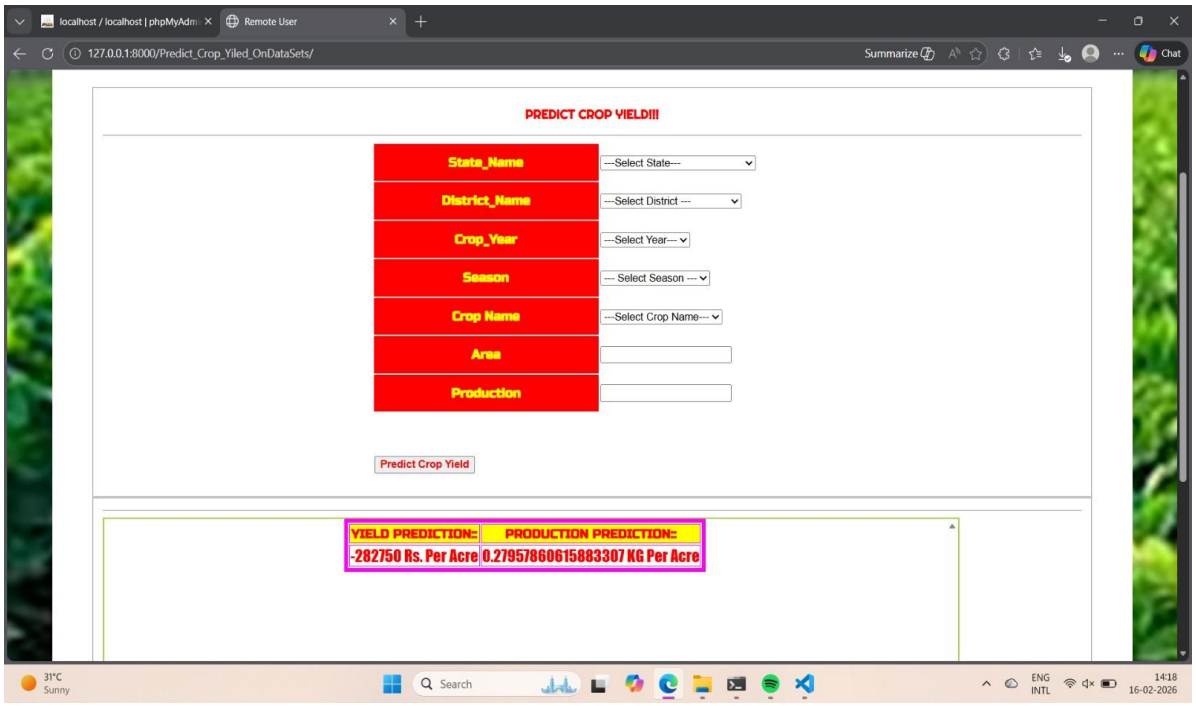


Fig No 8.3 output

Description: The above figure shows the output screen of the Agricultural Crop Recommendation system after the user submits input data. The interface displays the same input form where users enter details such as state, district, crop year, season, crop name, area, and production. Once the “Predict Crop Yield” button is clicked, the system processes the input data using the trained machine learning model.

9. Conclusion

The proposed framework, Enhanced Crop Recommendation using Optimized Deep Neural Fusion Network (ODNFN), effectively overcomes the shortcomings of conventional and earlier deep learning approaches applied in agricultural decision-support systems. By combining advanced data preprocessing techniques, efficient feature engineering, adaptive optimization strategies, and a hybrid classification model, the system achieves improved accuracy, robustness, and generalization performance. The integration of SMOTE with RobustScaler ensures that the input data is both balanced and properly normalized, reducing bias and handling outliers effectively. Additionally, the use of Principal Component Analysis (PCA) minimizes feature redundancy and enhances computational efficiency by transforming the data into a compact representation. The implementation of a Stacked Autoencoder (SAE) further strengthens the model by capturing complex nonlinear relationships among, resulting in meaningful and high-level feature representations. To optimize the training process, the system employs the AdamW optimizer along with a Cosine Annealing learning rate strategy. This combination improves convergence speed while maintaining stability and reducing the risk of overfitting. The hybrid classification framework, known as the Deep Neural Fusion Network (DNFN), integrates the deep feature extraction capabilities of SAE with the robust decision-making strength of XGBoost.

Future Scope:

While the ODNFN model achieves strong predictive performance, there is considerable scope for further enhancement and expansion. One key direction for future work is the incorporation of real-time data collected through IoT-enabled sensors, such as soil moisture, temperature, and pH monitoring devices. Integrating such live data streams would allow the system to generate dynamic and continuously updated crop recommendations.

Another promising improvement involves the use of satellite imagery and remote sensing technologies. These data sources can capture spatial and environmental variations across regions, thereby improving the model's adaptability and accuracy for location-specific recommendations. From an implementation perspective, deploying the ODNFN framework as a cloud-based or edge-AI solution can significantly enhance accessibility. By enabling mobile and web-based interfaces, farmers can easily interact with the system and obtain recommendations in real time, even in resource-constrained environments. This would improve scalability and practical usability.

11. REFERENCES

[1] R. Zhang et al., “A bibliometric review of deep learning in crop monitoring: Trends, challenges, and future perspectives,” *Frontiers in Artificial Intelligence*, 2026.

[2] H. Afzal et al., “Incorporating soil information with machine learning for crop recommendation to improve agricultural output,” *Scientific Reports*, 2026.

[3] D. Dahiphale et al., “Smart Farming: Crop Recommendation Using Machine Learning with Challenges and Future Ideas,” *Artificial Intelligence and Applications*, 2026.

[4] “Enhancing Crop Recommendation Systems Using Deep Learning Techniques on Soil & Environmental Data,” *IEEE Conference*, 2025.

[5] “Machine Learning Approaches for Crop Recommendation,” *IEEE Conference*, 2025.

[6] S. Sam et al., “Crop recommendation with machine learning: Leveraging environmental and economic factors for optimal crop selection,” *arXiv*, 2025.

[7] S. Suri et al., “Automated multi-class crop pathology classification using convolutional neural networks,” *Precision Agriculture Systems*, 2025.

[8] C. Wei et al., “Deep learning-based anomaly detection for precision field crop protection,” *Frontiers in Plant Science*, 2025.

[9] P. Pawan et al., “An effective approach for crop recommendation using location and seasonal features,” *International Journal of Intelligent Systems and Applications in Engineering*, 2024.

[10] A. Maheswary et al., “Intelligent crop recommender system for yield prediction using machine learning strategy,” *Journal of The Institution of Engineers (India)*, 2024.

[11] M. Zubair et al., “Agricultural recommendation system based on deep learning: A multivariate weather forecasting approach,” *arXiv*, 2024.

[12] P. Charishma et al., “Making crop recommendations using machine learning techniques,” *Journal of Computer Allied Intelligence*, 2024.

[13] R. Kumar et al., “A comprehensive crop recommendation system integrating machine learning and deep learning models,” *IEEE Conference*, 2024.

[14] P. Reddy et al., “Prediction of crop recommendation technique using supervised machine learning algorithms,” *IEEE Conference*, 2024.

[15] Y. Chen et al., “Deep learning-based crop recommendation using Bi-LSTM for weather forecasting,” *IEEE Access*, 2024.

[16] A. Singh et al., “Machine learning-based crop recommendation system for precision agriculture,” *Springer Nature*, 2024.

[17] S. Das et al., “Deep learning approach for crop recommendation system,” *International Journal of Intelligent Systems*, 2023.

[18] V. Patel et al., “Crop yield prediction using machine learning and deep learning techniques,” *Procedia Computer Science*, 2023.

[19] R. Kaur et al., “Crop recommendation system using data mining techniques,” *IEEE Conference*, 2022.

[20] V. Rao et al., “Crop yield prediction using machine learning techniques,” *IEEE Conference Proceedings*, 2020.