Creating a Flask app

Virtual Internship – Ahmed Abubekr Elageib
Ahmedkata@Gmail.com

20-Jan-2023

Background

Objective :

- Creating Flask Web app using a simple dataset to do a simple task

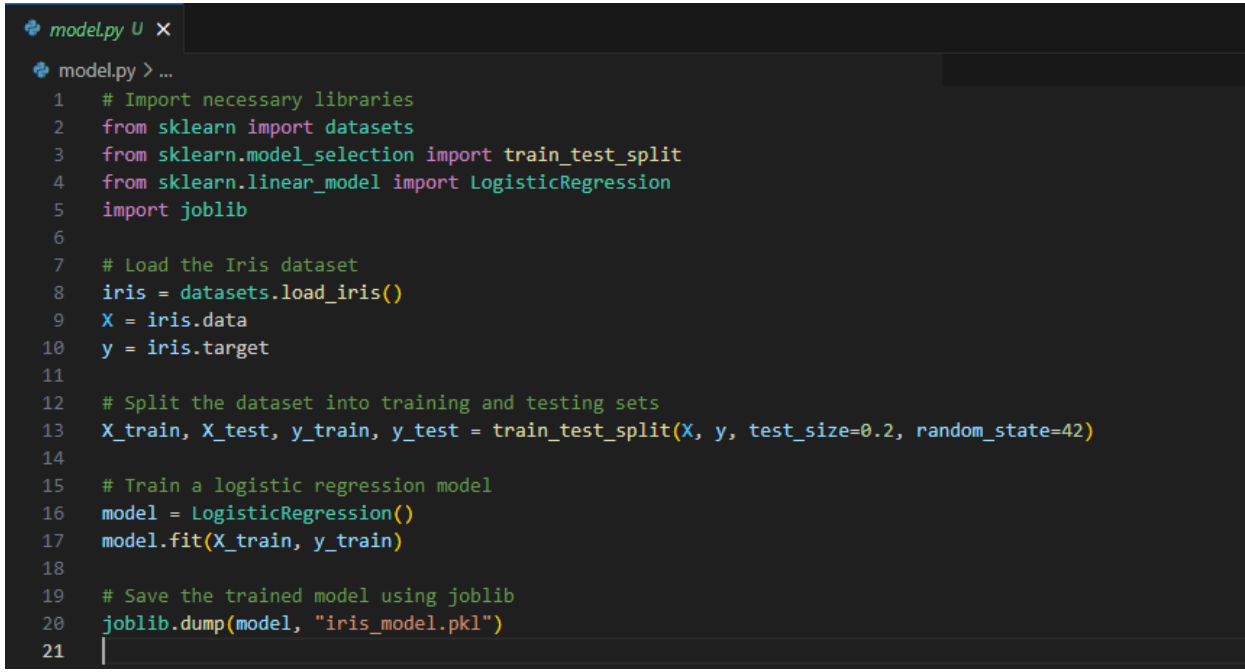
The guide has been divided into several parts:

- Model building
- App building
- deployment

Model Building

Firstly we create a python file to build our classification model
Here's a breakdown of the code you provided:

1. Import Libraries:
sklearn.datasets: This library provides access to various datasets, including the Iris dataset.
sklearn.model_selection: This library contains tools for splitting datasets into training and testing sets.
sklearn.linear_model: This library houses linear models for classification and regression, including LogisticRegression.
joblib: This library is used for saving and loading machine learning models.
2. Load the Iris Dataset:
iris = datasets.load_iris(): This loads the Iris dataset, which contains data about three different Iris flower species.
X = iris.data: This extracts the features (sepal length, sepal width, petal length, petal width) from the dataset.
y = iris.target: This extracts the target variable (the species of the Iris flower) from the dataset.
3. Split the Dataset:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42): This splits the dataset into two parts:
Training set (80% of the data): Used to train the model.
Testing set (20% of the data): Used to evaluate the model's performance.
4. Train the Model:
model = LogisticRegression(): This creates an instance of the LogisticRegression model.
model.fit(X_train, y_train): This trains the model using the training data.
5. Save the Model:
joblib.dump(model, "iris_model.pkl"): This saves the trained model to a file named "iris_model.pkl" using joblib.



If the code is executed correctly, a new file named iris_model.pkl should be created

App Building-Flask functions

Secondly we write the web app code, which builds a web application using Flask to deploy a machine learning model:

1. Import Libraries:

•flask: This library is used to create web applications in Python.
•request: A Flask object that allows you to access form data submitted by the user.
•render_template: A Flask function that renders HTML templates.
•numpy: This library is used for numerical computations and array operations.
•joblib: This library is used to load the saved machine learning model.
•sklearn.datasets: This library is used to load the Iris dataset to access target names.

2. Initialize Flask App:

•app = Flask(__name__): This creates an instance of the Flask application.

3. Load the Model:

•model = joblib.load(r"D:\DataGlacier\DataGlacierInternship\Week4\iris_model.pkl"): This loads the previously trained Iris flower classification model from a file.

4. Define Routes:

•Homepage (/):
    •@app.route('/'): This defines a route for the home page of the web application.
    •render_template('index.html'): This renders an HTML template named "index.html", which likely contains a form for user input.
•Prediction (/predict):
    •@app.route('/predict', methods=['POST']): This defines a route for handling predictions, specifically POST requests (which typically send form data).
    •Get Input:
        •The code retrieves the values of sepal length, sepal width, petal length, and petal width from the submitted form data.
    •Make Prediction:
        •It creates an array with the input data and uses the loaded model to make a prediction about the Iris flower species.
    •Map Prediction to Class Name:
        •It uses the iris.target_names to get the actual species name corresponding to the predicted class.
    •Render Result:
        •It renders an HTML template named "result.html", passing the predicted species as a variable to be displayed.

5. Run the App:

•if __name__ == '__main__':: This ensures the code within this block runs only when the script is executed directly, not when imported as a module.
•app.run(): This starts the Flask development server, making the web application accessible.

app.py M

app.py > ...

You, 49 minutes ago | 1 author (You)

```python
# Import necessary libraries
from flask import Flask, request, render_template
import numpy as np
import joblib
from sklearn import datasets
iris = datasets.load_iris()

# Initialize Flask app
app = Flask(__name__)

# Load the pre-trained model
model = joblib.load(r"D:\DataGlacier\DataGlacierInternship\Week4\iris_model.pkl")
```

You, 49 minutes ago • Uncommitted changes

```python
# Define a route for the home page
@app.route('/')
def home():
    return render_template('index.html')

#Define a route for prediction
@app.route('/predict', methods=['POST'])
def predict():
    # Get input from the form
    sepal_length = float(request.form['sepal_length'])
    sepal_width = float(request.form['sepal_width'])
    petal_length = float(request.form['petal_length'])
    petal_width = float(request.form['petal_width'])

    # Make prediction using the model
    input_data = np.array([[sepal_length, sepal_width, petal_length, petal_width]])
    prediction = model.predict(input_data)

    # Map prediction to class name
    species = iris.target_names[prediction[0]]

    return render_template('result.html', species=species)

if __name__ == '__main__':
    app.run()
```

App Building-html directories

Next we have to create two simple html pages to, one to receive values and the other to give the prediction. Let's break down each file:

index.html:
Defines the main structure of the homepage.
Includes a title (Iris Flower Prediction) and a heading (<h1>).
Provides a form with four input fields for sepal and petal measurements:
Each field has a clear label and accepts numerical input with any decimal precision (step="any").
A submit button allows users to send their input for prediction.

predict.html:
Defines the structure of the result page.
Includes a title (Prediction Result) and a heading (<h2>).
Uses Jinja2 template syntax to display the predicted species name dynamically using the {{ species }} variable passed from the Flask app.
Overall, these HTML files provide a user-friendly interface for your web application, allowing users to easily input flower measurements and see the predicted species in a clear and concise format.

index.html U

templates > index.html > html > body > form > input

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Iris Flower Prediction</title>
</head>
<body>
    <h1>Iris Flower Prediction</h1>
    <form action="/predict" method="post">
        <label for="sepal_length">Sepal Length:</label>
        <input type="number" step="any" name="sepal_length"><br>
        <label for="sepal_width">Sepal Width:</label>
        <input type="number" step="any" name="sepal_width"><br>
        <label for="petal_length">Petal Length:</label>
        <input type="number" step="any" name="petal_length"><br>
        <label for="petal_width">Petal Width:</label>
        <input type="number" step="any" name="petal_width"><br>
        <button type="submit">Predict</button>
    </form>
</body>
</html>

```

result.html U

templates > result.html > ...

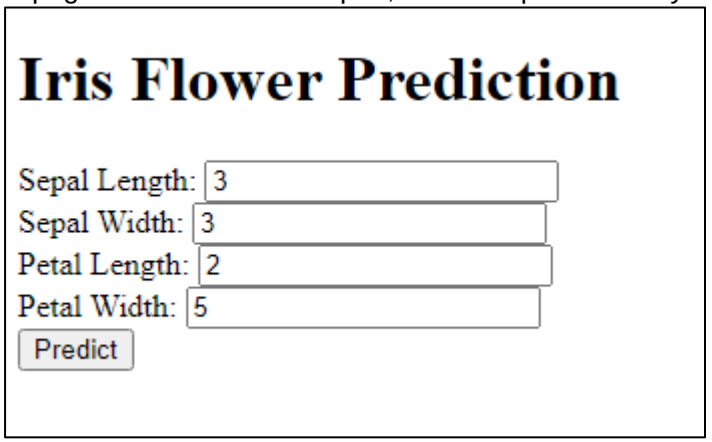
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Prediction Result</title>
</head>
<body>
    <h2>The predicted species is: {{ species }}</h2>
</body>
</html>
```

Deployment

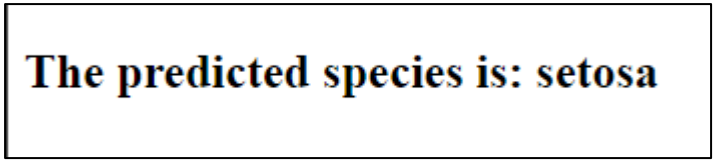
To deploy the app you simply run the flask app code from your IDE of choice, in your terminal a local host will pop up, you can copy that to your browser or simply (CTRL + click)

```
[Running] python -u "d:\DataGlacier\flaskapp\app.py"
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

A page like below should open, enter the parameters you want to predict the flower type for



A prediction like the following image should appear



Thank You