PROJECT DEVELOPMENT PHASE

Sprint - IV

Date	17 November 2022
Team ID	PNT2022TMID49457
Project Name	Fertilizers Recommendation System for Disease Prediction
Maximum Marks	8 Marks

Integration the Deployed Model with Flask

Web Application

App.py

```
# Importing essential libraries and modules

from flask import Flask, render_template, request, Markup
import numpy as np
import pandas as pd
from utils.disease import disease_dic
from utils.fertilizer import fertilizer_dic
import requests
import config
import pickle
import io
import torch
from torchvision import transforms
from PIL import Image
from utils.model import ResNet9

import mysql.connector

conn=mysql.connector.connect(host="localhost", user="root", password="", database="login")
cursor=conn.cursor()


disease_classes = ['Apple___Black_rot',
                   'Apple___healthy',
                   'Corn_(maize)___Northern_Leaf_Blight',
                   'Corn_(maize)___healthy',
                   'Peach___Bacterial_spot',
                   'Peach___healthy',
                   'Pepper,bell__Bacterial_spot',
```

```
 'Pepper,bell__healthy',
 'Potato___Early_blight',
 'Potato___Late_blight',
 'Potato___healthy',
 'Tomato___Bacterial_spot',
 'Tomato___Late_blight',
 'Tomato___Leaf_Mold',
 'Tomato___Septoria_leaf_spot']
disease_model_path = 'models/plant-disease-model.pth'
disease_model = ResNet9(3, len(disease_classes))
disease_model.load_state_dict(torch.load(
    disease_model_path, map_location=torch.device('cpu')))
disease_model.eval()
def predict_image(img, model=disease_model):
    """
    Transforms image to tensor and predicts disease label
    :params: image
    :return: prediction (string)
    """
    transform = transforms.Compose([
        transforms.Resize(256),
        transforms.ToTensor(),
    ])
    image = Image.open(io.BytesIO(img))
    img_t = transform(image)
    img_u = torch.unsqueeze(img_t, 0)

    # Get predictions from model
    yb = model(img_u)
    # Pick index with highest probability
    _, preds = torch.max(yb, dim=1)
    prediction = disease_classes[preds[0].item()]
    # Retrieve the class label
    return prediction

app = Flask(_name_)

# render home page
@app.route('/')
def home():
    title = 'Harvestify - Home'
    return render_template('index.html', title=title)

@app.route('/fertilizer')
def fertilizer_recommendation():
    title = 'Harvestify - Fertilizer Suggestion'

    return render_template('fertilizer.html', title=title)

@app.route('/login', methods=['GET', 'POST'])
def login():  # put application's code here
```

```
    return render_template('login.html')

@app.route('/register', methods=['GET', 'POST'])
def register():
    return render_template('register.html')

@app.route('/success', methods=['GET', 'POST'])
def success():
    return render_template('success.html')

@app.route('/login_validation', methods=['POST'])
def login_validation():
    email=request.form.get('email')
    password=request.form.get('password')

    cursor.execute("""SELECT * FROM users WHERE email LIKE'{}' AND password LIKE
'{}'""".format(email,password))
    users = cursor.fetchall()

    if len(users)>0:
        return render_template('success.html')
    else:
        return render_template('login.html', prediction_text = "1" )

@app.route('/add_user', methods=['POST'])
def add_user():
    name= request.form.get('name')
    email = request.form.get('email')
    password = request.form.get('password')

    cursor.execute("""INSERT INTO users(id, name, email, password) VALUES
(NULL,'{}','{}','{}')""".format(name,email,password))
    conn.commit()
    return render_template('login.html', prediction_text = "0")

@app.route('/fertilizer-predict', methods=['POST'])
def fert_recommend():
    title = 'Harvestify - Fertilizer Suggestion'

    crop_name = str(request.form['cropname'])
    N = int(request.form['nitrogen'])
    P = int(request.form['phosphorous'])
    K = int(request.form['pottasium'])
    # ph = float(request.form['ph'])

    df = pd.read_csv('Data/fertilizer.csv')

    nr = df[df['Crop'] == crop_name]['N'].iloc[0]
    pr = df[df['Crop'] == crop_name]['P'].iloc[0]
    kr = df[df['Crop'] == crop_name]['K'].iloc[0]
```

```
    n = nr - N
    p = pr - P
    k = kr – K
k = kr - K
    temp = {abs(n): "N", abs(p): "P", abs(k): "K"}
    max_value = temp[max(temp.keys())]
    if max_value == "N":
        if n < 0:
            key = 'NHigh'
        else:
            key = "Nlow"
    elif max_value == "P":
        if p < 0:
            key = 'PHigh'
        else:
            key = "Plow"
    else:
        if k < 0:
            key = 'KHigh'
        else:
            key = "Klow"

    response = Markup(str(fertilizer_dic[key]))

    return render_template('fertilizer-result.ht,ml', recommendation=response, title=title)

# render disease prediction result page
@app.route('/disease-predict', methods=['GET', 'POST'])
def disease_prediction():
    title = 'Harvestify - Disease Detection'

    if request.method == 'POST':
        if 'file' not in request.files:
            return redirect(request.url)
        file = request.files.get('file')
        if not file:
            return render_template('disease.html', title=title)
        try:
            img = file.read()

            prediction = predict_image(img)

            prediction = Markup(str(disease_dic[prediction]))
            return render_template('disease-result.html', prediction=prediction, title=title)
        except:
            pass
    return render_template('disease.html', title=title)

# if name == '_main_':
    app.run(debug=False)
```