



Data Glacier

Your Deep Learning Partner

Cloud and API deployment

Brennan Clinch

Data Science Intern at Data Glacier

Batch Code: LISUM12

Date: 5 September 2022

Introduction

For this project, we will be deploying a decision tree classifier model using Python's Flask API. We are wanting to build a model that will predict and classify a type of Iris flower based on certain characteristics.

Dataset Information

- Our dataset is a toy dataset included in Python and R called 'iris'. This dataset has 150 observations and 5 variables.
- The 5 variables are:
- Sepal length: Sepal length in cm
- Sepal width: Sepal width in cm
- Petal length: Petal length in cm
- Petal width: Petal width in cm
- Class: Type of variety of Iris flower. 3 classes: ("Iris-setosa, Iris-virginica, Iris-versicolor")

Import Libraries and Dataset

- We first import the appropriate libraries in Python along with the Iris dataset from the UCI Machine Learning Repository.

```
import pickle
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

```
# Load Iris Dataset
# Load through url
url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
attributes = ["sepal_length", "sepal_width", "petal_length", "petal_width", "class"]
iris = pd.read_csv(url, names = attributes)
iris.columns = attributes
iris.head(10)
```

Data Preprocessing

- We split the dataset into 70% for the training set and 20% for the test set. We also set up the appropriate feature columns and target variables. The feature columns are the attributes of the Iris flower such as sepal width, sepal length, petal width, and petal length. The target (response) variable is the type (class) of the Iris flower. It can be Iris-Setosa, Iris-Virginica, or Iris-Versicolor.

```
# Split up data for test and training before doing EDA on training set  
iris_train, iris_test = train_test_split(iris, test_size = 0.3, stratify = iris['class'], random_state = 3)
```

```
X_train = iris_train[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]  
Y_train = iris_train['class']  
  
X_test = iris_test[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]  
Y_test = iris_test['class']
```

Model Build

- After preprocessing the data, we build a machine learning model to classify the Iris flower based on the feature variables. For this model, we will use a Classification Tree since we want to classify the type of Iris flower.
- Accuracy looks pretty good for the training and test sets.

```
# Classification Tree
from sklearn.tree import DecisionTreeClassifier, plot_tree
DT_model = DecisionTreeClassifier(max_depth = 3, random_state = 5)
DT_model = model.fit(X_train,Y_train)
model_predict_train = DT_model.predict(X_train)
model_predict = DT_model.predict(X_test)
print('Accuracy of Test Set:', "{:.3f}".format(metrics.accuracy_score(model_predict,Y_test)))
print('Accuracy of Train Set:', "{:.3f}".format(metrics.accuracy_score(model_predict_train,Y_train)))
```

```
Accuracy of Test Set: 0.911
Accuracy of Train Set: 0.981
```

Save Model

- After building our model, we use the pickle serialization library to save it.

```
pickle.dump(DT_model, open('Treemodel.pickle', 'wb'))
```

Creating a web application

- Next, we will create a web application of our model using Python's Flask API. The way the information and files are stored is shown below. We will discuss the different files next and their purpose for the app.

```
App.py
templates/
    index.html
static/
    css/
        main.css
model/
    Treemodel.pickle
```


App.py

- This file contains the main code to be executed that will be ran in the Flask web app. The serialized model was then unserialized here to be used in production in Flask.

```
App.py x
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[7]:
5
6
7  from flask import Flask, request, render_template
8  import numpy as np
9  import pickle
10 import pandas as pd
11
12 app = Flask(__name__, template_folder = 'templates')
13 model = pickle.load(open('Model/Treemodel.pickle', 'rb'))
14
15 @app.route('/', methods=['GET', 'POST'])
16 def home():
17     return render_template('index.html')
18 @app.route('/predict', methods = ['POST'])
19 def predict():
20     '''
21     For rendering results on html GUI
22     '''
23     float_features = [float(x) for x in request.form.values()]
24     final_features = [np.array(float_features)]
25     prediction = model.predict(final_features)
26     output = prediction[0]
27     return render_template('index.html', prediction_text = 'Class should be {}'.format(output))
28 if __name__ == "__main__":
29     app.run(port = 5000, debug=True)
```

index.html

- This file stores the information necessary to write the application created with Flask, such as text and font along with other features.

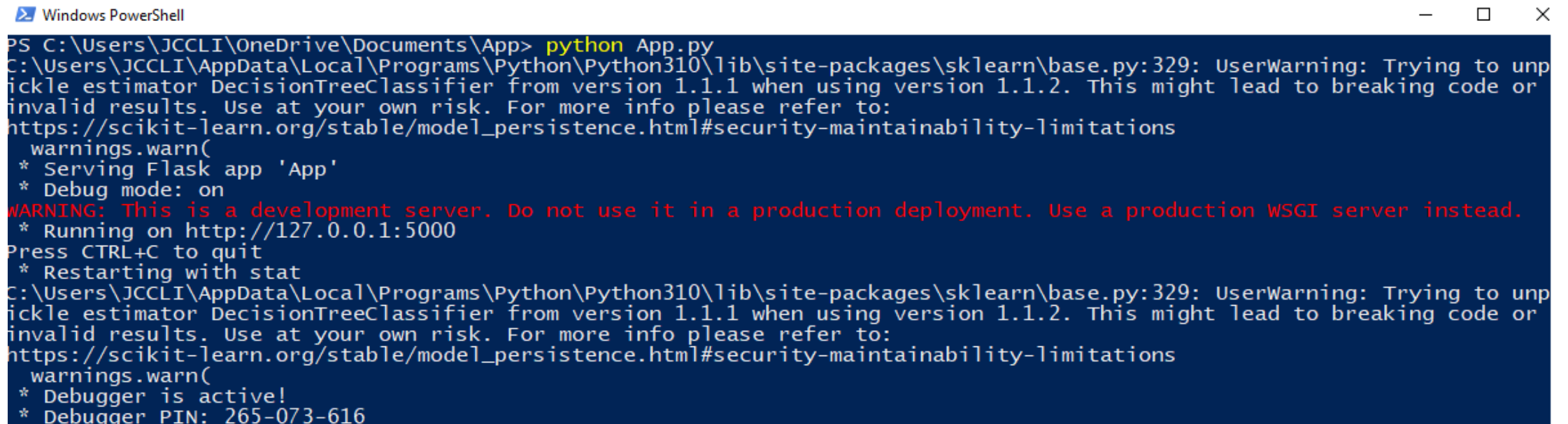
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4
5     <link rel="stylesheet" type = "text/css" href="{{ url_for('static', filename='css/main.css') }}">
6 </head>
7 <body>
8     <div class="container">
9         <h1>Iris Classification</h1>
10        <p>This model uses information from the iris dataset to classify the class of iris plant based on several feature characteristics.
11        </p>
12        <div class='Login'>
13            <form action="{{ url_for('predict') }}" method="post">
14                <input type="text" class='form-control' name="sepal_length" placeholder="Sepal Length (cm)" required="required" /><br>
15                <input type="text" class='form-control' name="sepal_width" placeholder="Sepal Width (cm)" required="required" /><br>
16                <input type="text" class='form-control' name="petal_length" placeholder="Petal Length (cm)" required="required" /><br>
17                <input type="text" class='form-control' name="petal_width" placeholder="Petal Width (cm)" required="required" /><br>
18
19                <button type="submit" class='btn btn-primary btn-block btn-Large'>Predict</button>
20            </form>
21        </div>
22        <br>
23        <br>
24        <h1>{{ prediction_text }}</h1>
25    </div>
26 </body>
27 </html>
```

main.css

- This file is used to create the styling of the web app (color, theming, etc).

Running the web application

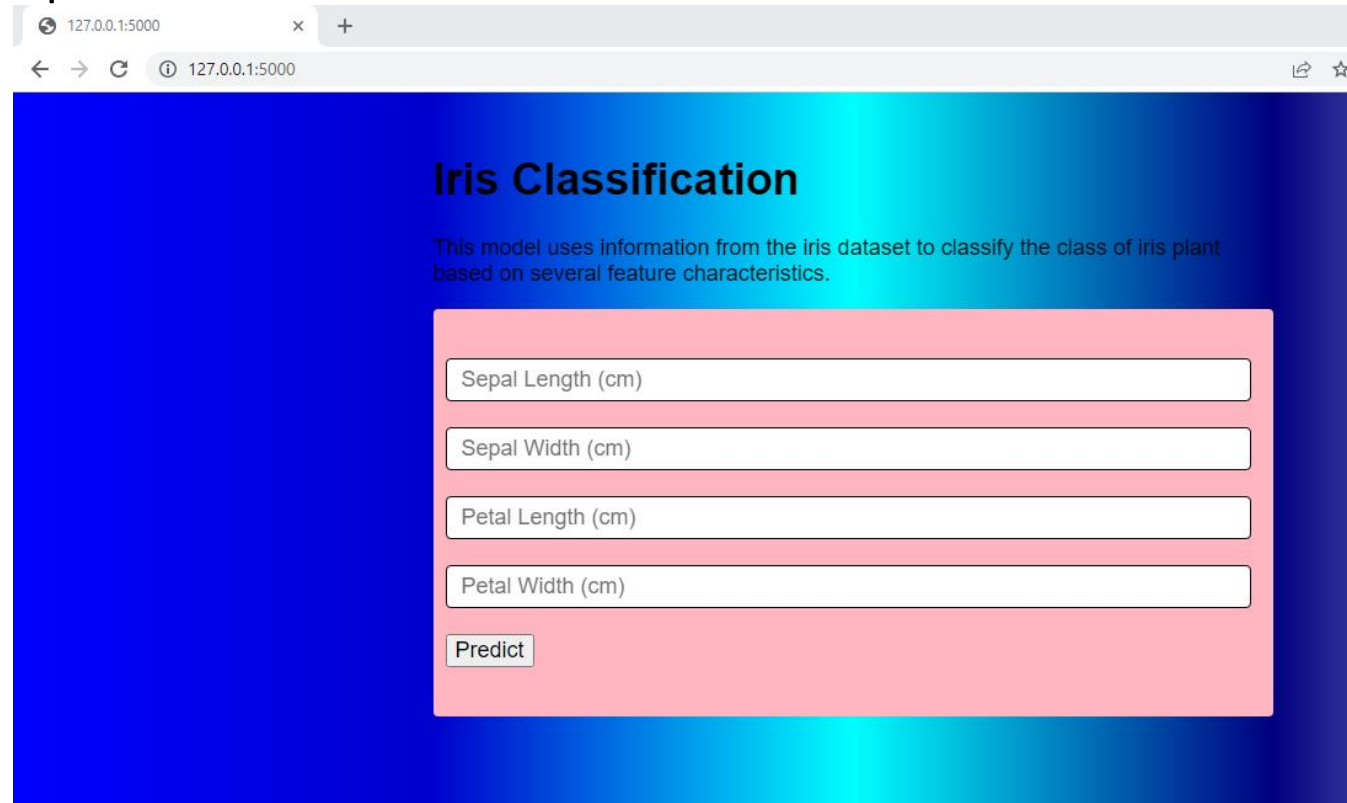
- We are now ready to run the app. To do this, we open a command prompt (I used PowerShell) or run the app by double clicking on the python file App.py. To get to our app, we must go to the url specified in the command prompt (<http://127.0.0.1:5000>).



```
Windows PowerShell
PS C:\Users\JCCLI\OneDrive\Documents\App> python App.py
C:\Users\JCCLI\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:329: UserWarning: Trying to unpickle estimator DecisionTreeClassifier from version 1.1.1 when using version 1.1.2. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
* Serving Flask app 'App'
* Debug mode: on
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
* Restarting with stat
C:\Users\JCCLI\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:329: UserWarning: Trying to unpickle estimator DecisionTreeClassifier from version 1.1.1 when using version 1.1.2. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
* Debugger is active!
* Debugger PIN: 265-073-616
```

Running the web application

- After going to the url specified, our app page should look like this. We can input numeric values (decimals too) into the 4 boxes and then click on the predict button to classify the type of Iris based on our inputs.



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5000'. The page has a blue gradient background and is titled 'Iris Classification'. Below the title, a subtitle reads: 'This model uses information from the iris dataset to classify the class of iris plant based on several feature characteristics.' The main content area is a light pink box containing four input fields and a button. The input fields are labeled 'Sepal Length (cm)', 'Sepal Width (cm)', 'Petal Length (cm)', and 'Petal Width (cm)'. Below these fields is a button labeled 'Predict'.

Iris Classification

This model uses information from the iris dataset to classify the class of iris plant based on several feature characteristics.

Sepal Length (cm)

Sepal Width (cm)

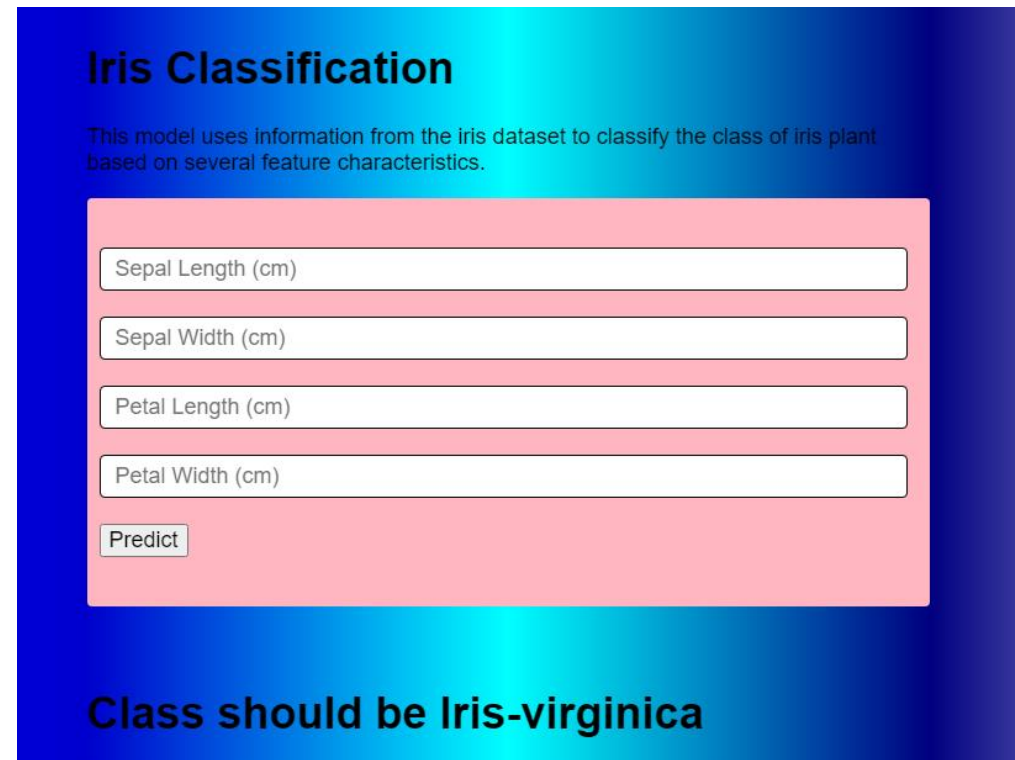
Petal Length (cm)

Petal Width (cm)

Predict

Running the web application

- Let's do a sample prediction. Let Sepal Length = 0.4, Sepal Width = 1.5, Petal Length = 3.3, and Petal Width = 1.7. After pushing the predict button, our classified Iris flower is Iris-virginica. We tested other values and the model works successfully by classifying other Iris flowers based on what we input.



The screenshot shows a web application titled "Iris Classification" on a blue background. Below the title is a subtitle: "This model uses information from the iris dataset to classify the class of iris plant based on several feature characteristics." The main input area is a light pink rectangle containing four text input fields stacked vertically, each with a label: "Sepal Length (cm)", "Sepal Width (cm)", "Petal Length (cm)", and "Petal Width (cm)". Below these fields is a "Predict" button. At the bottom of the interface, a dark blue banner displays the text "Class should be Iris-virginica" in white.

Deploying the app on Heroku (cloud)

- We are now ready to deploy our app to the cloud. We will be using the free open source cloud platform Heroku
- Before we can deploy our model, we must have the following files installed in github.
 - **Requirements.txt:** This is a text file listing all the files and packages needed to run our application.
 - **Procfile:** This file tells Heroku commands to be ran when we start the app. The Gunicorn command connects the Flask app to the HTTP server.
- We name our app Iris-deployment-demo.

The screenshot shows the Heroku dashboard. At the top, there's a navigation bar with the Heroku logo, a search bar with the text "Jump to Favorites, Apps, Pipelines, Spaces...", and a user profile icon. Below this, a secondary bar shows "Personal" with a dropdown arrow and a "New" button. The main content area features a purple banner with the Heroku logo, the text "Welcome to Heroku" and "Now that your account has been set up, here's how to get started.", a "Dismiss" button, and a link to "Show next steps". Below the banner is a light blue notification box stating that free Heroku Dynos, Postgres, and Data for Redis will no longer be available starting November 28th, 2022, and providing a link to learn more. At the bottom, there's a search bar for apps and pipelines, and a list of apps with "iris-deployment-demo" selected. The footer shows the selected app's technology stack: Python, heroku-22, and United States, along with a star icon.

HEROKU

Jump to Favorites, Apps, Pipelines, Spaces...

Personal

New

Welcome to Heroku
Now that your account has been set up, here's how to get started.

Dismiss

Show next steps

Starting November 28th, 2022, free Heroku Dynos, free Heroku Postgres, and free Heroku Data for Redis* will no longer be available.
If you have apps using any of these resources, you must upgrade to paid plans by this date to ensure your apps continue to run and to retain your data. For students, we will announce a new program by the end of September. [Learn more](#)

Filter apps and pipelines

iris-deployment-demo

Python · heroku-22 · United States

Deploying the app on Heroku (cloud)

We now configure our app by connecting to github.

The screenshot shows the Heroku dashboard interface. At the top, there's a navigation bar with the Heroku logo, a search bar containing "Jump to Favorites, Apps, Pipelines, Spaces...", and a user profile icon. Below the navigation bar, the main content area is divided into two columns. The left column is titled "Add this app to a pipeline" and contains instructions: "Create a new pipeline or choose an existing one and add this app to a stage in it." The right column is titled "Add this app to a stage in a pipeline to enable additional features" and contains two sections. The first section explains that pipelines let you connect multiple apps together and promote code between them, with a "Learn more" link. The second section explains that pipelines connected to GitHub can enable review apps and create apps for new pull requests, also with a "Learn more" link. Below these sections is a dropdown menu labeled "Choose a pipeline". At the bottom of the dashboard, there's a section titled "Deployment method" which shows three options: "Heroku Git Use Heroku CLI", "GitHub Connected" (with a green checkmark), and "Container Registry Use Heroku CLI". Below this, there's a section titled "App connected to GitHub" which states: "Code diffs, manual and auto deploys are available for this app." To the right of this text, there's a box showing the connection details: "Connected to Brennan-Clinch/iris-Flask-Heroku-deployment by Brennan-Clinch" with a "Disconnect..." button. Below this box, there's a link to the activity feed: "Releases in the activity feed link to GitHub to view commit diffs".

HEROKU

Jump to Favorites, Apps, Pipelines, Spaces...

Add this app to a pipeline

Create a new pipeline or choose an existing one and add this app to a stage in it.

Add this app to a stage in a pipeline to enable additional features

Pipelines let you connect multiple apps together and **promote code** between them. [Learn more.](#)

Pipelines connected to GitHub can enable **review apps**, and create apps for new pull requests. [Learn more.](#)

Choose a pipeline

Deployment method

Heroku Git
Use Heroku CLI

GitHub
Connected

Container Registry
Use Heroku CLI

App connected to GitHub

Code diffs, manual and auto deploys are available for this app.

Connected to [Brennan-Clinch/iris-Flask-Heroku-deployment](#) by [Brennan-Clinch](#) [Disconnect...](#)

Releases in the [activity feed](#) link to GitHub to view commit diffs

Deploying the app on Heroku (cloud)

- After connecting to github, go to the option named “Manual Deploy” and click “Deploy Branch”. If the github contains all the necessary files (requirements.txt, and Procfile) then the app should run successfully.

Manual deploy

Deploy the current state of a branch to this app.

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. [Learn more](#).

Choose a branch to deploy

 main



Deploy Branch