



Data Glacier

Your Deep Learning Partner

Deployment on Flask

Brennan Clinch

Data Science Intern at Data Glacier

Batch Code: LISUM12

Date: 25 August 2022

Submitted To: Data Glacier

Introduction

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

Model Building: Import Libraries & 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)
```

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")

Model Building: 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 Building: Build Model

- 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. The accuracy looks good enough so we are ready to save it.

```
# 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
```

Model Building: Save Model

After building our model, we use 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 package. 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
```


Creating a web application

- 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)
```

Creating a web application

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

```
index.html
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>
```

Creating a web application

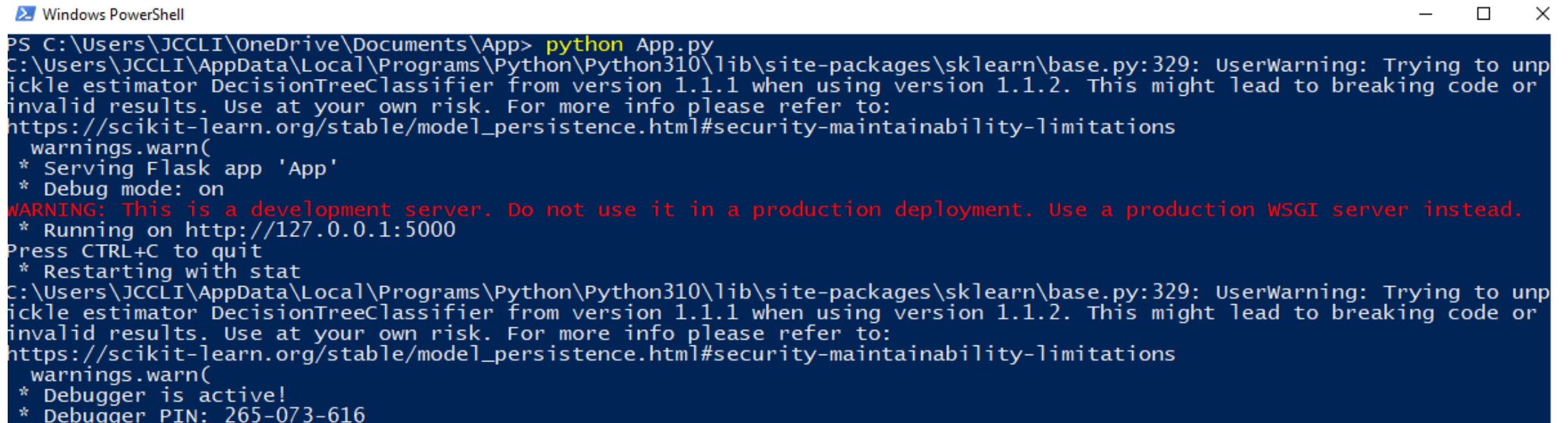
- main.css: This file is used to create the styling of the web app (color, theming, etc)

```
index.html x main.css x
1  *::after{
2    box-sizing: border-box;
3  }
4  body{
5    font-family: helvetica;
6    font-size: 18px;
7    background: repeating-linear-gradient(
8      90deg,
9      rgb(0, 0, 255),
10     rgb(0, 0, 205),
11     rgb(0, 255, 255),
12     rgb(0, 0, 128),
13     rgb(255, 255, 255));
14    margin: 0;
15  }
16
17  body::before {
18    content: '';
19    position: fixed;
20    top: 0;
21    left: 0;
22    height: 100%;
23    width: 100%;
24    z-index: -1;
25    background-size: cover;
26    background-position: center;
27  }
28  .container{
29    width: 50%;
30    margin: 3.125rem auto 0 auto;
31  }
32
33  form{
34    background: RGB(255, 182, 193);
35    padding: 2.5rem 0.625rem;
36    border-radius: 0.25rem;
37  }
```

```
38
39  .form-group{
40    margin: 0 auto 1.25rem auto;
41    padding: 0.25rem;
42  }
43
44  label{
45    display: flex;
46    align-items: center;
47    margin-bottom: 0.5rem;
48  }
49
50  input,
51  button,
52  select,
53  textarea{
54    margin: 0;
55    font-family: inherit;
56    font-size: inherit;
57    line-height: inherit;
58  }
59
60  .form-control{
61    display: block;
62    width: 95%;
63    padding: 0.375rem 0.75rem;
64    background-clip: padding-box;
65    border: 1px solid;
66    border-radius: 0.25rem
67  }
68
69  .input-textarea{
70    min-height: 120px;
71    width: 50%;
72    padding: 0.625rem;
73    resize: vertical;
74  }
```

Running the web application

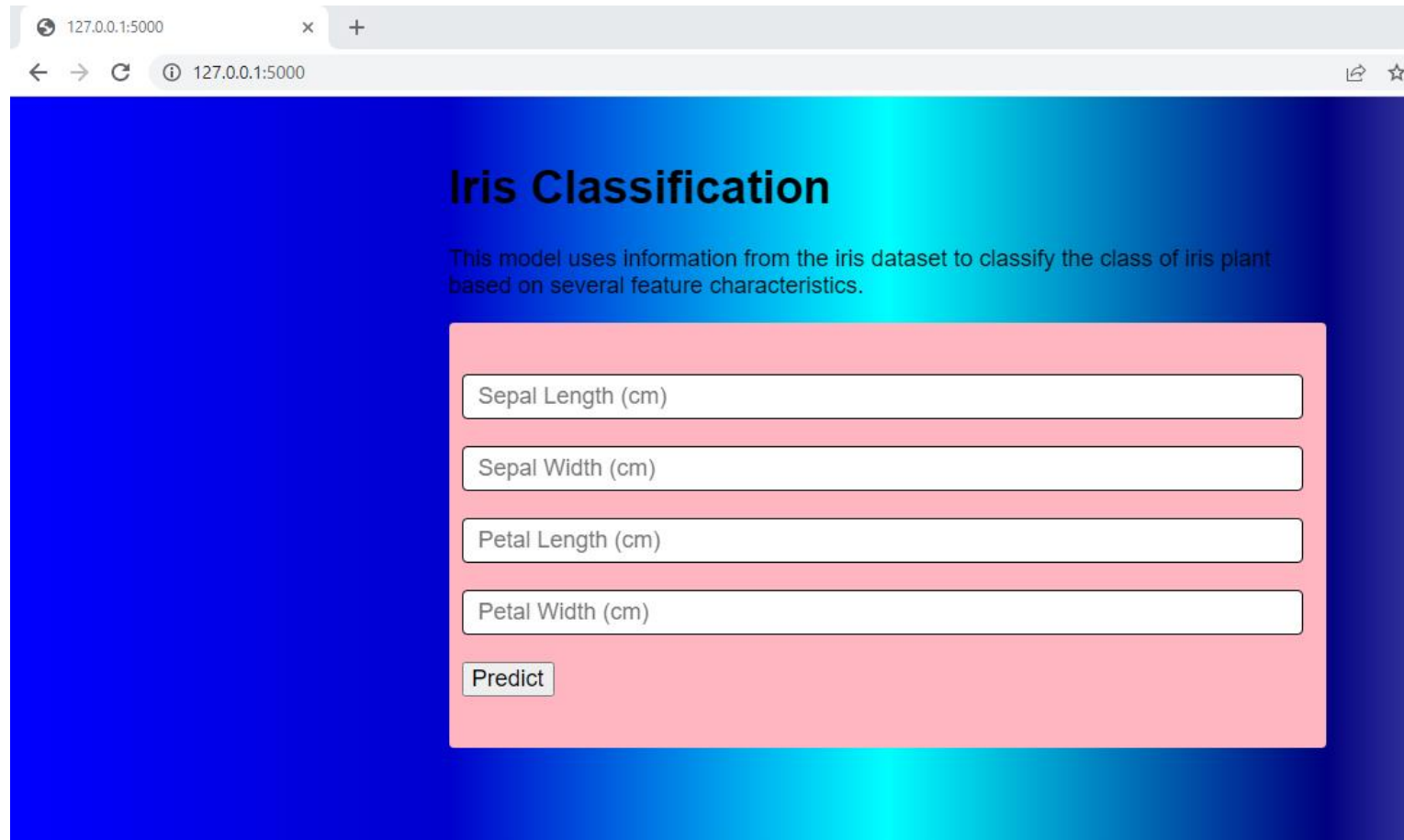
We are now ready to run the app. To do this, we open a command prompt (I used powershell) or run the app by double clicking on the python file App.py. To get to our app, we must go to the url specified in the command prompt (<http://127.0.0.1:5000>).



```
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 a single tab. The address bar displays the URL `127.0.0.1:5000`. The page content has a dark blue background with a bright blue horizontal gradient band. The title "Iris Classification" is displayed in bold black text. Below the title, a descriptive sentence reads: "This model uses information from the iris dataset to classify the class of iris plant based on several feature characteristics." A light pink rectangular box contains 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)

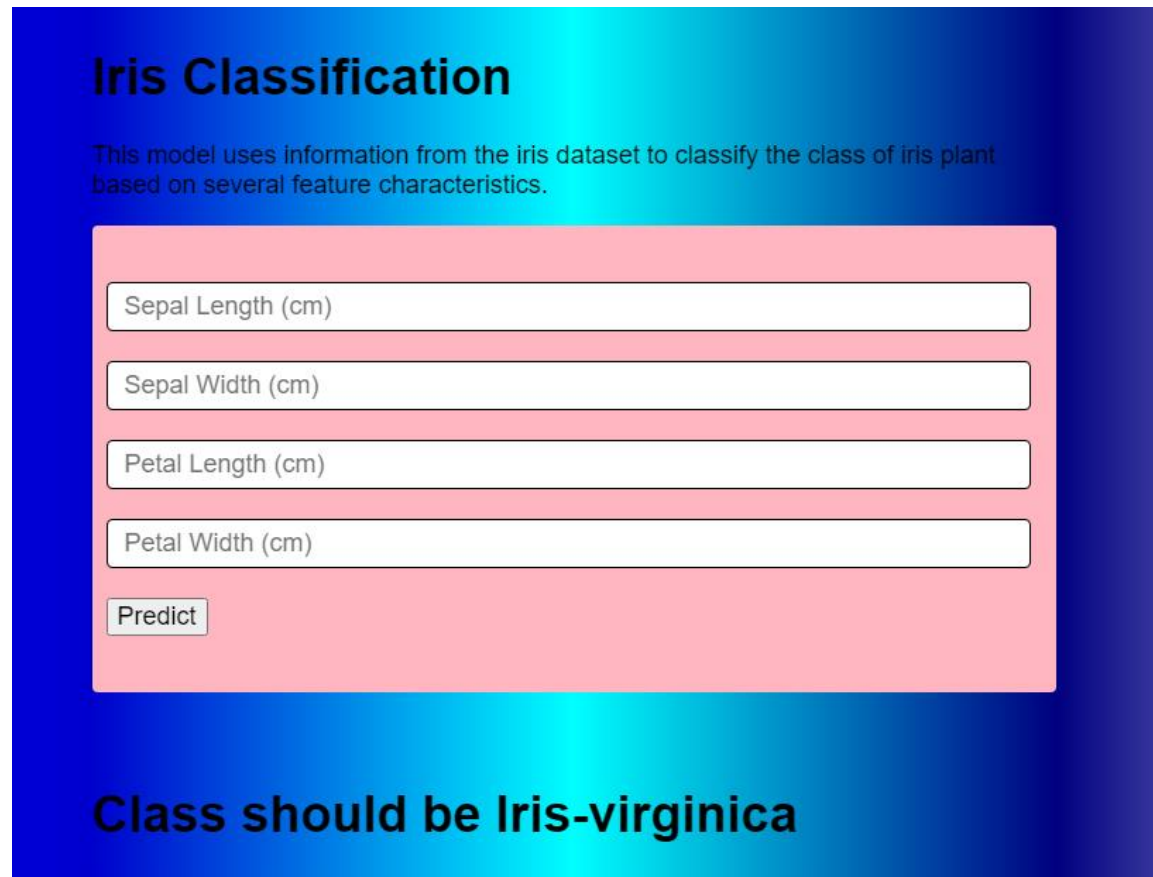
Petal Length (cm)

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 descriptive sentence: "This model uses information from the iris dataset to classify the class of iris plant based on several feature characteristics." The input area is a pink rectangle containing four text boxes labeled "Sepal Length (cm)", "Sepal Width (cm)", "Petal Length (cm)", and "Petal Width (cm)". Below these is a "Predict" button. At the bottom of the interface, a text label states "Class should be Iris-virginica".

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

Class should be Iris-virginica

Uploading Files/Next Steps

- We then uploaded all files to github and stored them in the appropriate folders for the app to run properly.
- We will then deploy the app on the cloud (Heroku) as the next step.