#Get input values from the user

N= int(input("Enter the value of N: "))

p = int(input("Enter the value of P: ")) K= int(input("Enter the value of K: "))

Temperature = int(input("Enter the value of Temperature: ")) humidity=int(input("Enter the value of Humidity:"))

ph int(input("Enter the value of pH: "))

rainfall int(input("Enter the value of Rainfall: "))

# Create a new input array with the user-provided values new\_input = [N, P, K, Temperature, humidity, ph, rainfall]

# Make a prediction using the RandomForest model predict RF.predict([new\_input])

print("The Recomende crop is ",predict)

from tkinter import \*

import tkinter

import tkinter as tk

from tkinter import filedialog

import numpy as np

from tkinter.filedialog import askopenfilename

import pandas as pd

from tkinter import simpledialog

import pandas as pd

import numpy as np

import seaborn as sns

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

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

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

import os

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve

from sklearn.metrics import roc\_auc\_score

from sklearn import metrics

main = tkinter.Tk()

main.title("MACHINE LEARNING-BASED CROP RECOMMENDATION SYSTEM FOR ENHANCED YIELD PRODUCTION")

main.geometry("1000x650")

global filename

global x\_train,y\_train,x\_test,y\_test

global X, Y

global le

global dataset

accuracy = []

precision = []

recall = []

fscore = []

global classifier

global cnn\_model

def uploadDataset():

global filename

global dataset

filename = filedialog.askopenfilename(initialdir = "Dataset")

text.delete('1.0', END)

text.insert(END,filename+' Loaded\n')

dataset = pd.read\_csv(filename)

text.insert(END,str(dataset.head())+"\n\n")

def preprocessDataset():

global X, Y

global le

global dataset

global x\_train,y\_train,x\_test,y\_test

le = LabelEncoder()

text.delete('1.0', END)

dataset.fillna(0, inplace = True)

text.insert(END,str(dataset.head())+"\n\n")

# Create a count plot

sns.set(style="darkgrid") # Set the style of the plot

plt.figure(figsize=(8, 6)) # Set the figure size

# Replace 'dataset' with your actual DataFrame and 'Drug' with the column name

ax = sns.countplot(x='Label', data=dataset, palette="Set3")

plt.title("Count Plot") # Add a title to the plot

plt.xlabel("Crop Categories") # Add label to x-axis

plt.ylabel("Count") # Add label to y-axis

# Annotate each bar with its count value

for p in ax.patches:

ax.annotate(f'{p.get\_height()}', (p.get\_x() + p.get\_width() / 2., p.get\_height()),

ha='center', va='center', fontsize=10, color='black', xytext=(0, 5),

textcoords='offset points')

plt.show() # Display the plot

le = LabelEncoder()

dataset['Label'] = le.fit\_transform(dataset['Label'])

X=dataset.iloc[:,:-1].values

y=dataset.iloc[:,-1].values

text.insert(END,"Total records found in dataset: "+str(X.shape[0])+"\n\n")

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.2, random\_state=0)

text.insert(END,"Total records found in dataset to train: "+str(x\_train.shape[0])+"\n\n")

text.insert(END,"Total records found in dataset to test: "+str(x\_test.shape[0])+"\n\n")

print(x\_train)

def rocGraph(testY, predict, algorithm):

random\_probs = [0 for i in range(len(testY))]

p\_fpr, p\_tpr, \_ = roc\_curve(testY, random\_probs, pos\_label=1)

plt.plot(p\_fpr, p\_tpr, linestyle='--', color='orange',label="True classes")

ns\_fpr, ns\_tpr, \_ = roc\_curve(testY, predict,pos\_label=1)

plt.plot(ns\_fpr, ns\_tpr, linestyle='--', label='Predicted Classes')

plt.title(algorithm+" ROC Graph")

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive rate')

plt.show()

def custom\_knn\_classifier():

global x\_train, y\_train

KNN = KNeighborsClassifier(n\_neighbors=10,leaf\_size=30,metric='minkowski',) # Create an instance of KNeighborsClassifier

#x\_train\_reshaped = np.array(x\_train).reshape(-1, 1)

#x\_test\_reshaped = np.array(x\_test).reshape(-1, 1)

KNN.fit(x\_train, y\_train)

predict = KNN.predict(x\_test)

p = precision\_score(y\_test, predict, average='macro',zero\_division=0) \* 100

r = recall\_score(y\_test, predict, average='macro',zero\_division=0) \* 100

f = f1\_score(y\_test, predict, average='macro',zero\_division=0) \* 100

a = accuracy\_score(y\_test, predict) \* 100

accuracy.append(a)

precision.append(p)

recall.append(r)

fscore.append(f)

text.insert(END, "KNN Precision : " + str(p) + "\n")

text.insert(END, "KNN Recall : " + str(r) + "\n")

text.insert(END, "KNN FMeasure : " + str(f) + "\n")

text.insert(END, "KNN Accuracy : " + str(a) + "\n\n")

rocGraph(y\_test, predict, "KNN")

# Compute confusion matrix

cm = confusion\_matrix(y\_test,predict)

# Compute classification report

report = classification\_report(y\_test,predict,zero\_division=0)

# Display confusion matrix in the Text widget

text.insert(END, "Confusion Matrix:\n")

text.insert(END, str(cm) + "\n\n")

# Display classification report in the Text widget

text.insert(END, "Classification Report:\n")

text.insert(END, report)

def Randomforestclassifier():

global x\_train, y\_train, x\_test, y\_test

global rf

rf=RandomForestClassifier()

rf.fit(x\_train, y\_train)

predict = rf.predict(x\_test)

p = precision\_score(y\_test, predict, average='macro', zero\_division=0) \* 100

r = recall\_score(y\_test, predict, average='macro', zero\_division=0) \* 100

f = f1\_score(y\_test, predict, average='macro', zero\_division=0) \* 100

a = accuracy\_score(y\_test, predict) \* 100

accuracy.append(a)

precision.append(p)

recall.append(r)

fscore.append(f)

# Display precision, recall, F1-score, and accuracy in the Text widget

text.insert(END, "RF Precision: " + str(p) + "\n")

text.insert(END, "RF Recall: " + str(r) + "\n")

text.insert(END, "RF FMeasure: " + str(f) + "\n")

text.insert(END, "RF Accuracy: " + str(a) + "\n\n")

rocGraph(y\_test, predict, "RFC")

# Compute confusion matrix

cm = confusion\_matrix(y\_test, predict)

# Compute classification report

report = classification\_report(y\_test, predict,zero\_division=0)

# Display confusion matrix in the Text widget

text.insert(END, "Confusion Matrix:\n")

text.insert(END, str(cm) + "\n\n")

# Display classification report in the Text widget

text.insert(END, "Classification Report:\n")

text.insert(END, report)

crop\_classes = ['rice', 'maize', 'chickpea', 'kidneybeans', 'pigeonpeas',

'mothbeans', 'mungbean', 'blackgram', 'lentil', 'pomegranate',

'banana', 'mango', 'grapes', 'watermelon', 'muskmelon', 'apple',

'orange', 'papaya', 'coconut', 'cotton', 'jute', 'coffee']

def Predict():

filename = filedialog.askopenfilename(initialdir="Dataset")

Dataset = pd.read\_csv(filename)

Dataset=Dataset.values

prediction=rf.predict(Dataset)

for i in range(len(prediction)):

if prediction[i] == 0:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'rice'\n\n")

elif prediction[i] == 1:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'maize'\n\n")

elif prediction[i] == 2:

text.insert(END,"Test Data : "+str(Dataset[i])+" ===> Class: 'chickpea'\n\n")

elif prediction[i] == 3:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'kidneybeans'\n\n")

elif prediction[i] == 4:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'pigeonpeas'\n\n")

elif prediction[i] == 5:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'mothbeans'\n\n")

elif prediction[i] == 6:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'mungbean'\n\n")

elif prediction[i] == 7:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'blackgram'\n\n")

elif prediction[i] == 8:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'lentil'\n\n")

elif prediction[i] == 9:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'pomegranate'\n\n")

elif prediction[i] == 10:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'banana'\n\n")

elif prediction[i] == 11:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'mango'\n\n")

elif prediction[i] == 12:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'grapes'\n\n")

elif prediction[i] == 13:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'watermelon'\n\n")

elif prediction[i] == 14:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'muskmelon'\n\n")

elif prediction[i] == 15:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'apple'\n\n")

elif prediction[i] == 16:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'orange'\n\n")

elif prediction[i] == 17:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'papaya'\n\n")

elif prediction[i] == 18:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'coconut'\n\n")

elif prediction[i] == 19:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'cotton'\n\n")

elif prediction[i] == 20:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'jute'\n\n")

elif prediction[i] == 21:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'coffee'\n\n")

else:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> UNKNOWN\n\n")

def graph():

# Create a DataFrame

df = pd.DataFrame([

['KNN', 'Precision', precision[0]],

['KNN', 'Recall', recall[0]],

['KNN', 'F1 Score', fscore[0]],

['KNN', 'Accuracy', accuracy[0]],

['rf', 'Precision', precision[-1]],

['rf', 'Recall', recall[-1]],

['rf', 'F1 Score', fscore[-1]],

['rf', 'Accuracy', accuracy[-1]],

], columns=['Parameters', 'Algorithms', 'Value'])

# Pivot the DataFrame and plot the graph

pivot\_df = df.pivot\_table(index='Parameters', columns='Algorithms', values='Value', aggfunc='first')

pivot\_df.plot(kind='bar')

# Set graph properties

plt.title('Classifier Performance Comparison')

plt.ylabel('Score')

plt.xticks(rotation=0)

plt.tight\_layout()

# Display the graph

plt.show()

def close():

main.destroy()

font = ('times', 16, 'bold')

title = Label(main, text='MACHINE LEARNING-BASED CROP RECOMMENDATION SYSTEM FOR ENHANCED YIELD PRODUCTION', justify=LEFT)

title.config(bg='lavender blush', fg='black')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=100,y=5)

title.pack()

font1 = ('times', 13, 'bold')

uploadButton = Button(main, text="Upload Dataset", command=uploadDataset)

uploadButton.place(x=200,y=100)

uploadButton.config(font=font1)

preprocessButton = Button(main, text="Preprocess Dataset", command=preprocessDataset)

preprocessButton.place(x=500,y=100)

preprocessButton.config(font=font1)

knnButton = Button(main, text="KNeighborsClassifier", command=custom\_knn\_classifier)

knnButton.place(x=500,y=150)

knnButton.config(font=font1)

LRButton = Button(main, text="Randomforestclassifier", command=Randomforestclassifier)

LRButton.place(x=200,y=150)

LRButton.config(font=font1)

predictButton = Button(main, text="Prediction", command=Predict)

predictButton.place(x=500,y=200)

predictButton.config(font=font1)

graphButton = Button(main, text="Comparison Graph", command=graph)

graphButton.place(x=200,y=250)

graphButton.config(font=font1)

exitButton = Button(main, text="Exit", command=close)

exitButton.place(x=500,y=250)

exitButton.config(font=font1)

font1 = ('times', 12, 'bold')

text=Text(main,height=20,width=120)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=10,y=300)

text.config(font=font1)

main.config(bg='LightSteelBlue1')

main.mainloop()

import pandas as pd

# Define the dataset as a list of dictionaries

data = [

{"N": 0.8, "P": 0.4, "K": 0.6, "Temperature": 25, "Humidity": 70, "pH": 6.5, "Rainfall": 150, "Label": "Rice"},

{"N": 0.6, "P": 0.3, "K": 0.5, "Temperature": 30, "Humidity": 65, "pH": 6.0, "Rainfall": 100, "Label": "Wheat"},

{"N": 0.7, "P": 0.5, "K": 0.4, "Temperature": 28, "Humidity": 75, "pH": 6.2, "Rainfall": 120, "Label": "Maize"},

{"N": 0.9, "P": 0.6, "K": 0.7, "Temperature": 22, "Humidity": 80, "pH": 5.8, "Rainfall": 180, "Label": "Rice"},

{"N": 0.5, "P": 0.4, "K": 0.3, "Temperature": 32, "Humidity": 60, "pH": 6.3, "Rainfall": 80, "Label": "Wheat"},

{"N": 0.8, "P": 0.5, "K": 0.6, "Temperature": 26, "Humidity": 72, "pH": 6.0, "Rainfall": 130, "Label": "Maize"},

{"N": 0.7, "P": 0.4, "K": 0.5, "Temperature": 24, "Humidity": 68, "pH": 6.1, "Rainfall": 110, "Label": "Rice"},

{"N": 0.6, "P": 0.3, "K": 0.4, "Temperature": 29, "Humidity": 70, "pH": 6.2, "Rainfall": 95, "Label": "Wheat"},

{"N": 0.9, "P": 0.6, "K": 0.7, "Temperature": 23, "Humidity": 78, "pH": 5.9, "Rainfall": 160, "Label": "Maize"},

{"N": 0.8, "P": 0.5, "K": 0.5, "Temperature": 27, "Humidity": 74, "pH": 6.1, "Rainfall": 140, "Label": "Rice"},

{"N": 0.7, "P": 0.4, "K": 0.6, "Temperature": 25, "Humidity": 67, "pH": 6.0, "Rainfall": 125, "Label": "Wheat"},

{"N": 0.6, "P": 0.3, "K": 0.4, "Temperature": 30, "Humidity": 73, "pH": 6.3, "Rainfall": 105, "Label": "Maize"},

{"N": 0.8, "P": 0.6, "K": 0.7, "Temperature": 21, "Humidity": 76, "pH": 5.8, "Rainfall": 175, "Label": "Rice"},

{"N": 0.5, "P": 0.4, "K": 0.5, "Temperature": 31, "Humidity": 62, "pH": 6.2, "Rainfall": 85, "Label": "Wheat"},

{"N": 0.7, "P": 0.5, "K": 0.6, "Temperature": 27, "Humidity": 71, "pH": 6.1, "Rainfall": 135, "Label": "Maize"}

]

# Convert to DataFrame

df = pd.DataFrame(data)

# Save to CSV file

df.to\_csv('crop\_dataset.csv', index=False

from tkinter import \*

import tkinter

import tkinter as tk

from tkinter import filedialog

import numpy as np

from tkinter.filedialog import askopenfilename

import pandas as pd

from tkinter import simpledialog

import pandas as pd

import numpy as np

import seaborn as sns

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

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

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

import os

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve

from sklearn.metrics import roc\_auc\_score

from sklearn import metrics

main = tkinter.Tk()

main.title("MACHINE LEARNING-BASED CROP RECOMMENDATION SYSTEM FOR ENHANCED YIELD PRODUCTION")

main.geometry("1000x650")

global filename

global x\_train,y\_train,x\_test,y\_test

global X, Y

global le

global dataset

accuracy = []

precision = []

recall = []

fscore = []

global classifier

global cnn\_model

def uploadDataset():

global filename

global dataset

filename = filedialog.askopenfilename(initialdir = "Dataset")

text.delete('1.0', END)

text.insert(END,filename+' Loaded\n')

dataset = pd.read\_csv(filename)

text.insert(END,str(dataset.head())+"\n\n")

def preprocessDataset():

global X, Y

global le

global dataset

global x\_train,y\_train,x\_test,y\_test

le = LabelEncoder()

text.delete('1.0', END)

dataset.fillna(0, inplace = True)

text.insert(END,str(dataset.head())+"\n\n")

# Create a count plot

sns.set(style="darkgrid") # Set the style of the plot

plt.figure(figsize=(8, 6)) # Set the figure size

# Replace 'dataset' with your actual DataFrame and 'Drug' with the column name

ax = sns.countplot(x='label', data=dataset, palette="Set3")

plt.title("Count Plot") # Add a title to the plot

plt.xlabel("Crop Categories") # Add label to x-axis

plt.ylabel("Count") # Add label to y-axis

# Annotate each bar with its count value

for p in ax.patches:

ax.annotate(f'{p.get\_height()}', (p.get\_x() + p.get\_width() / 2., p.get\_height()),

ha='center', va='center', fontsize=10, color='black', xytext=(0, 5),

textcoords='offset points')

plt.show() # Display the plot

le = LabelEncoder()

dataset['label'] = le.fit\_transform(dataset['label'])

X=dataset.iloc[:,:-1].values

y=dataset.iloc[:,-1].values

text.insert(END,"Total records found in dataset: "+str(X.shape[0])+"\n\n")

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.2, random\_state=0)

text.insert(END,"Total records found in dataset to train: "+str(x\_train.shape[0])+"\n\n")

text.insert(END,"Total records found in dataset to test: "+str(x\_test.shape[0])+"\n\n")

print(x\_train)

def rocGraph(testY, predict, algorithm):

random\_probs = [0 for i in range(len(testY))]

p\_fpr, p\_tpr, \_ = roc\_curve(testY, random\_probs, pos\_label=1)

plt.plot(p\_fpr, p\_tpr, linestyle='--', color='orange',label="True classes")

ns\_fpr, ns\_tpr, \_ = roc\_curve(testY, predict,pos\_label=1)

plt.plot(ns\_fpr, ns\_tpr, linestyle='--', label='Predicted Classes')

plt.title(algorithm+" ROC Graph")

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive rate')

plt.show()

def custom\_knn\_classifier():

global x\_train, y\_train

KNN = KNeighborsClassifier(n\_neighbors=10,leaf\_size=30,metric='minkowski',) # Create an instance of KNeighborsClassifier

#x\_train\_reshaped = np.array(x\_train).reshape(-1, 1)

#x\_test\_reshaped = np.array(x\_test).reshape(-1, 1)

KNN.fit(x\_train, y\_train)

predict = KNN.predict(x\_test)

p = precision\_score(y\_test, predict, average='macro') \* 100

r = recall\_score(y\_test, predict, average='macro') \* 100

f = f1\_score(y\_test, predict, average='macro') \* 100

a = accuracy\_score(y\_test, predict) \* 100

accuracy.append(a)

precision.append(p)

recall.append(r)

fscore.append(f)

text.insert(END, "KNN Precision : " + str(p) + "\n")

text.insert(END, "KNN Recall : " + str(r) + "\n")

text.insert(END, "KNN FMeasure : " + str(f) + "\n")

text.insert(END, "KNN Accuracy : " + str(a) + "\n\n")

rocGraph(y\_test, predict, "KNN")

# Compute confusion matrix

cm = confusion\_matrix(y\_test,predict)

# Compute classification report

report = classification\_report(y\_test,predict)

# Display confusion matrix in the Text widget

text.insert(END, "Confusion Matrix:\n")

text.insert(END, str(cm) + "\n\n")

# Display classification report in the Text widget

text.insert(END, "Classification Report:\n")

text.insert(END, report)

def Randomforestclassifier():

global x\_train, y\_train, x\_test, y\_test

global rf

rf=RandomForestClassifier()

rf.fit(x\_train, y\_train)

predict = rf.predict(x\_test)

p = precision\_score(y\_test, predict, average='macro', zero\_division=0) \* 100

r = recall\_score(y\_test, predict, average='macro', zero\_division=0) \* 100

f = f1\_score(y\_test, predict, average='macro', zero\_division=0) \* 100

a = accuracy\_score(y\_test, predict) \* 100

accuracy.append(a)

precision.append(p)

recall.append(r)

fscore.append(f)

# Display precision, recall, F1-score, and accuracy in the Text widget

text.insert(END, "RF Precision: " + str(p) + "\n")

text.insert(END, "RF Recall: " + str(r) + "\n")

text.insert(END, "RF FMeasure: " + str(f) + "\n")

text.insert(END, "RF Accuracy: " + str(a) + "\n\n")

rocGraph(y\_test, predict, "RFC")

# Compute confusion matrix

cm = confusion\_matrix(y\_test, predict)

# Compute classification report

report = classification\_report(y\_test, predict)

# Display confusion matrix in the Text widget

text.insert(END, "Confusion Matrix:\n")

text.insert(END, str(cm) + "\n\n")

# Display classification report in the Text widget

text.insert(END, "Classification Report:\n")

text.insert(END, report)

crop\_classes = ['rice', 'maize', 'chickpea', 'kidneybeans', 'pigeonpeas',

'mothbeans', 'mungbean', 'blackgram', 'lentil', 'pomegranate',

'banana', 'mango', 'grapes', 'watermelon', 'muskmelon', 'apple',

'orange', 'papaya', 'coconut', 'cotton', 'jute', 'coffee']

def Predict():

filename = filedialog.askopenfilename(initialdir="Dataset")

Dataset = pd.read\_csv(filename)

Dataset=Dataset.values

prediction=rf.predict(Dataset)

for i in range(len(prediction)):

if prediction[i] == 0:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'rice'\n\n")

elif prediction[i] == 1:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'maize'\n\n")

elif prediction[i] == 2:

text.insert(END,"Test Data : "+str(Dataset[i])+" ===> Class: 'chickpea'\n\n")

elif prediction[i] == 3:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'kidneybeans'\n\n")

elif prediction[i] == 4:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'pigeonpeas'\n\n")

elif prediction[i] == 5:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'mothbeans'\n\n")

elif prediction[i] == 6:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'mungbean'\n\n")

elif prediction[i] == 7:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'blackgram'\n\n")

elif prediction[i] == 8:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'lentil'\n\n")

elif prediction[i] == 9:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'pomegranate'\n\n")

elif prediction[i] == 10:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'banana'\n\n")

elif prediction[i] == 11:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'mango'\n\n")

elif prediction[i] == 12:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'grapes'\n\n")

elif prediction[i] == 13:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'watermelon'\n\n")

elif prediction[i] == 14:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'muskmelon'\n\n")

elif prediction[i] == 15:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'apple'\n\n")

elif prediction[i] == 16:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'orange'\n\n")

elif prediction[i] == 17:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'papaya'\n\n")

elif prediction[i] == 18:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'coconut'\n\n")

elif prediction[i] == 19:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'cotton'\n\n")

elif prediction[i] == 20:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'jute'\n\n")

elif prediction[i] == 21:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> Class: 'coffee'\n\n")

else:

text.insert(END,"Test Data : "+str(Dataset[i])+" ====> UNKNOWN\n\n")

def graph():

# Create a DataFrame

df = pd.DataFrame([

['KNN', 'Precision', precision[0]],

['KNN', 'Recall', recall[0]],

['KNN', 'F1 Score', fscore[0]],

['KNN', 'Accuracy', accuracy[0]],

['rf', 'Precision', precision[-1]],

['rf', 'Recall', recall[-1]],

['rf', 'F1 Score', fscore[-1]],

['rf', 'Accuracy', accuracy[-1]],

], columns=['Parameters', 'Algorithms', 'Value'])

# Pivot the DataFrame and plot the graph

pivot\_df = df.pivot\_table(index='Parameters', columns='Algorithms', values='Value', aggfunc='first')

pivot\_df.plot(kind='bar')

# Set graph properties

plt.title('Classifier Performance Comparison')

plt.ylabel('Score')

plt.xticks(rotation=0)

plt.tight\_layout()

# Display the graph

plt.show()

def close():

main.destroy()

font = ('times', 16, 'bold')

title = Label(main, text='MACHINE LEARNING-BASED CROP RECOMMENDATION SYSTEM FOR ENHANCED YIELD PRODUCTION', justify=LEFT)

title.config(bg='lavender blush', fg='black')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=100,y=5)

title.pack()

font1 = ('times', 13, 'bold')

uploadButton = Button(main, text="Upload Dataset", command=uploadDataset)

uploadButton.place(x=200,y=100)

uploadButton.config(font=font1)

preprocessButton = Button(main, text="Preprocess Dataset", command=preprocessDataset)

preprocessButton.place(x=500,y=100)

preprocessButton.config(font=font1)

knnButton = Button(main, text="KNeighborsClassifier", command=custom\_knn\_classifier)

knnButton.place(x=500,y=150)

knnButton.config(font=font1)

LRButton = Button(main, text="Randomforestclassifier", command=Randomforestclassifier)

LRButton.place(x=200,y=150)

LRButton.config(font=font1)

predictButton = Button(main, text="Prediction", command=Predict)

predictButton.place(x=500,y=200)

predictButton.config(font=font1)

graphButton = Button(main, text="Comparison Graph", command=graph)

graphButton.place(x=200,y=250)

graphButton.config(font=font1)

exitButton = Button(main, text="Exit", command=close)

exitButton.place(x=500,y=250)

exitButton.config(font=font1)

font1 = ('times', 12, 'bold')

text=Text(main,height=20,width=120)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=10,y=300)

text.config(font=font1)

main.config(bg='LightSteelBlue1')

main.mainloop()