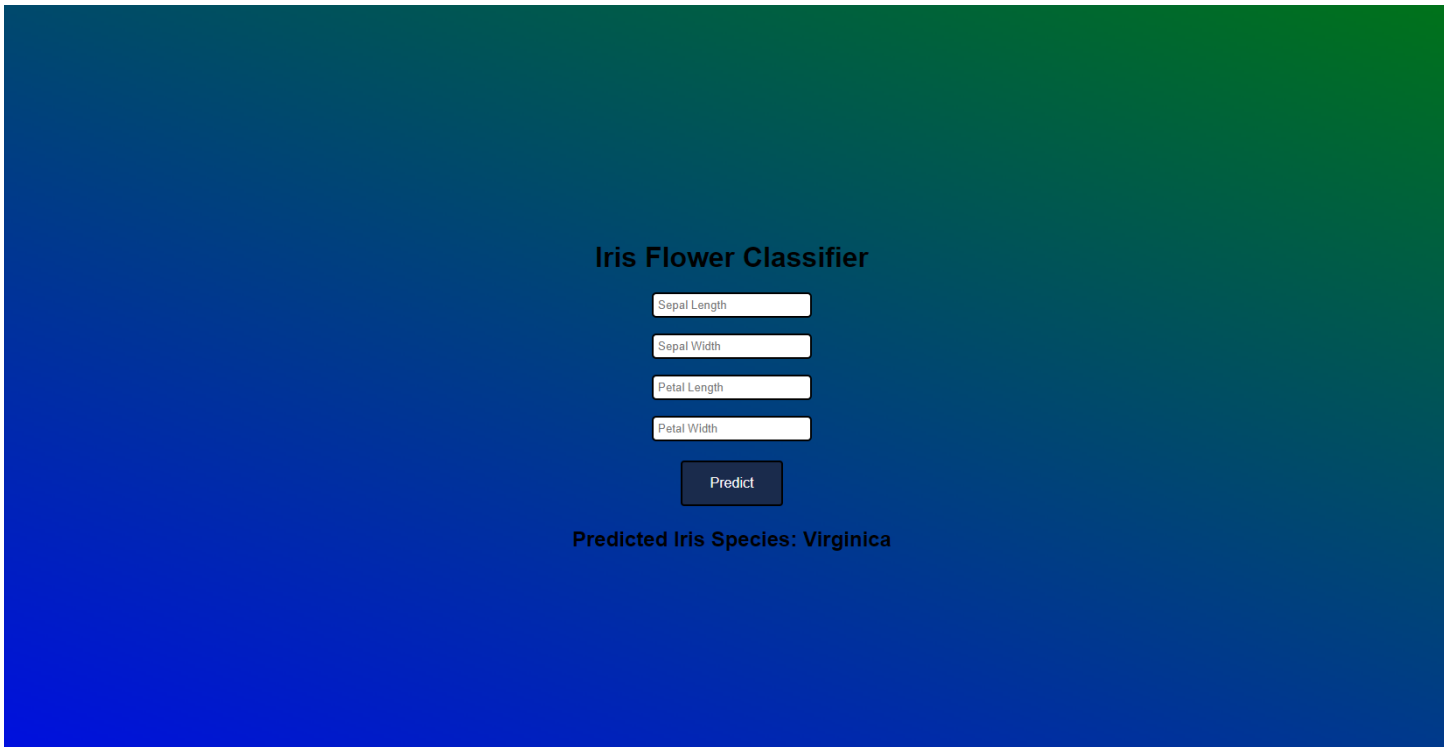
Flask Model Deployment Assignment Documentation

Noah Gallego



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
```

```
# 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

```
<!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. Use pip freeze

Using pip freeze, I was able to see the version requirements of all libraries needed to run my ML model. This is necessary for Heroku to know.

2. Published to Github with Procfile file

I also added a Procfile file to tell Heroku that we are deploying a Python-based web app.

3. Linking Github Account

I then went onto Heroku, linked a payment method, and then linked my github account in the 'deploy' section of my heroku deployment.

4. Fixing Bugs

I then fixed some small bugs that were preventing my application from running, such as removing a default port (since Heroku chooses a port when running a program) and other small bugs. And voila, we have my Flask deployment on the web!