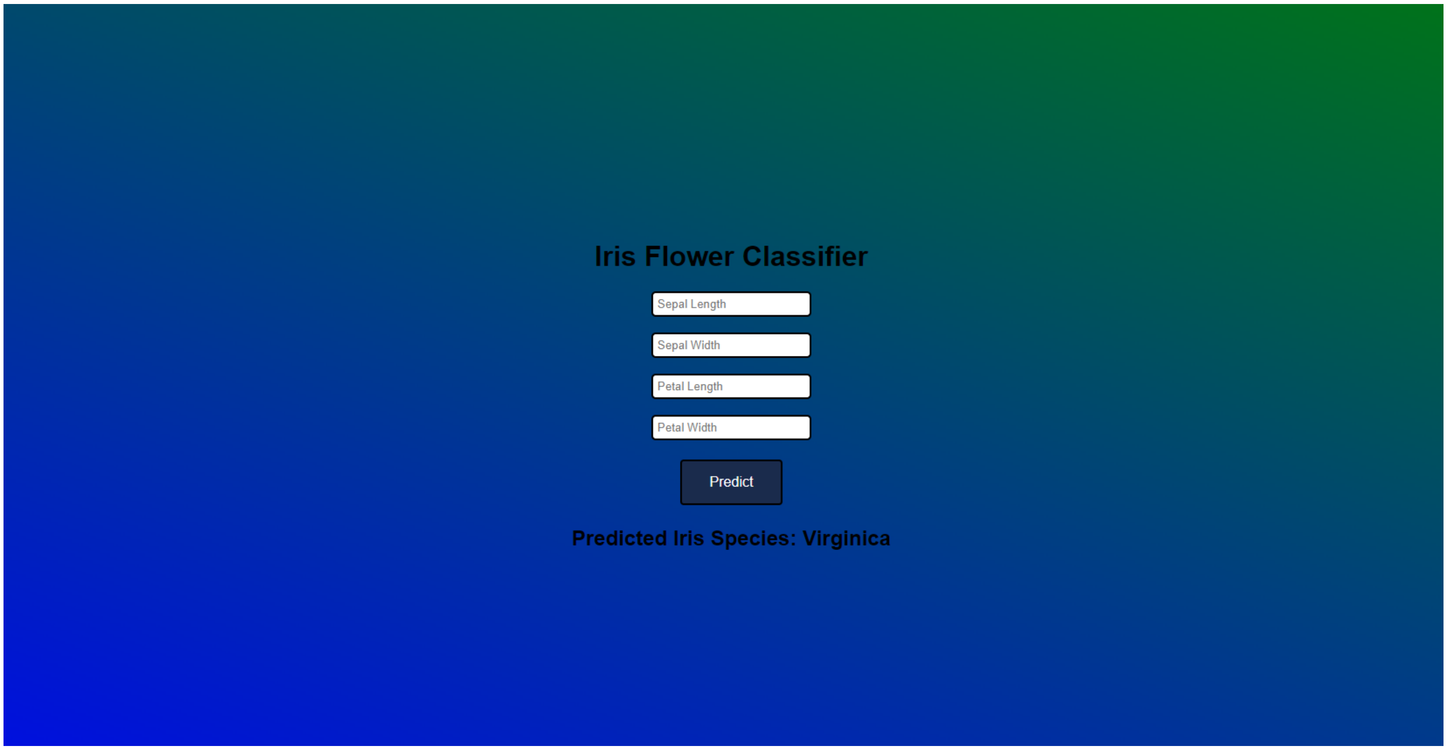
Flask Model Deployment Assignment Documentation

Noah Gallego



Iris Flower Classifier

Predicted Iris Species: Virginica

Prepared for DataGlacier

Batch Code: LISUM26

Date: 10/28/2023

Source Code

app.py

```
from flask import Flask, render_template, request
import pickle
import numpy as np
import os
print("Current working directory:", os.getcwd())
app = Flask(__name__, template_folder='templates')

# Load the model
with open('iris_model.pkl', 'rb') as f:
    model = pickle.load(f)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict', methods=['POST', 'GET'])
def predict():
    int_features = [float(x) for x in request.form.values()]
    print('Received input:', int_features)
    final_features = [np.array(int_features)]
    print('Final input:', final_features)
    prediction = model.predict(final_features)

    iris_names = ['Setosa', 'Versicolor', 'Virginica']
    predicted_name = iris_names[int(prediction[0])]

    return render_template('index.html', prediction_text='Predicted Iris Species:
{}'.format(predicted_name))

if __name__ == "__main__":
    app.run(port = 5000, debug=True)
```

train_model.py

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
import pickle

# Load the data
iris = load_iris()
X, y = iris.data, iris.target
```

```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Instantiate a decision tree classifier
clf = DecisionTreeClassifier(random_state=42)

# Fit the model to the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = clf.score(X_test, y_test)
print(f"Accuracy: {accuracy}")

# Save the model to a file
with open('iris_model.pkl', 'wb') as f:
    pickle.dump(clf, f)
```

index.html

```html
<!DOCTYPE html>
<html lang = "en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Iris Flower Classifier</title>
    <style>
      body {
        background-image: linear-gradient(to left bottom, green, blue);
        background-size: cover;
        font-family: Arial, sans-serif;
        display: flex;
        flex-direction: column;
        justify-content: center;
        align-items: center;
        min-height: 100vh;
      }

      form, h1 {
        text-align: center;
      }

      form {
        margin: 0 auto;
        max-width: 400px;
      }
      .result {
        text-align: center;
        font-size: 24px;
        font-weight: bold;
        margin-top: 20px;
      }

      button[type="submit"] {
        background-color: #1A2B4C;
        border: none;
        border-radius: 4px;
        color: white;
        padding: 15px 32px;
        text-align: center;
        text-decoration: none;
        display: inline-block;
        font-size: 16px;
        margin: 4px 2px;
        cursor: pointer;
        border: 2px solid black;
      }

      input[type="text"] {
        padding: 5px;
        color: black;
        background-color: white;
        border-radius: 5px;
        border: none;
        border: 2px solid black;
      }

    </style>
  </head>
  <body>
    <h1>Iris Flower Classifier</h1>
    <form method="POST" action="/predict">
      <label for="sepal_length"></label>
      <input type="text" id="sepal_length" name="sepal_length" placeholder="Sepal Length"><br><br>
      <label for="sepal_width"></label>
      <input type="text" id="sepal_width" name="sepal_width" placeholder="Sepal Width"><br><br>
      <label for="petal_length"></label>
      <input type="text" id="petal_length" name="petal_length" placeholder="Petal Length"><br><br>
      <label for="petal_width"></label>
      <input type="text" id="petal_width" name="petal_width" placeholder="Petal Width"><br><br>
      <button type="submit">Predict</button>
    </form>
    <p class="result">{{ prediction_text }}</p>
  </body>
</html>
```

style.css

```
body {
    background-image: url("background.jpg");
    background-size: cover;
    font-family: Arial, sans-serif;
}

form, h1 {
    text-align: center;
}

form {
    margin: 0 auto;
    max-width: 400px;
}

.result {
    text-align: center !important;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 50vh;
}

button[type="submit"] {
    background-color: #4CAF50;
    border: none;
    color: white;
    padding: 15px 32px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    margin: 4px 2px;
    cursor: pointer;
}
```

1. Pick Data

- The Iris dataset is a widely used dataset in machine learning and is available through libraries like scikit-learn.
- The Iris dataset contains data on three species of iris flowers: setosa, versicolor, and virginica. Each sample includes four features: sepal length, sepal width, petal length, and petal width.
- The dataset consists of 150 samples and 4 features.
- No significant data preprocessing was performed as the Iris dataset is well-cleaned and commonly used in its original form.
- Basic data exploration was conducted, including scatter plots to visualize feature distributions and species separability.

2. Model Training

Importing Libraries:

I started by importing the necessary libraries at the beginning of my script. This included load_iris to load the Iris dataset, DecisionTreeClassifier to create a decision tree model, and pickle to save the trained model to a file.

Loading the Iris Dataset:

I loaded the Iris dataset using load_iris(). This function returned a dictionary-like object containing the dataset's data and target values.

iris.data contained the feature data, which included sepal length, sepal width, petal length, and petal width.

iris.target contained the target values, representing the species of iris flowers (setosa, versicolor, virginica).

Creating and Training the Model:

I created a DecisionTreeClassifier model by instantiating it with DecisionTreeClassifier(). This is a classification algorithm used for creating decision trees.

The next step was to train the model. I used the model.fit(data, target) method, where data represented the feature data (sepal and petal measurements) and target represented the corresponding species labels.

The model was trained to learn patterns in the data and make predictions based on these patterns.

Saving the Trained Model:

Once the model was trained, I saved it using pickle.dump(model, 'iris_model.pkl'). This line of code saved the trained decision tree model to a file named 'iris_model.pkl' using the pickle library. The saved model could be loaded and used for making predictions in my Flask application.

3. Flask Application

Creating the Flask Application:

I created a Flask application by importing the Flask class from the Flask library and initializing it with app = Flask(__name__). This prepared the foundation for my web-based deployment.

Loading the Pre-trained Model:

To make predictions, I loaded the pre-trained machine learning model using model = pickle.load('iris_model.pkl'). This model was saved after training in a previous step and contains the knowledge to make predictions.

4. API Endpoint

Setting Up the API Endpoint:

I set up an API endpoint at /predict using the @app.route decorator. This endpoint would handle both GET and POST requests, making it accessible for receiving data and returning predictions.

Receiving and Processing Data:

In the /predict route, I retrieved incoming data in JSON format using data = request.get_json(). This data typically included the input features required for making predictions.

Making Predictions:

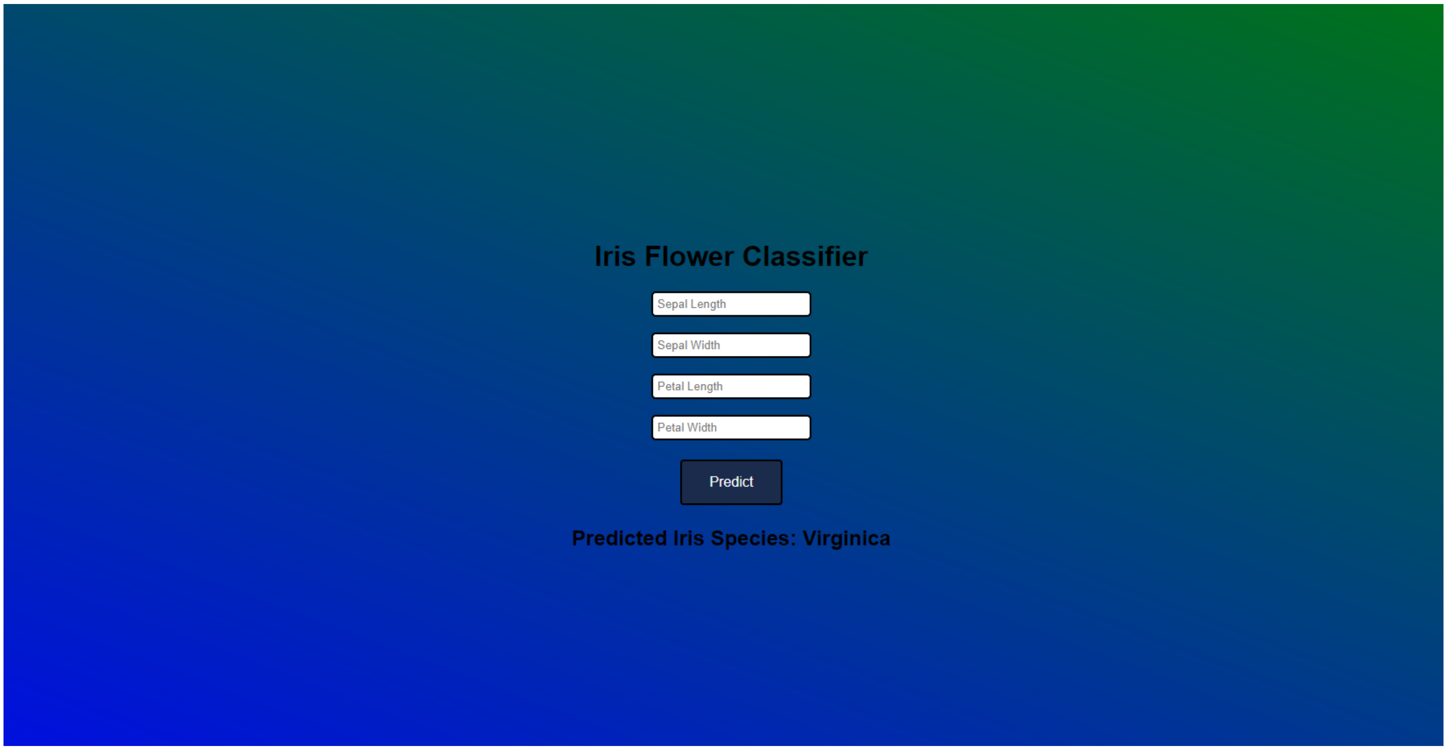
I extracted the input data from the JSON request using input_data = data['data']. This data was then used as input for the pre-trained machine learning model to make predictions.

Running the Flask Application:

I ran the Flask application locally using if __name__ == '__main__': app.run(host='localhost', port=5000). This started the web service, making it available for receiving requests on the specified host and port (localhost:5000).

5. User Interface

I then made a simple HTML webpage that was designed as a form. The form took in various data from the user designed to classify which species of iris. I made a paragraph element at the bottom of the page that displays the predicted species upon the button click (submission).



Iris Flower Classifier

Predicted Iris Species: Virginica