INTEL-UNNATI AI&ML JOURNAL
Number 2 (2024), pp. 1–22

Team:Strivers

MACHINE LEARNING

Knowledge Representation And Insights Generation From Structured Datasets

Lara Marium Jacob, Nandana A, Prathitha S, Lekshmipriya S, and Samudra S Sunil

Saintgits Group of Institutions, Kottayam, Kerala

Abstract: In contemporary agriculture, the effective utilization of structured datasets is pivotal for enhancing productivity and sustainability. Farmers help to feed a nation whose population is nearly 1.4 billion, however the productivity of farms is threatened by various natural factors that ruin the crops and farmer's livelihood. FarmGuard is a small initiative enhancing agriculture, making smart decisions to consider the demographics of the field, the factors affecting the crop, as well as how to keep the farm healthy."FarmGuard," is a website that focuses on leveraging machine learning to generate insights and facilitate decision-making in three critical areas: crop recommendation, fertilizer optimization, and plant disease detection. By representing knowledge extracted from soil quality assessments, climatic data, and historical crop performance, FarmGuard offers tailored recommendations to farmers. These insights empower agricultural stakeholders with actionable intelligence, enabling them to make informed decisions that optimize resource allocation and mitigate risks.

Keywords: Knowledge Representation, Insights Generation, Machine Learning, Agriculture, Crop Recommendation, Fertilizer Optimization, Plant Disease Detection

1 Introduction

In the realm of modern agriculture, the intersection of data science and agronomy has paved the way for significant advancements in productivity, sustainability, and decision-making. Leveraging structured datasets and machine learning techniques, agricultural practitioners can now extract actionable insights to optimize crop selection, enhance fertilizer application strategies, and detect plant diseases promptly. These advancements are

crucial as they not only improve yield outcomes but also contribute to resource efficiency and economic viability for farmers."FarmGuard" exemplifies this transformative approach by harnessing knowledge representation and insight generation from diverse datasets. By integrating information from soil quality assessments, climatic conditions, and historical crop performance, FarmGuard offers tailored recommendations that empower farmers to make informed decisions. These recommendations are instrumental in mitigating risks associated with crop management while promoting sustainable agricultural practices.It explores how knowledge representation through machine learning enables precise crop recommendations, optimal fertilizer usage, and early detection of plant diseases. Ultimately, these advancements contribute to fostering resilience and efficiency in agricultural systems, addressing global food security challenges and paving the way for a more sustainable future.

2 Libraries Used

In the project for various tasks, following packages are used.

```
NumPy
Pandas
Seaborn
plotly
imblearn
pickle
Matplotlib
Scikit-learn
```

3 Methodology

The objective of crop and fertilizer recommendation,and plant disease detection model is to recommend appropriate crop and fertilizer and detect disease which will be helpful for our farmers. Methodology of crop recommendation and fertilizer recommendation is as follows:

Data Loading: Load data for the Machine Learning task from Kaggle.

Pre-processing & Data cleaning: Clean and prepare the loaded data for Machine Learning algorithms. This involves handling missing values, outliers, and selecting relevant features. Crop Recommendation:Soil type, pH level, rainfall, temperature, and historical crop yield. Fertilizer Recommendation:Soil nutrient levels (nitrogen, phosphorus, potassium), pH, moisture content, and crop type.

Dataset Preparation: Split the dataset into training and testing sets

Classification: Implement various Machine Learning Classification models to recommend the most suitable crops and fertilizers for given soil and climate conditions. Models can include Decision Trees, Random Forest, Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), and Logistic Regression.

Model Training: Choose K-Nearest Neighbors (KNN) and train the model using the training set and extracted features like soil type, pH, and temperature.Choose Random Forest and train the model using the training set and features such as soil nutrients, crop type, and environmental conditions.

Model Evaluation: Test the trained models on the testing set to evaluate their performance.Use metrics such as accuracy and confusion matrices to assess effectiveness.

Model selection and reporting: Select the best model based on various performance measures.Report model performance metrics, highlighting the best models for crop (KNN) and fertilizer (Random Forest) recommendations.

Methodology of plant disease detection is as follows:

Data collection: Images: Collect a dataset of images showing healthy and diseased plants. Labels: Label each image with the corresponding disease type or healthy status.

Data Pre-processing: Resizing: Ensure all images are of the same size. Normalization: Normalize pixel values. Augmentation:Apply transformations (e.g., rotation, flipping, cropping) to increase dataset diversity.

Model Selection: Use CNN architectures suitable for image classification, such as Inception, DenseNet, or custom models designed for feature extraction from images. Train the model using a suitable deep learning framework like TensorFlow or PyTorch. Implement techniques like data augmentation (to generate additional training data), dropout (to prevent overfitting), and batch normalization (to speed up training and improve model performance).

Evaluation: Evaluate model performance using metrics such as accuracy,precision.

4 Implementation

A Implementation for crop recommendation

The primary goal of our machine learning project is to recommend suitable crops based on soil and climate conditions for agricultural purposes.The dataset consists of 2200 instances with 8 features, including soil pH, nitrogen content, temperature, rainfall, and humidity.

- Preprocessing Steps:Prior to training the models, we conducted several preprocessing steps:
  1. Normalization: Ensured uniform scale across different features.
  2. Handling Missing Data: Inputed missing values using technique
  3. Feature Engineering:Engineered additional features such as tempratrature,humidity,rainfall to enhance model performance.
- Machine Learning Algorithms Used:
  1. Logistic Regression:Logistic regression is suitable for binary classification tasks.Model accuracy achieved: 95.23
  2. Support Vector Machine (SVM):SVM finds the hyperplane that best separates different classes in the feature space. Used linear kernel for this project. Model accuracy achieved: 97.73
  3. K-Nearest Neighbors (KNN): KNN makes predictions based on the majority class among its k-nearest neighbors.Model accuracy achieved: 97.50
  4. Decision Tree:Decision trees recursively partition the feature space into distinct regions.Model accuracy achieved: 97.73
  5. Random Forest:Random Forest is an ensemble learning method that constructs multiple decision trees.Model accuracy achieved: 99.55

- Evaluation Metrics:To evaluate the performance of our models, we utilized the following metrics.
  1. Accuracy:Ratio of correctly predicted instances to the total instances.

Figure 1: EDA of crop recommendation

B Implementation for fertilizer recommendation

The primary objective of our machine learning project is to develop a model capable of recommending the most suitable fertilizer for crops based on soil and environmental conditions. We utilized a dataset containing soil properties, crop requirements, and environmental factors, obtained from kaggle. The dataset consists of 99 instances with 9 features, including soil pH, nitrogen levels, phosphorus levels, potassium levels, and crop type.

- Preprocessing Steps:Prior to training the models, we conducted several preprocessing steps:
  1. Normalization: We normalized the features to ensure a uniform scale across different features.
  2. Handling Missing Data:Since some fertilizer recommendations might be rarer than others, we employed the technique of oversampling to address class imbalance.
  3. Feature Engineering:We engineered additional features such as the ratio of nutrient levels, the interaction between soil properties and environmental conditions, and historical yield data to enhance model performance.
- Machine Learning Algorithms Used:
  1. Logistic Regression:Logistic Regression is a classic algorithm for binary classification tasks. It models the probability of a binary outcome based on one or more predictor variables.

2. Support Vector Machine (SVM):In SVM, the algorithm tries to find the hyperplane that best separates different classes in the feature space. This hyperplane is chosen so that the margin between the hyperplane and the nearest data point of any class is maximized. SVM can handle both linear and non-linear data by using different kernel functions like linear, polynomial, radial basis function (RBF), etc. Here we have used the linear method.
3. Decision Tree: Decision trees recursively partition the feature space into distinct regions, making decisions based on the values of input features.
4. Random Forest Classifier:Random Forest is an ensemble learning method that constructs a multitude of decision trees during training and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.
5. Gradient Boosting Classifier:Gradient Boosting is an ensemble technique that builds models sequentially. Each new model attempts to correct the errors made by the previous models.

- Evaluation Metrics:To evaluate the performance of our models, we utilized the following metrics.
  1. Accuracy:Ratio of correctly predicted instances to the total instances.

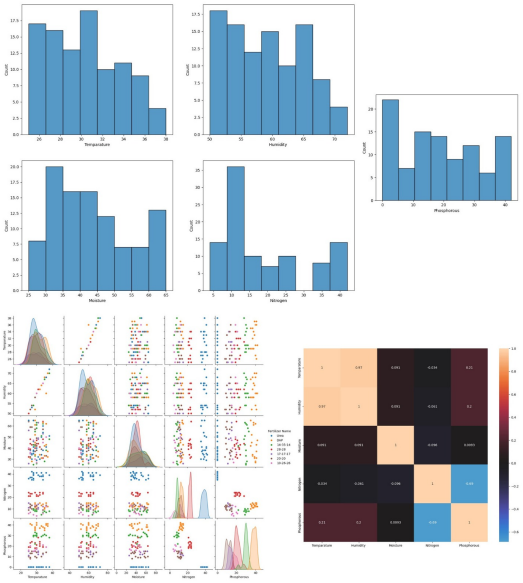


Figure 2: EDA of fertilizer recommendation

C Implementation for plant disease detection

The "New Plant Diseases Dataset" from Kaggle includes images of plant leaves affected by various diseases as well as healthy leaves. This dataset is crucial for developing models

that can automatically identify plant diseases from leaf images, which can help in early disease detection and management in agriculture.

1. Convolutional Neural Network (CNN) was designed to classify the images. The architecture included:
    - Multiple convolutional layers to extract features from the images.
    - Max-pooling layers to reduce the spatial dimensions and computational load.
    - Fully connected layers to perform the final classification.
    - The model included techniques like dropout to prevent overfitting.
2. Training the Model:The model was compiled using an appropriate optimizer and loss function. It was trained on the training dataset for a specified number of epochs, with its performance monitored on the validation dataset. During training, the model learned to identify different features in the images corresponding to healthy and diseased leaves.
3. Evaluation and Results:The model's performance was evaluated on the validation dataset, yielding metrics such as accuracy and loss.

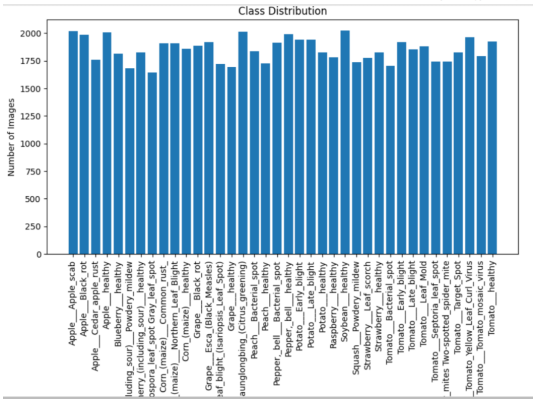
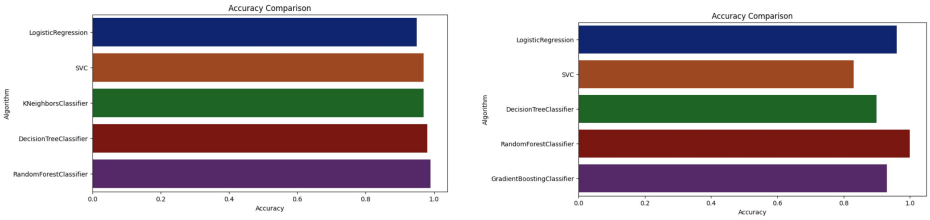


Figure 3: EDA of plant disease detection

D Results & Discussion

The accuracy comparison of various classification algorithms on the given dataset reveals insightful trends. Logistic Regression and Support Vector Classifier (SVC) consistently show high performance, serving as reliable baseline models with robust accuracy across different scenarios. The Decision Tree Classifier, while effective, demonstrates lower accuracy, highlighting its tendency to overfit without proper pruning. On the other hand, ensemble methods like Random Forest and Gradient Boosting Classifiers emerge as the top performers, achieving near-perfect accuracy. These methods benefit from combining multiple models to enhance robustness and accuracy, addressing the weaknesses of individual classifiers. The K-Neighbors Classifier, although slightly less accurate, offers simplicity and effectiveness, particularly in scenarios where interpretability and ease of implementation are prioritized. Overall, the results indicate that while Logistic Regression and SVC are

dependable choices, ensemble methods provide superior performance, making them ideal for complex classification tasks.Plant disease detection model has been trained using the deep learning model CNN and it has an accuracy of 0.93.



(a) Comparison of accuracy of crop recommendation

(b) Comparison of accuracy of fertilizer recommendation

E Conclusions

The evaluation of crop recommendation, fertilizer optimization, and plant disease detection in the "FarmGuard" project demonstrated effective performance using both classical machine learning algorithms and advanced deep learning techniques. Classical models like Random Forest, Support Vector Machines (SVM), and K-Nearest Neighbors (KNN) achieved comparable accuracy to deep learning models such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). These classical algorithms are advantageous for their efficiency in computation, making them suitable for real-time applications where quick decision-making is crucial. Deep learning models, while adept at capturing complex relationships, require more computational resources and longer training times. The integration of hardware accelerators like the Intel©AI accelerator has notably enhanced inference speed, benefiting both classical and deep learning approaches in agricultural decision support systems. Ultimately, the choice between these methodologies should consider factors such as computational efficiency, accuracy requirements, and real-time processing constraints.Looking ahead, the ongoing development and adoption of such technologies hold promise for addressing global food security challenges. By leveraging data-driven solutions, stakeholders in agriculture can collaborate more effectively, ensuring resilience and efficiency in agricultural systems.

In conclusion, the integration of knowledge representation and insight generation in projects like FarmGuard underscores their essential role in shaping the future of agriculture. As we confront challenges such as climate change and population growth, these technologies will play a crucial role in fostering innovation and securing food supplies for a growing global population.

Acknowledgments

We extend our sincere gratitude to Intel ©Corporation for providing us with the opportunity to embark on this project. We are deeply thankful to our mentor Er.Veena A Kumar for her invaluable guidance and unwavering support throughout every phase of our project development.Special thanks to Saintgits College of Engineering and Technology for equipping us with essential resources and facilitating insightful sessions on machine learning.

We also express our appreciation to all the researchers, scholars, and experts in the fields of machine learning, natural language processing, and artificial intelligence, whose pioneering contributions have laid the foundation for our project.Lastly, we acknowledge the mentors, institutional leaders, and industry guides whose expertise and encouragement through the Intel©- Unnati Programme have played a pivotal role in shaping our work and enriching our learning experience. []

References

[1] BENEDUZZI, H. M., SOUZA, E. G. D., MOREIRA, W. K., SOBJAK, R., BAZZI, C. L., AND RODRIGUES, M. Fertilizer recommendation methods for precision agriculture–a systematic literature study. *Engenharia Agrícola 42* (2022), e20210185.

[2] GOSAI, D., RAVAL, C., NAYAK, R., JAYSWAL, H., AND PATEL, A. Crop recommendation system using machine learning. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology 7*, 3 (2021), 558–569.

[3] LI, L., ZHANG, S., AND WANG, B. Plant disease detection and classification by deep learningâĂŤa review. *IEEE Access 9* (2021), 56683–56698.

[4] OLFS, H.-W., BLANKENAU, K., BRENTRUP, F., JASPER, J., LINK, A., AND LAMMEL, J. Soil-and plant-based nitrogen-fertilizer recommendations in arable farming. *Journal of Plant Nutrition and Soil Science 168*, 4 (2005), 414–431.

[5] PUDUMALAR, S., RAMANUJAM, E., RAJASHREE, R. H., KAVYA, C., KIRUTHIKA, T., AND NISHA, J. Crop recommendation system for precision agriculture. In *2016 eighth international conference on advanced computing (ICoAC)* (2017), IEEE, pp. 32–36.

A Main code sections for crop recommendation

A.1 Importing necessary libraries

```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
colorarr = ['#0592D0','#Cd7f32', '#E97451', '#Bdb76b', '#954535', '#C2b280', '#
                                        808000','#C2b280', '#E4d008', '#9acd32',
                                        '#Eedc82', '#E4d96f',
          '#32cd32','#39ff14','#00ff7f', '#008080', '#36454f', '#F88379', '#
                                                Ff4500', '#Ffb347', '#A94064',
                                                 '#E75480', '#Ffb6c1', '#
                                                E5e4e2',
          '#Faf0e6', '#8c92ac', '#Dbd7d2','#A7a6ba', '#B38b6d']
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.preprocessing import LabelEncoder
```

```
label_encoder = LabelEncoder()
```

A.2 Loading dataset

```
cropdf = pd.read_csv("crop_recommendation.csv")
cropdf.head()
cropdf.shape
cropdf.columns
print("Number of various crops: ", len(cropdf['label'].unique()))
print("List of crops: ", cropdf['label'].unique())
cropdf['label'].value_counts()
```

A.3 Python function for visualization

The general function for basic EDA of the data is created in python. The code for this task is given bellow:

```
crop_summary_N = crop_summary.sort_values(by='N', ascending=False)
fig = make_subplots(rows=1, cols=2)
top = {
    'y' : crop_summary_N['N'][0:10].sort_values().index,
    'x' : crop_summary_N['N'][0:10].sort_values()
}
last = {
    'y' : crop_summary_N['N'][-10:].index,
    'x' : crop_summary_N['N'][-10:]
}
fig.add_trace(
    go.Bar(top,
           name="Most nitrogen required",
           marker_color=random.choice(colorarr),
           orientation='h',
          text=top['x']),

    row=1, col=1
)
fig.add_trace(
    go.Bar(last,
           name="Least nitrogen required",
           marker_color=random.choice(colorarr),
           orientation='h',
          text=last['x']),
    row=1, col=2
)
fig.update_traces(texttemplate='%{text}', textposition='inside')
fig.update_layout(title_text="Nitrogen (N)",
                  plot_bgcolor='white',
                  font_size=12,
                  font_color='black',
                 height=500)

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
crop_summary_P = crop_summary.sort_values(by='P', ascending=False)
```

```
fig = make_subplots(rows=1, cols=2)
top = {
    'y' : crop_summary_P['P'][0:10].sort_values().index,
    'x' : crop_summary_P['P'][0:10].sort_values()
}
last = {
    'y' : crop_summary_P['P'][-10:].index,
    'x' : crop_summary_P['P'][-10:]
}
fig.add_trace(
    go.Bar(top,
           name="Most phosphorus required",
           marker_color=random.choice(colorarr),
           orientation='h',
          text=top['x']),

    row=1, col=1
)
fig.add_trace(
    go.Bar(last,
           name="Least phosphorus required",
           marker_color=random.choice(colorarr),
           orientation='h',
         text=last['x']),
    row=1, col=2
)
fig.update_traces(texttemplate='%{text}', textposition='inside')
fig.update_layout(title_text="Phosphorus (P)",
                  plot_bgcolor='white',
                  font_size=12,
                  font_color='black',
                 height=500)

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
crop_summary_K = crop_summary.sort_values(by='K', ascending=False)
fig = make_subplots(rows=1, cols=2)
top = {
    'y' : crop_summary_K['K'][0:10].sort_values().index,
    'x' : crop_summary_K['K'][0:10].sort_values()
}
last = {
    'y' : crop_summary_K['K'][-10:].index,
    'x' : crop_summary_K['K'][-10:]
}
fig.add_trace(
    go.Bar(top,
           name="Most potassium required",
           marker_color=random.choice(colorarr),
           orientation='h',
         text=top['x']),
    row=1, col=1
)
fig.add_trace(
    go.Bar(last,
           name="Least potassium required",
           marker_color=random.choice(colorarr),
           orientation='h',
```

```
          text=last['x']),
    row=1, col=2
)
fig.update_traces(texttemplate='%{text}', textposition='inside')
fig.update_layout(title_text="Potassium (K)",
                  plot_bgcolor='white',
                  font_size=12,
                  font_color='black',
                 height=500)

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
fig = go.Figure()
fig.add_trace(go.Bar(
    x=crop_summary.index,
    y=crop_summary['N'],
    name='Nitrogen',
    marker_color='indianred'
))
fig.add_trace(go.Bar(
    x=crop_summary.index,
    y=crop_summary['P'],
    name='Phosphorous',
    marker_color='lightsalmon'
))
fig.add_trace(go.Bar(
    x=crop_summary.index,
    y=crop_summary['K'],
    name='Potash',
    marker_color='crimson'
))
fig.update_layout(title="N, P, K values comparision between crops",
                  plot_bgcolor='white',
                  barmode='group',
                  xaxis_tickangle=-45)

fig.show()
labels = ['Nitrogen(N)','Phosphorous(P)','Potash(K)']
fig = make_subplots(rows=1, cols=5, specs=[[{'type':'domain'}, {'type':'domain'},
                                            {'type':'domain'}, {'type':'domain'},
                                            {'type':'domain'}]])
rice_npk = crop_summary[crop_summary.index=='rice']
values = [rice_npk['N'][0], rice_npk['P'][0], rice_npk['K'][0]]
fig.add_trace(go.Pie(labels=labels, values=values,name="Rice"),1, 1)
cotton_npk = crop_summary[crop_summary.index=='cotton']
values = [cotton_npk['N'][0], cotton_npk['P'][0], cotton_npk['K'][0]]
fig.add_trace(go.Pie(labels=labels, values=values,name="Cotton"),1, 2)
jute_npk = crop_summary[crop_summary.index=='jute']
values = [jute_npk['N'][0], jute_npk['P'][0], jute_npk['K'][0]]
fig.add_trace(go.Pie(labels=labels, values=values,name="Jute"),1, 3)
maize_npk = crop_summary[crop_summary.index=='maize']
values = [maize_npk['N'][0], maize_npk['P'][0], maize_npk['K'][0]]
fig.add_trace(go.Pie(labels=labels, values=values,name="Maize"),1, 4)
lentil_npk = crop_summary[crop_summary.index=='lentil']
values = [lentil_npk['N'][0], lentil_npk['P'][0], lentil_npk['K'][0]]
fig.add_trace(go.Pie(labels=labels, values=values,name="Lentil"),1, 5)
fig.update_traces(hole=.4, hoverinfo="label+percent+name")
fig.update_layout(
```

```
    title_text="NPK ratio for rice, cotton, jute, maize, lentil",
    annotations=[dict(text='Rice',x=0.06,y=0.8, font_size=15, showarrow=False),
                 dict(text='Cotton',x=0.26,y=0.8, font_size=15, showarrow=False),
                 dict(text='Jute',x=0.50,y=0.8, font_size=15, showarrow=False),
                 dict(text='Maize',x=0.74,y=0.8, font_size=15, showarrow=False),
                dict(text='Lentil',x=0.94,y=0.8, font_size=15, showarrow=False)])
fig.show()
labels = ['Nitrogen(N)','Phosphorous(P)','Potash(K)']
specs = [[{'type':'domain'}, {'type':'domain'}, {'type':'domain'}, {'type':'domain
                                        '}, {'type':'domain'}],[
         {'type':'domain'}, {'type':'domain'}, {'type':'domain'}, {'type':'domain'
                                            }, {'type':'domain'}]]
fig = make_subplots(rows=2, cols=5, specs=specs)
cafe_colors =  ['rgb(255, 128, 0)', 'rgb(0, 153, 204)', 'rgb(173, 173, 133)']
apple_npk = crop_summary[crop_summary.index=='apple']
values = [apple_npk['N'][0], apple_npk['P'][0], apple_npk['K'][0]]
fig.add_trace(go.Pie(labels=labels, values=values,name="Apple", marker_colors=
                                        cafe_colors),1, 1)
banana_npk = crop_summary[crop_summary.index=='banana']
values = [banana_npk['N'][0], banana_npk['P'][0], banana_npk['K'][0]]
fig.add_trace(go.Pie(labels=labels, values=values,name="Banana", marker_colors=
                                        cafe_colors),1, 2)
grapes_npk = crop_summary[crop_summary.index=='grapes']
values = [grapes_npk['N'][0], grapes_npk['P'][0], grapes_npk['K'][0]]
fig.add_trace(go.Pie(labels=labels, values=values,name="Grapes", marker_colors=
                                        cafe_colors),1, 3)
orange_npk = crop_summary[crop_summary.index=='orange']
values = [orange_npk['N'][0], orange_npk['P'][0], orange_npk['K'][0]]
fig.add_trace(go.Pie(labels=labels, values=values,name="Orange", marker_colors=
                                        cafe_colors),1, 4)
mango_npk = crop_summary[crop_summary.index=='mango']
values = [mango_npk['N'][0], mango_npk['P'][0], mango_npk['K'][0]]
fig.add_trace(go.Pie(labels=labels, values=values,name="Mango", marker_colors=
                                        cafe_colors),1, 5)
coconut_npk = crop_summary[crop_summary.index=='coconut']
values = [coconut_npk['N'][0], coconut_npk['P'][0], coconut_npk['K'][0]]
fig.add_trace(go.Pie(labels=labels, values=values,name="Coconut", marker_colors=
                                        cafe_colors),2, 1)
papaya_npk = crop_summary[crop_summary.index=='papaya']
values = [papaya_npk['N'][0], papaya_npk['P'][0], papaya_npk['K'][0]]
fig.add_trace(go.Pie(labels=labels, values=values,name="Papaya", marker_colors=
                                        cafe_colors),2, 2)
pomegranate_npk = crop_summary[crop_summary.index=='pomegranate']
values = [pomegranate_npk['N'][0], pomegranate_npk['P'][0], pomegranate_npk['K'][0
                                        ]]
fig.add_trace(go.Pie(labels=labels, values=values,name="Pomegranate",
                                        marker_colors=cafe_colors),2, 3)
watermelon_npk = crop_summary[crop_summary.index=='watermelon']
values = [watermelon_npk['N'][0], watermelon_npk['P'][0], watermelon_npk['K'][0]]
fig.add_trace(go.Pie(labels=labels, values=values,name="Watermelon", marker_colors
                                        =cafe_colors),2, 4)
muskmelon_npk = crop_summary[crop_summary.index=='muskmelon']
values = [muskmelon_npk['N'][0], muskmelon_npk['P'][0], muskmelon_npk['K'][0]]
fig.add_trace(go.Pie(labels=labels, values=values,name="Muskmelon", marker_colors=
                                        cafe_colors),2, 5)
fig.update_layout(
    title_text="NPK ratio for fruits",
    annotations=[dict(text='Apple',x=0.06,y=1.08, font_size=15, showarrow=False),
                 dict(text='Banana',x=0.26,y=1.08, font_size=15, showarrow=False),
```

```
                dict(text='Grapes',x=0.50,y=1.08, font_size=15, showarrow=False),
                dict(text='Orange',x=0.74,y=1.08, font_size=15, showarrow=False),
               dict(text='Mango',x=0.94,y=1.08, font_size=15, showarrow=False),
               dict(text='Coconut',x=0.06,y=0.46, font_size=15, showarrow=False),
                dict(text='Papaya',x=0.26,y=0.46, font_size=15, showarrow=False),
                dict(text='Pomegranate',x=0.50,y=0.46, font_size=15, showarrow=
                                                        False),
                dict(text='Watermelon',x=0.74,y=0.46, font_size=15, showarrow=
                                                        False),
               dict(text='Muskmelon',x=0.94,y=0.46, font_size=15, showarrow=False
                                                       )])
fig.show()
crop_scatter = cropdf[(cropdf['label']=='rice') |
                      (cropdf['label']=='jute') |
                      (cropdf['label']=='cotton') |
                     (cropdf['label']=='maize') |
                     (cropdf['label']=='lentil')]
fig = px.scatter(crop_scatter, x="temperature", y="humidity", color="label",
                                        symbol="label")
fig.update_layout(plot_bgcolor='white')
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
fig = px.bar(crop_summary, x=crop_summary.index, y=["rainfall", "temperature", "
                                        humidity"])
fig.update_layout(title_text="Comparision between rainfall, temerature and
                                        humidity",
                  plot_bgcolor='white',
                 height=500)
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
numeric_df = cropdf.select_dtypes(include=np.number)
fig, ax = plt.subplots(1, 1, figsize=(15, 9))
sns.heatmap(numeric_df.corr(), annot=True, cmap='Wistia')
ax.set(xlabel='Features', ylabel='Features')
plt.show()
features=cropdf[['N','P','K','temperature','humidity','ph','rainfall']]
target=cropdf['label']
labels=cropdf['label']
```

A.4 Test & train splitting

Splitting the dataset into training and testing sets.It helps to avoid overfitting and to accurately evaluate your model.

```
Xtrain,Xtest,Ytrain,Ytest = train_test_split(features,target_encoded,test_size=0.2
                                        ,random_state=2)
```

A.5 Loading models and evaluvation metrices

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.tree import DecisionTreeClassifier, ExtraTreeClassifier
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier,
                                        GradientBoostingClassifier,
                                        AdaBoostClassifier
from sklearn.metrics import accuracy_score
```

```
models = {
    'LogisticRegression': LogisticRegression(),
    'SVC':SVC(),
    'KNeighborsClassifier':KNeighborsClassifier(),
    'DecisionTreeClassifier':DecisionTreeClassifier(),
    'RandomForestClassifier':RandomForestClassifier(),
     }
```

A.6 Training models

```
for name, model in models.items():
    model.fit(Xtrain, Ytrain)
    y_pred = model.predict(Xtest)
    score = accuracy_score(Ytest, y_pred)
    print(f"{name} model with accuracy: {score}")
```

A.7 Comparison of accuracy models

```
import matplotlib.pyplot as plt
import seaborn as sns
acc = [0.95, 0.97, 0.97, 0.98, 0.99]
model = ['LogisticRegression', 'SVC', 'KNeighborsClassifier', '
                                        DecisionTreeClassifier', '
                                        RandomForestClassifier']
print(len(acc), len(model))
plt.figure(figsize=[10, 5], dpi=100)
plt.title('Accuracy Comparison')
plt.xlabel('Accuracy')
plt.ylabel('Algorithm')
sns.barplot(x=acc, y=model, palette='dark')
plt.show()
```

B Main code for fertilizer recommendation

B.1 importing necessary libraries

```
import pandas as pd
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,accuracy_score
                                        ,precision_score
from sklearn import metrics
from sklearn.svm import SVC
```

```
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import imblearn
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.neighbors import  KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
import xgboost
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, ConfusionMatrixDisplay,
                                        confusion_matrix
import pickle
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

B.2 Loading dataset

```
data = pd.read_csv("Fertilizer Prediction.csv")
data.head()
```

B.3 EDA

```
data["Fertilizer Name"].unique()
data.shape
data["Soil Type"].unique()
data["Crop Type"].unique()
data.columns
labels = data["Fertilizer Name"].unique()
counts = list(data["Fertilizer Name"].value_counts())
plt.figure(figsize = (9,5))
plt.barh(labels, counts)
for index, value in enumerate(counts):
    plt.text(value, index,
             str(value))
plt.show()
continuous_data_cols = ["Temparature", "Humidity ", "Moisture", "Nitrogen", "
                                        Phosphorous"]
categorical_data_cols = ["Soil Type", "Crop Type"]
plt.figure(figsize=(15,13))
i = 1
for column in continuous_data_cols[:-1]:
    plt.subplot(2,2,i)
    sns.histplot(data[column])
```

```
    i+=1
plt.show()
sns.histplot(data[continuous_data_cols[-1]])
plt.show()
plt.figure(figsize=(15,13))
i = 1
for column in continuous_data_cols[:-1]:
    plt.subplot(2,2,i)
    sns.histplot(data[column])
    i+=1
plt.show()

sns.histplot(data[continuous_data_cols[-1]])
plt.show()
plt.figure(figsize=(17,5))
i = 1
for column in categorical_data_cols:
    plt.subplot(1,2,i)
    sns.countplot(data[column])
    plt.xticks(rotation = 90)
    i+=1
plt.show()
plt.figure(figsize=(21,17))
sns.pairplot(data[continuous_data_cols + ["Fertilizer Name"]], hue = "Fertilizer
                                        Name")
plt.show()
plt.figure(figsize = (13,11))
sns.heatmap(data[continuous_data_cols].corr(), center = 0, annot = True)
plt.show()
soil_type_label_encoder = LabelEncoder()
data["Soil Type"] = soil_type_label_encoder.fit_transform(data["Soil Type"])
crop_type_label_encoder = LabelEncoder()
data["Crop Type"] = crop_type_label_encoder.fit_transform(data["Crop Type"])
croptype_dict = {}
for i in range(len(data["Crop Type"].unique())):
    croptype_dict[i] = crop_type_label_encoder.inverse_transform([i])[0]
print(croptype_dict)
soiltype_dict = {}
for i in range(len(data["Soil Type"].unique())):
    soiltype_dict[i] = soil_type_label_encoder.inverse_transform([i])[0]
print(soiltype_dict)
fertname_label_encoder = LabelEncoder()
data["Fertilizer Name"] = fertname_label_encoder.fit_transform(data["Fertilizer
                                        Name"])
fertname_dict = {}
for i in range(len(data["Fertilizer Name"].unique())):
    fertname_dict[i] = fertname_label_encoder.inverse_transform([i])[0]
print(fertname_dict)
X = data[data.columns[:-1]]
y = data[data.columns[-1]]
counter = Counter(y)
counter
print(f"Total Data after Upsampling: {len(X)}")
```

B.4 Train test split

```
X_train, X_test, y_train, y_test = train_test_split(X.values, y, test_size = 0.3,
                                                    random_state = 0)
print(f"Train Data: {X_train.shape}, {y_train.shape}")
print(f"Train Data: {X_test.shape}, {y_test.shape}")
```

```
  from sklearn.preprocessing import MinMaxScaler
  mx = MinMaxScaler()
  X_train = mx.fit_transform(X_train)
  X_test = mx.transform(X_test)
  from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train = sc.transform(X_train)
X_test=sc.transform(X_test)
```

B.5 loading model and evaluation metrices

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier, ExtraTreeClassifier
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier,
                                          GradientBoostingClassifier,
                                          AdaBoostClassifier
from sklearn.metrics import accuracy_score
```

```
models = {
    'LogisticRegression': LogisticRegression(),
    'SVC':SVC(),
    'DecisionTreeClassifier':DecisionTreeClassifier(),
    'RandomForestClassifier':RandomForestClassifier(),
    'GradientBoostingClassifier':GradientBoostingClassifier()
}
```

B.6 Model training

```
 for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    print(f"{name} model with accuracy: {score}")
```

B.7 Comparison of model accuracies

```
import matplotlib.pyplot as plt
import seaborn as sns
acc = [0.96,0.83,0.9,1.0,0.93]
model = ['LogisticRegression', 'SVC', 'DecisionTreeClassifier', '
                                          RandomForestClassifier','
                                          GradientBoostingClassifier']
```

```
print(len(acc), len(model))
plt.figure(figsize=[10, 5], dpi=100)
plt.title('Accuracy Comparison')
plt.xlabel('Accuracy')
plt.ylabel('Algorithm')
sns.barplot(x=acc, y=model, palette='dark')
plt.show()
```

C Main code for plant disease detection

C.1 importing necessary libraries

```
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.metrics import
import numpy as np
import matplotlib.pyplot as plt
```

C.2 Data Preprocessing

Training Image preprocessing

```
training_set = tf.keras.utils.image_dataset_from_directory(
    'train',
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)
```

Validation Image Preprocessing

```
validation_set = tf.keras.utils.image_dataset_from_directory(
    'valid',
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=True,
    seed=None,
    validation_split=None,
```

```
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)
```

C.3 Building Model

```
cnn = tf.keras.models.Sequential()
```

Building Convolution Layer

```
cnn.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,padding='same',activation=
                                        'relu',input_shape=[128,128,3]))
cnn.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

```
cnn.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,padding='same',activation=
                                        'relu'))
cnn.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

```
cnn.add(tf.keras.layers.Conv2D(filters=128,kernel_size=3,padding='same',activation
                                        ='relu'))
cnn.add(tf.keras.layers.Conv2D(filters=128,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

```
cnn.add(tf.keras.layers.Conv2D(filters=256,kernel_size=3,padding='same',activation
                                        ='relu'))
cnn.add(tf.keras.layers.Conv2D(filters=256,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

```
cnn.add(tf.keras.layers.Conv2D(filters=512,kernel_size=3,padding='same',activation
                                        ='relu'))
cnn.add(tf.keras.layers.Conv2D(filters=512,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

```
cnn.add(tf.keras.layers.Dropout(0.25))
```

```
cnn.add(tf.keras.layers.Flatten())
```

```
cnn.add(tf.keras.layers.Dense(units=1500,activation='relu'))
```

```
cnn.add(tf.keras.layers.Dropout(0.4))
```

```
cnn.add(tf.keras.layers.Dense(units=38,activation='softmax'))
```

C.4 Compiling and Training Phase

```
cnn.compile(optimizer=tf.keras.optimizers.legacy.Adam(   learning_rate=0.0001),
                                        loss='categorical_crossentropy',metrics=[
                                        'accuracy'])
```

```
cnn.summary()
```

```
training_history = cnn.fit(x=training_set,validation_data=validation_set,epochs=10
                                        )
```

C.5 Evaluating Model

```
train_loss, train_acc = cnn.evaluate(training_set)
print('Training accuracy:', train_acc)
```

```
val_loss, val_acc = cnn.evaluate(validation_set)
print('Validation accuracy:', val_acc)
```

C.6 Saving Model

```
cnn.save('trained_plant_disease_model.keras')
```

```
training_history.history
```

```
import json
with open('training_hist.json','w') as f:
  json.dump(training_history.history,f)
```

```
print(training_history.history.keys())
```

C.7 Accuracy Visualization

```
epochs = [i for i in range(1,11)]
plt.plot(epochs,training_history.history['accuracy'],color='red',label='Training
                                        Accuracy')
plt.plot(epochs,training_history.history['val_accuracy'],color='blue',label='
                                        Validation Accuracy')
plt.xlabel('No. of Epochs')
plt.title('Visualization of Accuracy Result')
plt.legend()
plt.show()
```

C.8 model evaluation

```
test_set = tf.keras.utils.image_dataset_from_directory(
    'valid',
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=1,
    image_size=(128, 128),
    shuffle=False,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)
```

```
y_pred = cnn.predict(test_set)
predicted_categories = tf.argmax(y_pred, axis=1)
```

```
true_categories = tf.concat([y for x, y in test_set], axis=0)
Y_true = tf.argmax(true_categories, axis=1)
Y_true
predicted_categories
```

```
confusion_matrix,classification_report
cm = confusion_matrix(Y_true,predicted_categories)
print(classification_report(Y_true,predicted_categories,target_names=class_name))
```

Confusion Matrix Visualization

```
plt.figure(figsize=(40, 40))
sns.heatmap(cm,annot=True,annot_kws={"size": 10})
plt.xlabel('Predicted Class',fontsize = 20)
plt.ylabel('Actual Class',fontsize = 20)
plt.title('Plant Disease Prediction Confusion Matrix',fontsize = 25)
plt.show()
```

C.9 EDA

```
class_names = training_set.class_names
num_classes = len(class_names)
print(f"Number of classes: {num_classes}")
print(f"Class names: {class_names}")
num_images = sum(1 for _ in training_set.unbatch())
print(f"Number of images:{num_images}")
class_counts = {class_name: 0 for class_name in class_names}
for _, labels in training_set.unbatch():
    class_index = np.argmax(labels)
    class_name = class_names[class_index]
    class_counts[class_name] += 1
plt.figure(figsize=(10, 5))
plt.bar(class_counts.keys(), class_counts.values())
```

```
plt.title("Class Distribution")
plt.xlabel("Class")
plt.ylabel("Number of Images")
plt.xticks(rotation=90)
plt.show()
plt.figure(figsize=(10, 10))
for images, labels in training_set.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[np.argmax(labels[i])])
        plt.axis("off")
plt.show()
image_shapes = [image.shape for image, _ in training_set.unbatch()]
image_shapes = np.array(image_shapes)
unique_shapes, shape_counts = np.unique(image_shapes, axis=0, return_counts=True)
print("Unique image shapes and their counts:")
for shape, count in zip(unique_shapes, shape_counts):
    print(f"{shape}: {count}")
aspect_ratios = image_shapes[:, 1] / image_shapes[:, 0]
plt.figure(figsize=(10, 5))
plt.hist(aspect_ratios, bins=20, color='blue', edgecolor='black')
plt.title("Aspect Ratio Distribution")
plt.xlabel("Aspect Ratio (Width / Height)")
plt.ylabel("Frequency")
plt.show()
def plot_color_distribution(images, title):
    color_channels = ('Red', 'Green', 'Blue')
    colors = np.stack([images[:, :, :, i] for i in range(3)], axis=-1).mean(axis=(
                                            0, 1, 2))

    plt.figure(figsize=(10, 5))
    plt.bar(color_channels, colors, color=['r', 'g', 'b'])
    plt.title(title)
    plt.ylabel("Mean Pixel Value")
    plt.show()

for images, _ in training_set.take(1):
    images = images.numpy().astype("float32") / 255.0
    plot_color_distribution(images, "Color Distribution in Sample Images")
    print("Class Balance Check:")
for class_name, count in class_counts.items():
    print(f"{class_name}: {count} images")
```