Machine Learning and Applications

UE20EC352

Domain : Agriculture

1. Crop Identification
2. Soil Identification
3. Plant Disease Detection

Team Members:
Dhyey Udeshi - PES1UG20EC062
Dheeraj M - PES1UG20EC060
Ankush Gupta - PES1UG20EC031

- Motivation for the domain and the problem

Agriculture is a crucial sector that provides food, fiber, and other essential resources for the world's population. As the global population continues to grow, the demand for agricultural products increases, making it a vital sector for economic development and sustainability. However, agriculture faces many challenges, including climate change, water scarcity, soil degradation, pest and disease outbreaks, and low productivity.

These challenges highlight the need for innovative and sustainable agricultural practices that can increase productivity while minimizing the impact on the environment

The motivation for the domain and the problem for agriculture is to develop and implement innovative and sustainable agricultural practices that can address the challenges facing the sector.

Dataset Description (For D1 dataset)

- Dataset consists of two features, Latitude and Longitude.
- It is a binary classification problem. The two classes are Rice fields and Non Rice fields.
- The model was split into training and testing in the ratio of 8:2.
- We have used evaluation metrics, ROC curve, confusion matrix to evaluate the model.

ML algorithm used for D1.

- Machine learning algorithm used for this dataset is Naive Bayes algorithm. The Machine Algorithm used for Crop Identification and Soil Identification is Naive Bayes Classifier. It is based on Bayes Theorem.
- Bayes Theorem is formulated as shown $P(A|B)=\frac{P(B|A)P(A)}{P(B)}$
- Where P(A|B) is Posterior Probability, P(B|A) is likelihood probability, P(A) is prior probability and P(B) is Marginal probability.

Python code for D1

```
pip install scikit-optimize
pip install gdown
!gdown "1SQOwKCJu9nmd0U3yq6hIUv5K51OP8gr4"
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from     sklearn.metrics     import     accuracy_score,     confusion_matrix,
classification_report, roc_curve, auc
import matplotlib.pyplot as plt
# Load the data from CSV file
data = pd.read_csv("Crop_Location_Data_20221201_seperate.csv")
# Map land class to numerical values
class_map = {'Rice': 0, 'Non Rice': 1}
data['Class of Land'] = data['Class of Land'].map(class_map)
# Separate features and target variable
X = data[['Latitude', 'Longitude']].values
y = data['Class of Land'].values
# Split the data into train and test sets
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,  y,  test_size=0.2,
random_state=42)
# Train a Naive Bayes classifier
nb_clf = GaussianNB()
nb_clf.fit(X_train, y_train)
# Make predictions on test data
y_pred = nb_clf.predict(X_test)
# Calculate accuracy and print classification report
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print(classification_report(y_test, y_pred))
# Plot the classification
plt.scatter(X_test[:, 1], X_test[:, 0], c=y_pred, cmap='viridis')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Naive Bayes Classification')
plt.show()
# Calculate and plot ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)'
% roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt
# Read data from CSV file
data = pd.read_csv("Crop_Location_Data_20221201_seperate.csv")
# Map land class to numerical values
class_map = {'Rice': 0, 'Non Rice': 1}
data['Class of Land'] = data['Class of Land'].map(class_map)
# Separate features and target variable
X = data[['Latitude', 'Longitude']].values
y = data['Class of Land'].values
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Train a Naive Bayes classifier
nb_clf = GaussianNB()
nb_clf.fit(X_train, y_train)
# Predict class labels for test data
y_pred = nb_clf.predict(X_test)
# Plot the decision boundary
x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                     np.linspace(y_min, y_max, 100))
Z = nb_clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.4, cmap='viridis')
plt.colorbar()
# Plot the test data points
colors = ['blue' if label == 0 else 'red' for label in y_test]
plt.scatter(X_test[:, 0], X_test[:, 1], c=colors, s=50, edgecolors='k')
# Add labels and title
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.title('Naive Bayes Classification')
# Show the plot
```

```
plt.show()
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
# Read data from CSV file
data = pd.read_csv("Crop_Location_Data_20221201_seperate.csv")
# Map land class to numerical values
class_map = {'Rice': 0, 'Non Rice': 1}
data['Class of Land'] = data['Class of Land'].map(class_map)
# Separate features and target variable
X = data[['Latitude', 'Longitude']].values
y = data['Class of Land'].values
# Split the data into train and test sets
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,  y,  test_size=0.2,
random_state=42)
# Train a Naive Bayes classifier
nb_clf = GaussianNB()
nb_clf.fit(X_train, y_train)
# Predict class labels for test data
y_pred = nb_clf.predict(X_test)
# Print the confusion matrix
print(confusion_matrix(y_test, y_pred))
# Print the classification report
print(classification_report(y_test, y_pred))
# Plot the decision boundary
x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                     np.linspace(y_min, y_max, 100))
Z = nb_clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.4, cmap='viridis')
plt.colorbar()
# Plot the test data points
colors = ['blue' if label == 0 else 'red' for label in y_test]
plt.scatter(X_test[:, 0], X_test[:, 1], c=colors, s=50, edgecolors='k')
# Add labels and title
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.title('Naive Bayes Classification')
```

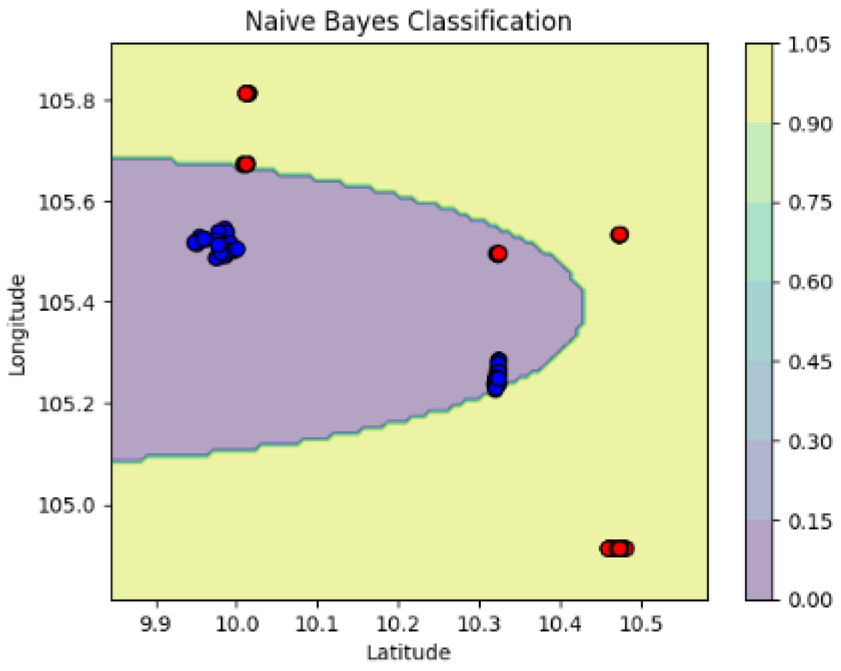
```
# Show the plot
plt.show()
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
# Read data from CSV file
data = pd.read_csv("Crop_Location_Data_20221201_seperate.csv")
# Map land class to numerical values
class_map = {'Rice': 0, 'Non Rice': 1}
data['Class of Land'] = data['Class of Land'].map(class_map)
# Separate features and target variable
X = data[['Latitude', 'Longitude']].values
y = data['Class of Land'].values
# Split the data into train and test sets
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,  y,  test_size=0.2,
random_state=42)
# Define the parameter grid to search over
param_grid = {'var_smoothing': np.logspace(0,-9, num=100)}
#  Train  a  Naive  Bayes  classifier  with  GridSearchCV  to  find  the  best
hyperparameters
nb_clf = GaussianNB()
grid_search  =  GridSearchCV(estimator=nb_clf,  param_grid=param_grid,  cv=5,
verbose=1, n_jobs=-1)
grid_search.fit(X_train, y_train)
# Print the best hyperparameters and the corresponding score
print("Best Hyperparameters: ", grid_search.best_params_)
print("Best Score: ", grid_search.best_score_)
# Predict class labels for test data using the best hyperparameters
y_pred = grid_search.predict(X_test)

# Print the confusion matrix
print(confusion_matrix(y_test, y_pred))
# Print the classification report
print(classification_report(y_test, y_pred))
# Plot the decision boundary
x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                     np.linspace(y_min, y_max, 100))
Z = grid_search.predict(np.c_[xx.ravel(), yy.ravel()])
```

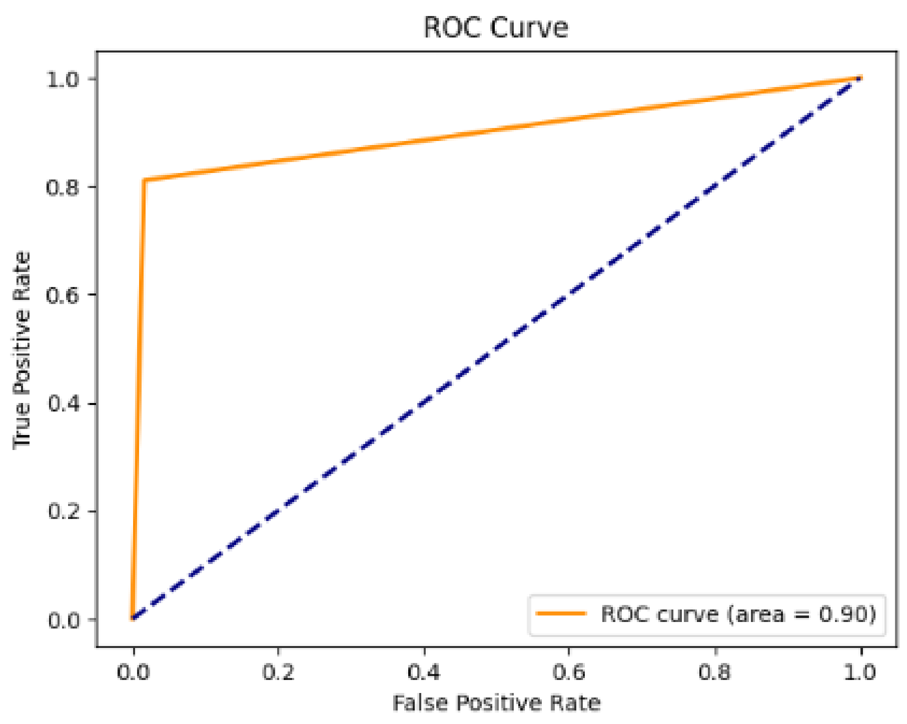
```
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.4, cmap='viridis')
plt.colorbar()
# Plot the test data points
colors = ['blue' if label == 0 else 'red' for label in y_test]
plt.scatter(X_test[:, 0], X_test[:, 1], c=colors, s=50, edgecolors='k')
# Add labels and title
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.title('Naive Bayes Classification')
# Show the plot
plt.show()
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.naive_bayes import GaussianNB
from     sklearn.metrics     import     accuracy_score,     confusion_matrix,
classification_report, roc_curve, auc
import matplotlib.pyplot as plt
import numpy as np
# Read data from CSV file
data = pd.read_csv("Crop_Location_Data_20221201_seperate.csv")
# Map land class to numerical values
class_map = {'Rice': 0, 'Non Rice': 1}
data['Class of Land'] = data['Class of Land'].map(class_map)
# Separate features and target variable
X = data[['Latitude', 'Longitude']].values
y = data['Class of Land'].values
# Split the data into train and test sets
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,  y,  test_size=0.2,
random_state=42)
# Define the parameter grid to search over
param_grid = {'var_smoothing': np.logspace(0,-9, num=100)}

#  Train  a  Naive  Bayes  classifier  with  GridSearchCV  to  find  the  best
hyperparameters
nb_clf = GaussianNB()
grid_search  =  GridSearchCV(estimator=nb_clf,  param_grid=param_grid,  cv=5,
verbose=1, n_jobs=-1)
grid_search.fit(X_train, y_train)
# Print the best hyperparameters and the corresponding score
print("Best Hyperparameters: ", grid_search.best_params_)
print("Best Score: ", grid_search.best_score_)
# Predict class labels for test data using the best hyperparameters
```

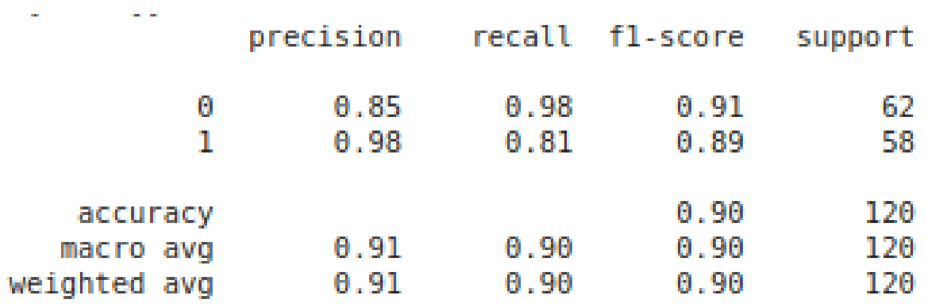
```
y_pred = grid_search.predict(X_test)
# Print the confusion matrix
print(confusion_matrix(y_test, y_pred))
# Print the classification report
print(classification_report(y_test, y_pred))
# Plot the decision boundary
x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                    np.linspace(y_min, y_max, 100))
Z = grid_search.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.4, cmap='viridis')
plt.colorbar()
# Plot the test data points
colors = ['blue' if label == 0 else 'red' for label in y_test]
plt.scatter(X_test[:, 0], X_test[:, 1], c=colors, s=50, edgecolors='k')
# Add labels and title
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.title('Naive Bayes Classification')
# Show the plot
plt.show()
# Calculate and plot ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)'
% roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()

import seaborn as sns
import matplotlib.pyplot as plt
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
# Create heatmap
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
from sklearn.metrics import accuracy_score, f1_score, recall_score,
precision_score
# Calculate and print accuracy, F1 score, recall, and precision
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
print("Accuracy: ", accuracy)
print("F1 Score: ", f1)
print("Recall: ", recall)
print("Precision: ", precision)
```

Experimental Results and Performance metrics for D1

- **Before tuning the hyperparameters:**

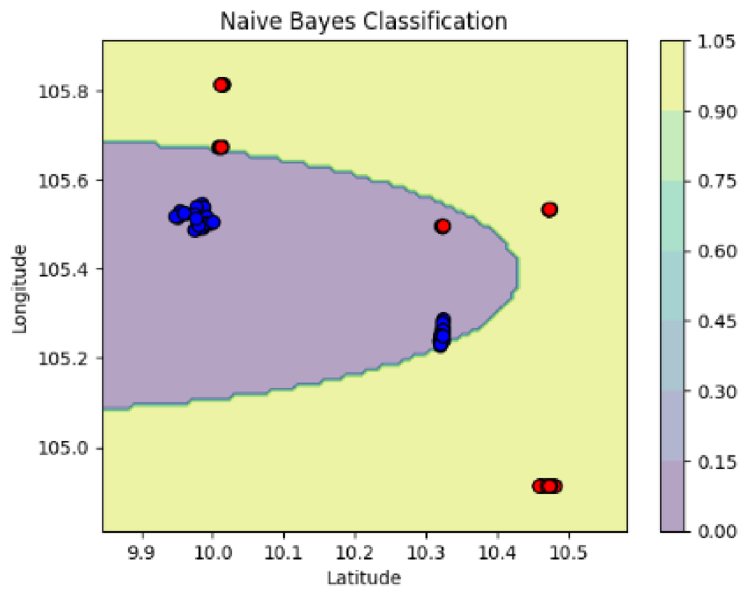


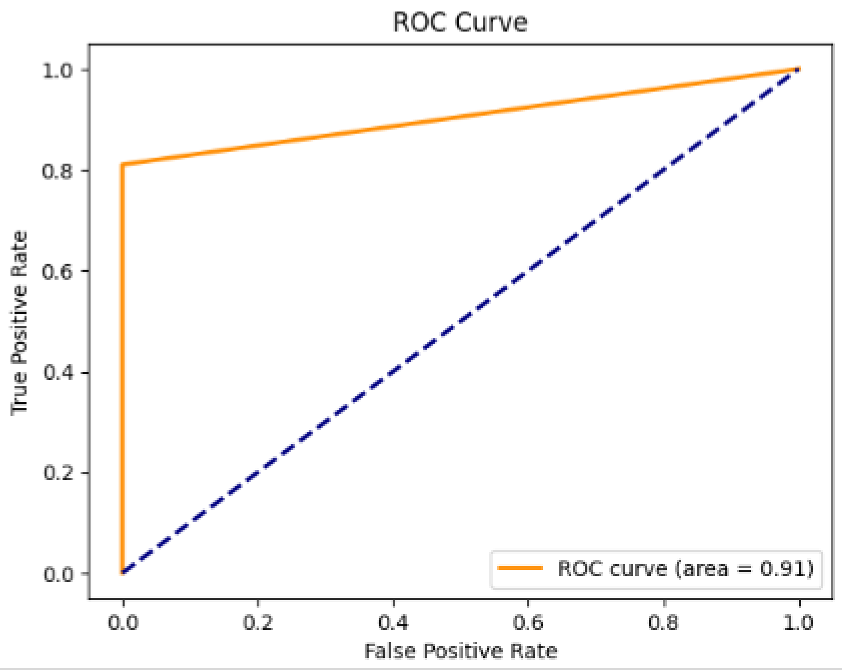


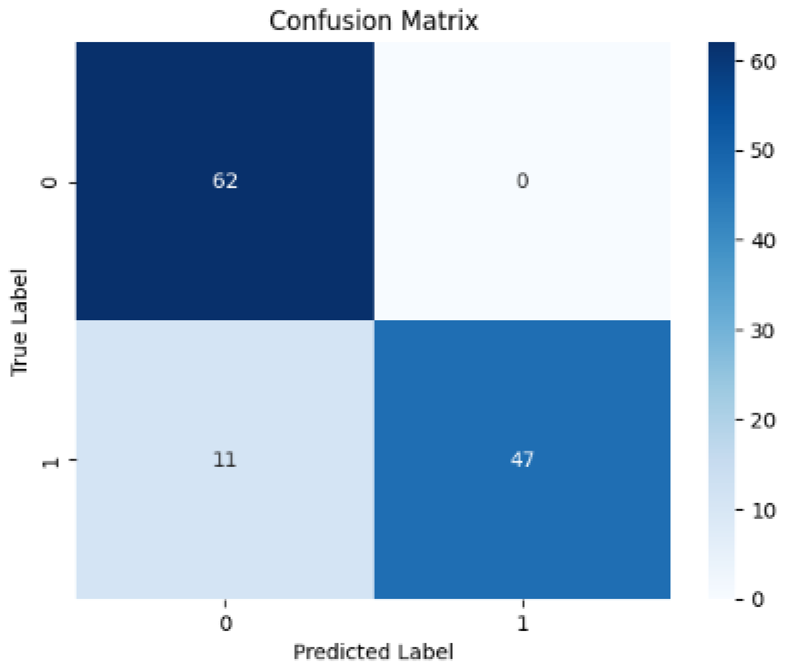


```
              precision    recall  f1-score   support

           0       0.85      0.98      0.91        62
           1       0.98      0.81      0.89        58

    accuracy                           0.90       120
   macro avg       0.91      0.90      0.90       120
weighted avg       0.91      0.90      0.90       120
```

- **After tuning the hyperparameters:**







```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92        62
           1       1.00      0.81      0.90        58

    accuracy                           0.91       120
   macro avg       0.92      0.91      0.91       120
weighted avg       0.92      0.91      0.91       120
```

Observations, inferences, Conclusion for D1

The accuracy of the testing set for Naive Bayes algorithm was found as 90%. After hypertuning the parameters, the accuracy increased to 91%. The dataset was converted to a higher dimension for classification.

Hence we can conclude that this algorithm can be used to classify spatial data, i.e., crop fields for this example.

Dataset Description (For D2 dataset)

- Dataset consists of multiple features such as temperature,humidity,pH value of the soil,N,P and k values
- It is a multi class classification problem. The classes are different types of crops which are suitable for growth eg apple,banana,grape,chickpea,coffee etc
- The model was split into training and testing in the ratio of 8:2.
- We have used evaluation metrics, ROC curve, confusion matrix to evaluate the model.

ML algorithm used for D2.

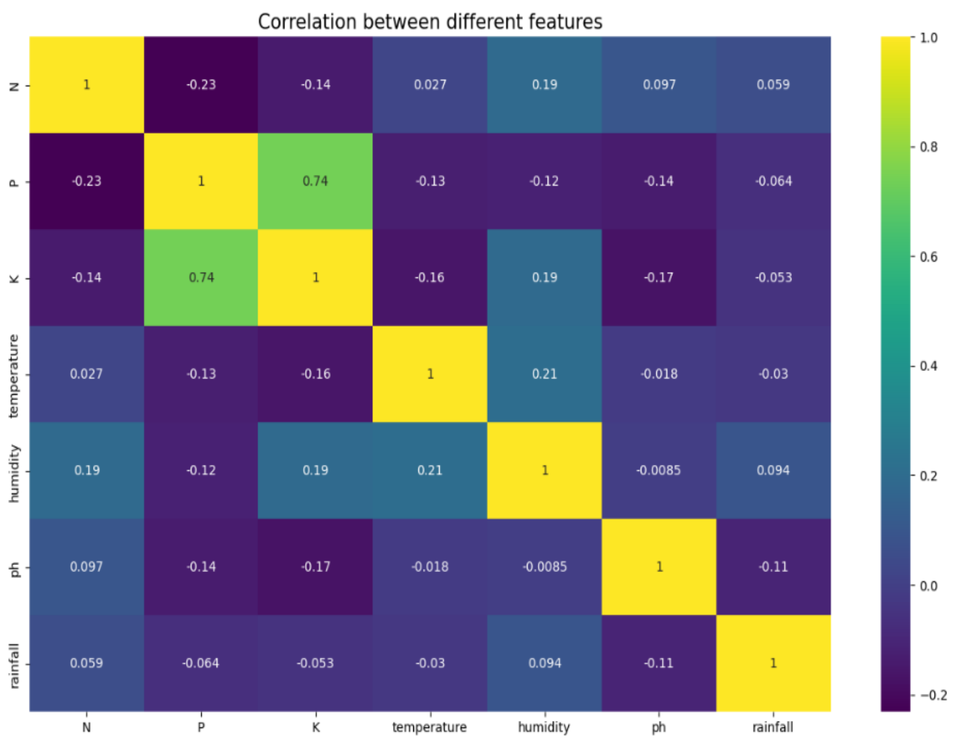
- Machine learning algorithm used for this dataset is Naive Bayes algorithm. The Machine Algorithm used for Crop Identification and Soil Identification is Naive Bayes Classifier. It is based on Bayes Theorem.
- Bayes Theorem is formulated as shown $P(A|B)=\frac{P(B|A)P(A)}{P(B)}$
- Where P(A|B) is Posterior Probability, P(B|A) is likelihood probability, P(A) is prior probability and P(B) is Marginal probability.

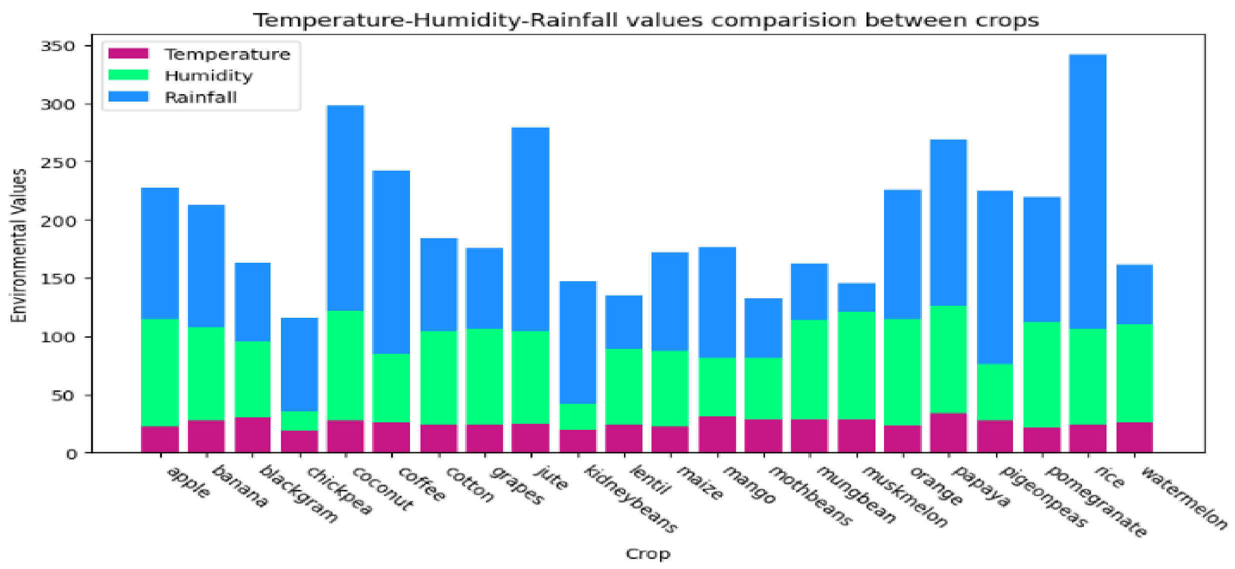
Python code for D2 :

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn import tree
from sklearn.model_selection import cross_val_score
from google.colab import drive
# import the necessary libraries
import pandas as pd
!gdown --id 1T8z08oI2alEcINPNLQwEG7_Lb7VsfXv8
# Enter the path to the CSV file in your Google Drive
file_path = '/content/Crop_recommendation.csv'
# Use pandas to read in the CSV file from the file path
crop = pd.read_csv(file_path)
# Display the first 5 rows of the DataFrame
crop.head(5)
crop.isnull().sum()
crop.info()
crop.describe()
crop.columns
crop.shape
crop['label'].unique()
crop['label'].nunique()
crop['label'].value_counts()
fig, ax = plt.subplots(1, 1, figsize=(15, 9))
sns.heatmap(crop.corr(), annot=True,cmap='viridis')
plt.title('Correlation between different features', fontsize = 15, c='black')
plt.show()
crop_summary = pd.pivot_table(crop,index=['label'],aggfunc='mean')
crop_summary.head()
x = crop_summary.index
y1 = crop_summary['N']
y2 = crop_summary['P']
y3 = crop_summary['K']
color1 = 'mediumvioletred'
color2 = 'springgreen'
color3 = 'dodgerblue'
fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(x, y1, color=color1, label='Nitrogen')
ax.bar(x, y2, color=color2, bottom=y1, label='Phosphorous')
ax.bar(x, y3, color=color3, bottom=y1+y2, label='Potash')
ax.set_title("N-P-K values comparision between crops")
ax.set_xlabel("Crop")
ax.set_ylabel("Nutrient Value")
plt.xticks(rotation=-45, ha='left', va='top')
ax.legend()
plt.subplots_adjust(bottom=0.2)
plt.show()
x = crop_summary.index
y1 = crop_summary['temperature']
y2 = crop_summary['humidity']
y3 = crop_summary['rainfall']
fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(x, y1, color=color1, label='Temperature')
ax.bar(x, y2, color=color2, bottom=y1, label='Humidity')
ax.bar(x, y3, color=color3, bottom=y1+y2, label='Rainfall')
ax.set_title("Temperature-Humidity-Rainfall   values   comparision   between
crops")
ax.set_xlabel("Crop")
ax.set_ylabel("Environmental Values")
plt.xticks(rotation=-45, ha='left', va='top')
ax.legend()
plt.subplots_adjust(bottom=0.2)
plt.show()
features = crop[['N', 'P','K','temperature', 'humidity', 'ph', 'rainfall']]
target = crop['label']
acc = []
model = []
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(features,target,test_size
= 0.2,random_state =2)
a=[]
for i in range(1,47,2):
    a.append(i)
len(a)
from sklearn.model_selection import GridSearchCV
grid_params = { 'n_neighbors' : a,
               'weights' : ['uniform','distance'],
               'metric' : ['minkowski','euclidean','manhattan']}

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

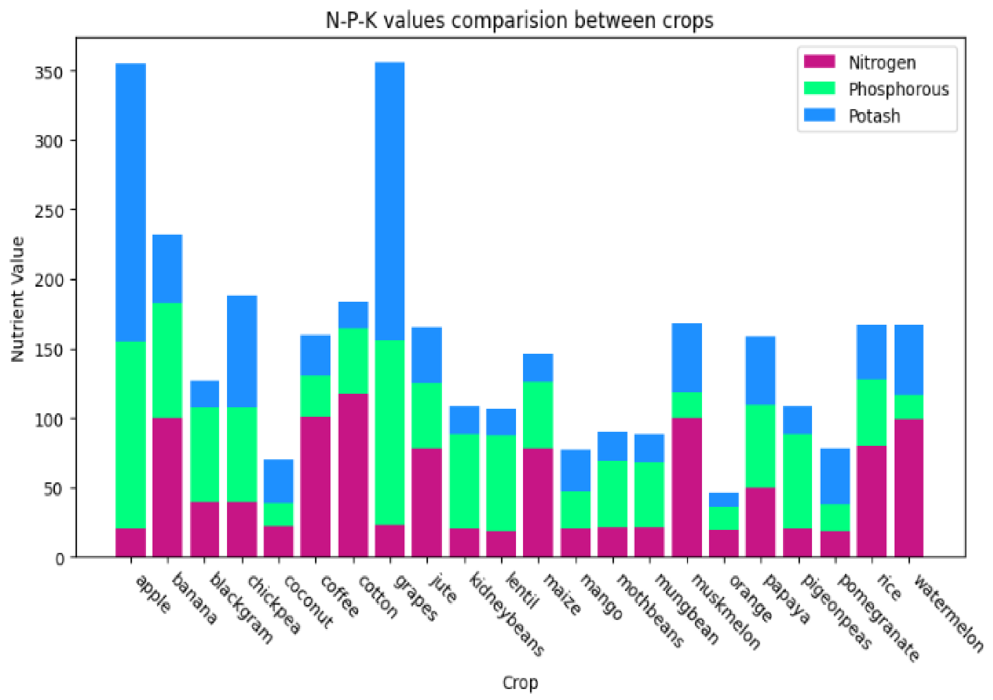
```
predicted_values = NaiveBayes.predict(x_test)
x = metrics.accuracy_score(y_test, predicted_values)
params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}
gs = GridSearchCV(GaussianNB(), params_NB, verbose = 1, cv=7, n_jobs = -1)
g_res = gs.fit(x_train, y_train)
gsresult=pd.DataFrame(g_res.cv_results_)
gsresult.head()
g_res.best_score_
g_res.best_params_
NaiveBayes=GaussianNB(var_smoothing= 1.873817422860383e-05)
score=cross_val_score(GaussianNB(var_smoothing=1.873817422860383e-05),feature
s,target,cv=5)
print('Cross validation score: ',score)
NaiveBayes.fit(x_train,y_train)
nb_train_accuracy = NaiveBayes.score(x_train,y_train)
print("Training accuracy = ",NaiveBayes.score(x_train,y_train))
nb_test_accuracy = NaiveBayes.score(x_test,y_test)
print("Testing accuracy = ",NaiveBayes.score(x_test,y_test))
score = cross_val_score(NaiveBayes,features,target,cv=5)
print('Cross validation score: ',score)
print(score.mean())
acc.append(score.mean())
model.append('Naive Bayes')
y_pred = NaiveBayes.predict(x_test)
y_true = y_test
from sklearn.metrics import confusion_matrix
cm_nb = confusion_matrix(y_true,y_pred)
f, ax = plt.subplots(figsize=(15,10))
sns.heatmap(cm_nb, annot=True, linewidth=0.5, fmt=".0f",  cmap='viridis', ax
= ax)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title('Predicted vs actual')
plt.show()
print(classification_report(y_true,y_pred))
```

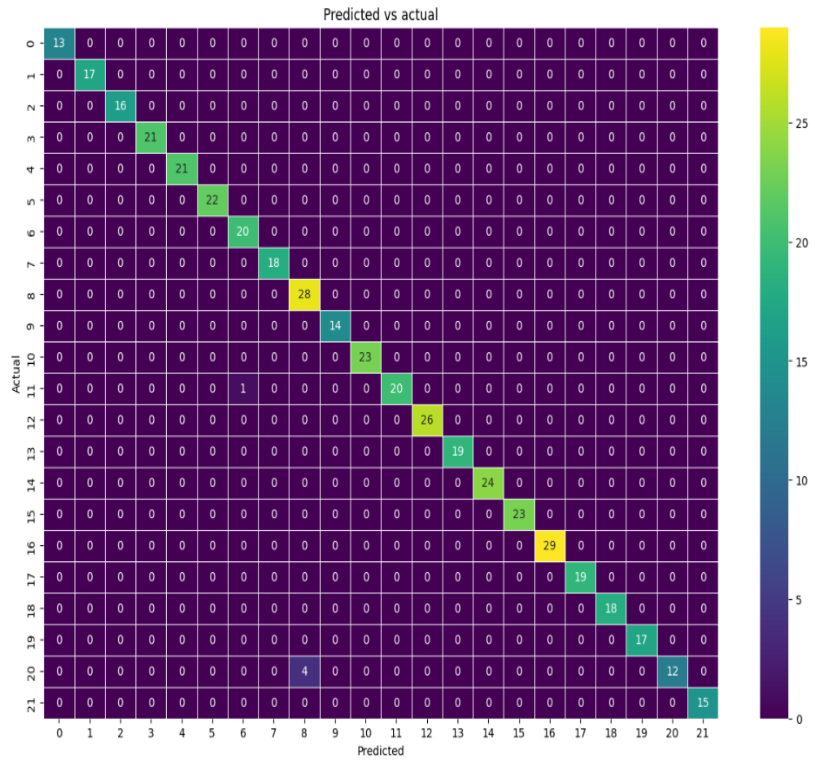
Experimental Results and Performance metrics for D2

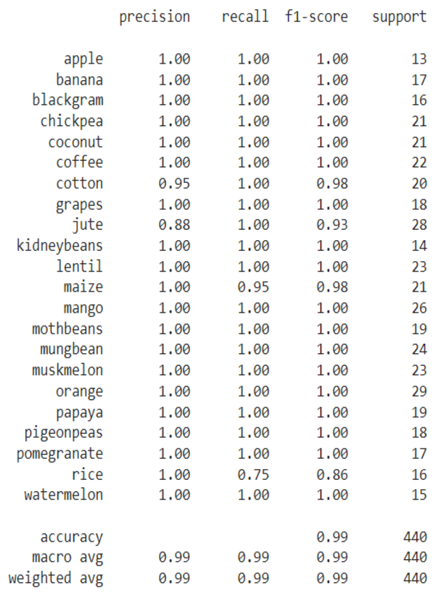




```
Fitting 7 folds for each of 100 candidates, totalling 700 fits
Cross validation score:  [0.99772727 0.99545455 0.99545455 0.99545455 0.99090909]
Training accuracy =  0.9960227272727272
Testing accuracy =  0.9886363636363636
Cross validation score:  [0.99772727 0.99545455 0.99545455 0.99545455 0.99090909]
0.9950000000000001
```






```
              precision    recall  f1-score   support

       apple       1.00      1.00      1.00        13
      banana       1.00      1.00      1.00        17
   blackgram       1.00      1.00      1.00        16
    chickpea       1.00      1.00      1.00        21
     coconut       1.00      1.00      1.00        21
      coffee       1.00      1.00      1.00        22
      cotton       0.95      1.00      0.98        20
      grapes       1.00      1.00      1.00        18
        jute       0.88      1.00      0.93        28
 kidneybeans       1.00      1.00      1.00        14
      lentil       1.00      1.00      1.00        23
       maize       1.00      0.95      0.98        21
       mango       1.00      1.00      1.00        26
   mothbeans       1.00      1.00      1.00        19
    mungbean       1.00      1.00      1.00        24
   muskmelon       1.00      1.00      1.00        23
      orange       1.00      1.00      1.00        29
      papaya       1.00      1.00      1.00        19
  pigeonpeas       1.00      1.00      1.00        18
 pomegranate       1.00      1.00      1.00        17
        rice       1.00      0.75      0.86        16
  watermelon       1.00      1.00      1.00        15

    accuracy                           0.99       440
   macro avg       0.99      0.99      0.99       440
weighted avg       0.99      0.99      0.99       440
```

Observations, inferences, Conclusion for D2

From the bar graph we can infer that a few crops require more nutrients than others like Nitrogen, Phosphorus and potassium. These crops also demand specific environmental conditions to grow. Naive Bayes algorithm’s accuracy has increased after tuning the hyperparameters. Hence this algorithm can be used to classify different crops based on its suitable soil conditions with good accuracy.

D3 Dataset Description

- The dataset is for a parametric classification problem with 3 classes: Bacterial Leaf Blight, Brown Spot, and Leaf Smut.
- The data has been divided into 80% for training, 15% for testing, and 5% for validation.
- To evaluate the model's performance, various metrics have been calculated, including accuracy, F1 score, precision, recall, and confusion matrix.
- The dataset is available for download from the UCI Machine Learning Repository.
- The dataset is titled "Rice Leaf Diseases Data Set".

Random Forest Algorithm:

Random Forest is a popular ensemble learning algorithm used for both classification and regression tasks. It **is an extension of decision trees, where a large number of trees are created and each tree makes its own individual prediction**. The final output is determined based on the mode or average of the predictions made by each tree.

Random forest works on the **Bagging principle.**

Working of Random Forest Algorithm:

1. The first step is to **randomly select a subset of the features** from the given dataset.
2. Then, the algorithm **builds a decision tree on this subset of features. It decides on the best split for each node based on the selected subset of features.**
3. This **process of selecting random features and building trees is repeated multiple times** until a forest of decision trees is created.

4. During the prediction phase, **each tree in the forest makes a prediction, and the final output is determined based on the mode or average of the individual predictions.**

Formula for Random Forest Algorithm:

1. The formula for calculating the impurity of a node in a decision tree is:
   impurity = sum(p(i) * (1 - p(i))), where i is the class label and p(i) is the probability of occurrence of class i.
2. The formula for calculating the information gain of a node in a decision tree is:
   information gain = impurity(parent) - [weighted average] impurity(children), where impurity(parent) is the impurity of the parent node, and impurity(children) is the impurity of the child nodes.
3. The formula for calculating the **mean squared error (**MSE) for regression tasks is:
   MSE = (1/n) * sum((y_i - y_hat_i)^2), where y_i is the actual value, y_hat_i is the predicted value, and n is the number of instances.

Advantages of Random Forest Algorithm:

1. Random Forest is **less prone to overfitting** compared to other machine learning algorithms.
2. It can **handle a large number of input features and maintain accuracy** even with noisy or missing data.
3. Random Forest can handle both **categorical and continuous data**.
4. It can provide feature importance ranking, which is useful for feature selection.

Limitations of Random Forest Algorithm:

1. Random Forest may take **longer to train** than other algorithms, especially on large datasets.
2. It is **difficult to interpret the results of a Random Forest model** compared to other algorithms like decision trees.
3. Random Forest may **not perform well on imbalanced datasets** where one class has significantly more samples than the others.

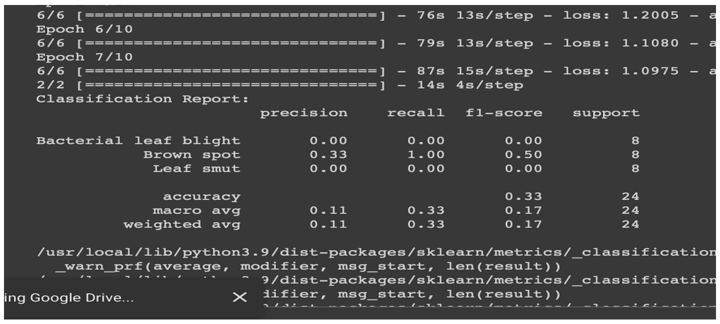
Code:

```
import os
import tensorflow as tf
```

```
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score, roc_auc_score, classification_report
from sklearn.model_selection import cross_val_score
import numpy as np
import matplotlib.pyplot as plt
import os
import seaborn as sns
from google.colab import drive
import seaborn as sns
!gdown --id 1yZhvtAn0c7KDaQqhgNMNYmNEET61uK98
!unzip ankush.zip
# Set the paths to the training, validation and testing directories
train_path = 'train'
val_path = 'val'
test_path = 'test'
# Define the data generators for training, validation and testing
train_datagen = ImageDataGenerator(
  rescale=1./255,
  shear_range=0.2,
  zoom_range=0.2,
  horizontal_flip=True
)
val_datagen = ImageDataGenerator(
  rescale=1./255
)
test_datagen = ImageDataGenerator(
  rescale=1./255
)
train_dataset = train_datagen.flow_from_directory(
  train_path,
  target_size=(320, 320), # Increase image size
  batch_size=16, # Decrease batch size
  class_mode='categorical',
  shuffle=False
)
val_dataset = val_datagen.flow_from_directory(
  val_path,
  target_size=(320, 320), # Increase image size
  batch_size=16, # Decrease batch size
  class_mode='categorical',
  shuffle=False
)
test_dataset = test_datagen.flow_from_directory(
  test_path,
  target_size=(320, 320), # Increase image size
  batch_size=16, # Decrease batch size
  class_mode='categorical',
  shuffle=False
)
# Load the pre-trained convolutional base of the model
conv_base = keras.applications.VGG16(
  weights='imagenet',
  include_top=False,
  input_shape=(320, 320, 3)
)
# Extract features from the training set using the pre-trained convolutional base
train_features = conv_base.predict(train_dataset, verbose=1)
# Extract features from the validation set using the pre-trained convolutional base
val_features = conv_base.predict(val_dataset, verbose=1)
# Extract features from the test set using the pre-trained convolutional base
test_features = conv_base.predict(test_dataset, verbose=1)
# Reshape the features to 2D arrays
train_features = np.reshape(train_features, (train_features.shape[0], -1))
val_features = np.reshape(val_features, (val_features.shape[0], -1))
test_features = np.reshape(test_features, (test_features.shape[0], -1))
# Get the labels for the training and validation sets
train_labels = train_dataset.classes
val_labels = val_dataset.classes
X = train_features
y = train_labels
# Train a random forest classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(train_features, train_labels)
# Evaluate the model on the validation set
val_pred = clf.predict(val_features)
val_f1_score = f1_score(val_labels, val_pred, average='weighted')
print('Validation Weighted F1 score:', val_f1_score)
# Evaluate the model on the test set and generate a classification report
test_pred = clf.predict(test_features)
target_names = list(test_dataset.class_indices.keys())
print('Classification Report:')
print(classification_report(test_dataset.classes, test_pred, target_names=target_names))
# Use cross-validation to get an estimate of the model's performance
from sklearn.model_selection import cross_val_score
# Perform 10-fold cross validation
scores = cross_val_score(clf, X, y, cv=10)
# Print the accuracy score for each fold
for i, score in enumerate(scores):
  print("Fold %d: %0.2f" % (i+1, score))
# Calculate the mean accuracy score and standard deviation
mean_score = scores.mean()
std_score = scores.std()
# Print the mean and standard deviation
print("Mean score: %0.2f" % mean_score)
print("Standard deviation: %0.2f" % std_score)
# Generate a confusion matrix for the test set
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(test_dataset.classes, test_pred)
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', xticklabels=target_names, yticklabels=target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
import os
import numpy as np
import matplotlib.pyplot as plt
# Set the paths to the training, validation and testing directories
train_path = 'train'
val_path = 'val'
test_path = 'test'
# Get the number of images in each directory
num_train = len(os.listdir(train_path))
num_val = len(os.listdir(val_path))
num_test = len(os.listdir(test_path))
# Print the number of images in each directory
print('Number of training classes:', num_train)
print('Number of validation classes:', num_val)
print('Number of test classes:', num_test)
# Get the class names and indices from the training set
class_indices = train_dataset.class_indices
class_names = list(class_indices.keys())
# Count the number of images in each class
num_images_per_class = [len(os.listdir(os.path.join(train_path, class_name))) for class_name in class_names]
# Plot a histogram of the class distribution
plt.bar(class_names, num_images_per_class)
plt.title('Class Distribution')
plt.xlabel('Class')
plt.ylabel('Number of Images')
plt.show()
# Create a list of the predicted and actual labels
predicted_labels = test_pred
actual_labels = test_dataset.classes
# Create a scatter plot of predicted vs actual labels
plt.scatter(actual_labels, predicted_labels)
# Set the axis labels and title
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Predicted vs Actual Labels')
# Show the plot
plt.show()
# Calculate the R2 score on the test set
from sklearn.metrics import r2_score
r2 = r2_score(test_dataset.classes, test_pred)
print('R2 Score:', r2)
# Define the hyperparameters to tune
hyperparameters = {
  'n_estimators': [50, 100, 200],
  'max_depth': [5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'class_weight': [None, 'balanced']
}
# Create a random forest classifier with default parameters
clf = RandomForestClassifier(random_state=42)
# Use grid search to find the best hyperparameters
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(clf, hyperparameters, cv=5, scoring='f1_weighted', n_jobs=-1)
grid_search.fit(train_features, train_labels)
# Print the best hyperparameters and F1 score
print('Best hyperparameters:', grid_search.best_params_)
# Use the best hyperparameters to train a new model and evaluate on the test set
clf = RandomForestClassifier(**grid_search.best_params_, random_state=42)
clf.fit(train_features, train_labels)
# Evaluate the model on the validation set
val_pred = clf.predict(val_features)
val_f1_score = f1_score(val_labels, val_pred, average='weighted')
print('Validation Weighted F1 score:', val_f1_score)
# Evaluate the model on the test set and generate a classification report
test_pred = clf.predict(test_features)
target_names = list(test_dataset.class_indices.keys())
print('Classification Report:')
print(classification_report(test_dataset.classes, test_pred, target_names=target_names))
# Generate a confusion matrix for the test set
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(test_dataset.classes, test_pred)
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', xticklabels=target_names, yticklabels=target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
```

Experimental Results and Performance Metrics Reported (D3)



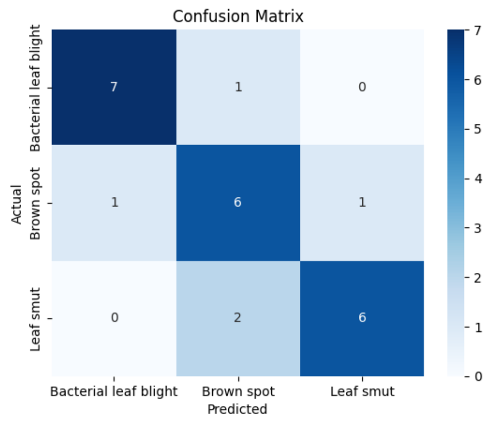
```
6/6 [==============================] - 76s 13s/step - loss: 1.2005 - a
Epoch 6/10
6/6 [==============================] - 79s 13s/step - loss: 1.1080 - a
Epoch 7/10
6/6 [==============================] - 87s 15s/step - loss: 1.0975 - a
2/2 [==============================] - 14s 4s/step
Classification Report:
                      precision    recall  f1-score   support

Bacterial leaf blight       0.00      0.00      0.00         8
           Brown spot       0.33      1.00      0.50         8
            Leaf smut       0.00      0.00      0.00         8

             accuracy                           0.33        24
            macro avg       0.11      0.33      0.17        24
         weighted avg       0.11      0.33      0.17        24

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification
  _warn_prf(average, modifier, msg_start, len(result))
                       9/dist-packages/sklearn/metrics/_classification
ing Google Drive...    difier, msg_start, len(result))
```

This is for cnn which was not considered

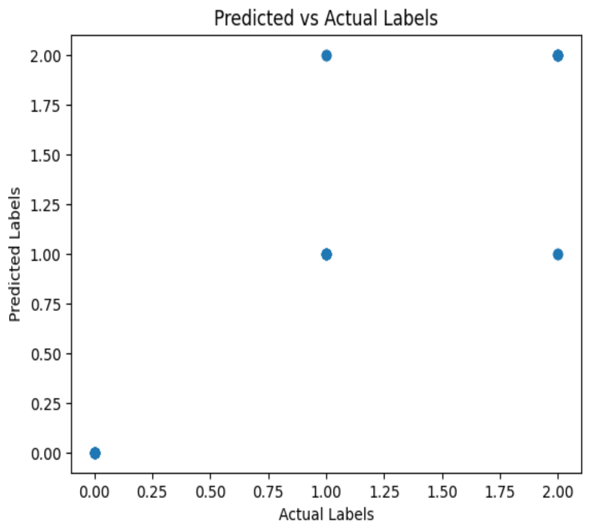
#no hyperparameters

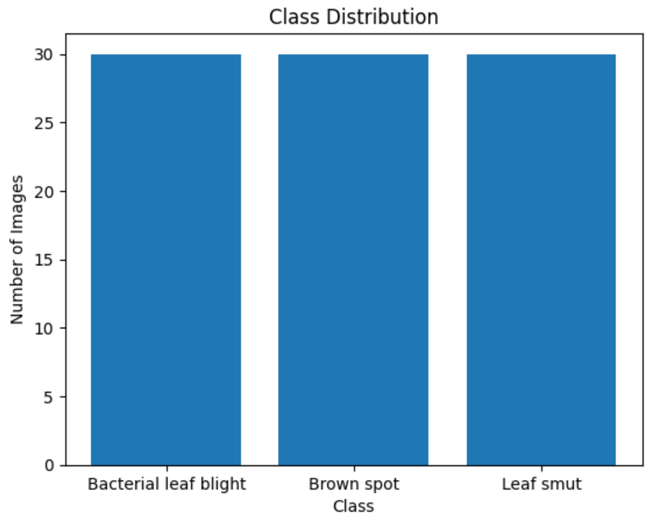
Found 90 images belonging to 3 classes.
Found 18 images belonging to 3 classes.
Found 24 images belonging to 3 classes.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5

```
58889256/58889256 [==============================] - 0s 0us/step
6/6 [==============================] - 131s 19s/step
2/2 [==============================] - 24s 3s/step
2/2 [==============================] - 29s 10s/step
Validation Weighted F1 score: 0.7282051282051282
Classification Report:
                    precision    recall  f1-score   support

Bacterial leaf blight       0.88      0.88      0.88         8
           Brown spot       0.67      0.75      0.71         8
            Leaf smut       0.86      0.75      0.80         8

             accuracy                           0.79        24
            macro avg       0.80      0.79      0.79        24
         weighted avg       0.80      0.79      0.79        24

Fold 1: 0.67
Fold 2: 0.78
Fold 3: 0.78
Fold 4: 0.89
Fold 5: 0.67
Fold 6: 0.67
Fold 7: 0.89
Fold 8: 0.44
Fold 9: 0.44
Fold 10: 0.11
Mean score: 0.63
Standard deviation: 0.23
```



```
Number of training classes: 3
Number of validation classes: 3
Number of test classes: 3
```



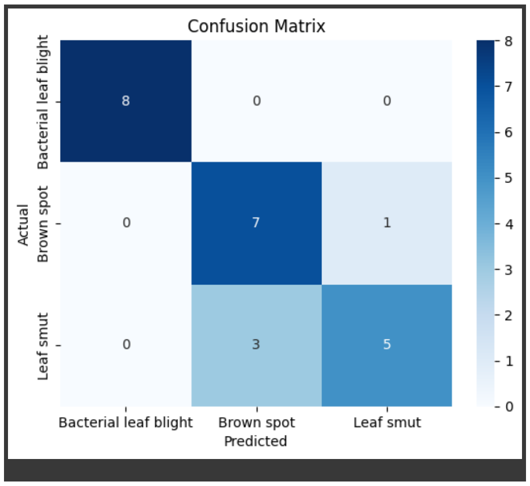


R2

Score: 0.8125

```
#with hyperparameters
```

```
Best hyperparameters: {'class_weight': None, 'max_depth': 5, 'min_samples_leaf': 4,
'min_samples_split': 2, 'n_estimators': 200}
Validation Weighted F1 score: 0.7714285714285714
Classification Report:
                       precision    recall  f1-score   support
Bacterial leaf blight       1.00      1.00      1.00         8
           Brown spot       0.70      0.88      0.78         8
            Leaf smut       0.83      0.62      0.71         8
             accuracy                           0.83        24
            macro avg       0.84      0.83      0.83        24
         weighted avg       0.84      0.83      0.83        24
```



Observations, inferences, Conclusion for D3

Plant disease detection models are an effective tool for maintaining health and productivity of crops. By using the random forest algorithm for this problem statement, a good accuracy was achieved. Overall accuracy was found out to be 83% . Classification of Bacterial leaf blight, Brown spot class and leaf smut achieved an accuracy of 88%, 82% and 85% respectively after the hyper-parameters were tuned.