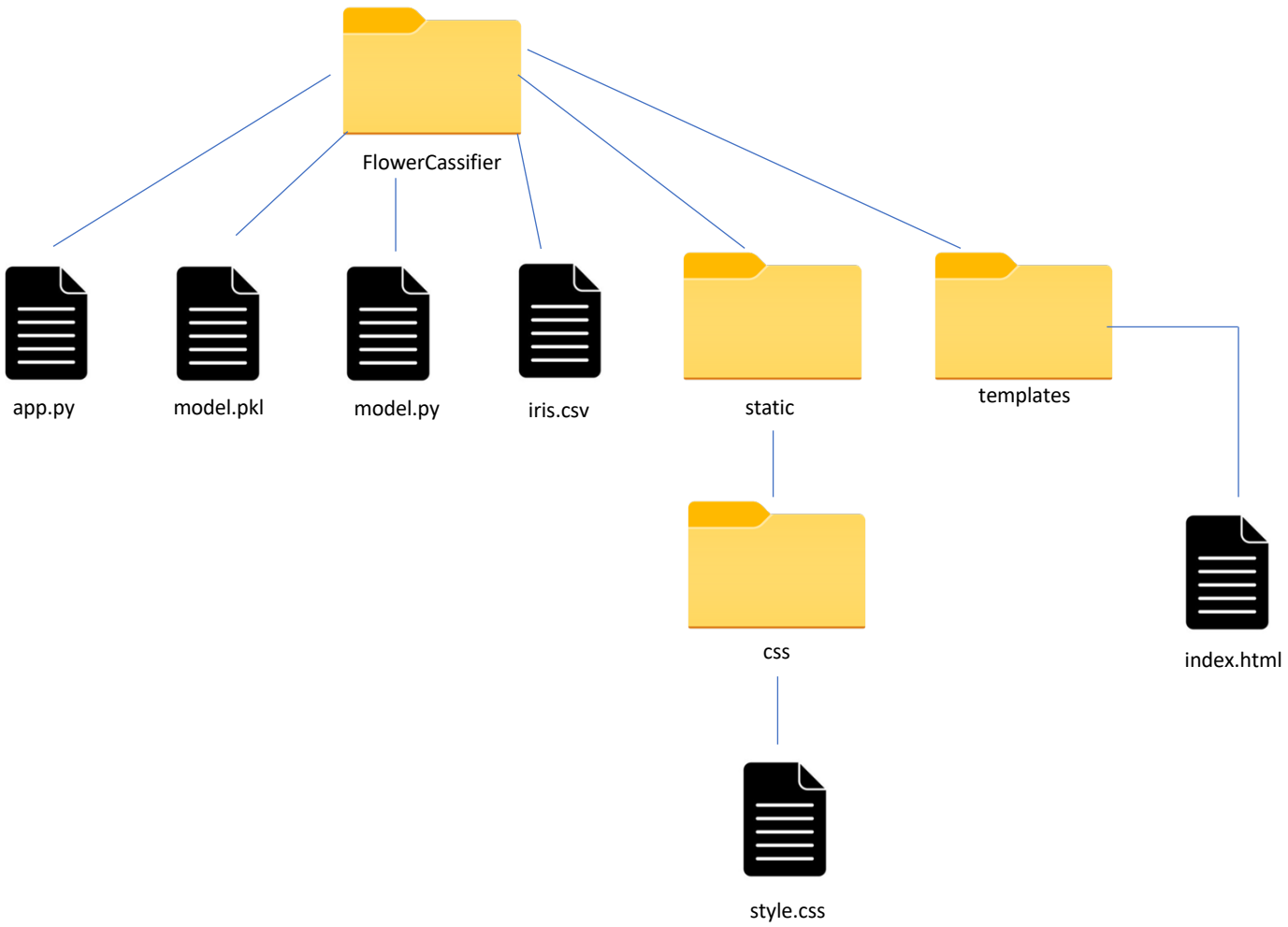
Flask Model Deployment Document

Name: Amogh Vig
Batch code: LISUM17
Submission date: 01/28/2023
Submitted to: https://github.com/avig00/FlowerClassifier

The Application

A simple flower classifier app, called FlowerClassifier, which was built in PyCharm, trained using the iris dataset, and deployed using Flask. The type of model used is KNN.

Overview of Directory Structure



Step 1: The Dataset

The model was trained on a cleaned version of the iris dataset.
Source: https://github.com/siddiquiamir/ML-MODEL-DEPLOYMENT-USING-FLASK/blob/main/iris.csv

Step 2: Writing the model script

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
# from sklearn.model_selection import GridSearchCV
import pickle

# Load data
df = pd.read_csv("iris.csv")

# Select independent and dependent variables
X = df[["Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width"]]
y = df["Class"]

# Split the data set into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Feature scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test= sc.transform(X_test)

# Instantiate the model
classifier = KNeighborsClassifier(leaf_size=1, p=2, n_neighbors=13)

# Fit the model
classifier.fit(X_train, y_train)

# Make pickle file of the model
pickle.dump(classifier, open("model.pkl", "wb"))
```

- The script above was used to fit the KNeighbors Classifier from sklearn to the data set.
- The dependent variable was the class (species) of flower, and the independent variables were sepal length, sepal width, and petal width
- The features were scaled using the StandardScaler from sklearn preprocessing
- The trained model was saved as pickle file (.pkl)

Step 3: HTML and CSS

```html
<!DOCTYPE html>
<html >
<!--From https://codepen.io/frytyler/pen/EGdtg-->
<head>
  <meta charset="UTF-8">
  <title>ML API</title>
  <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">

</head>

<body>
 <div class="login">
   <h1>Flower Species Classifier</h1>

    <!-- Main Input For Receiving Query to our ML -->
   <form action="{{ url_for('predict')}}"method="post">
       <input type="text" name="Sepal_Length" placeholder="Sepal_Length" required="required" />
       <input type="text" name="Sepal_Width" placeholder="Sepal_Width" required="required" />
       <input type="text" name="Petal_Length" placeholder="Petal_Length" required="required" />
       <input type="text" name="Petal_Width" placeholder="Petal_Width" required="required" />

       <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
   </form>

   <br>
   <br>
   {{ prediction_text }}

 </div>


</body>
</html>
```

index.html

```css
@import url(https://fonts.googleapis.com/css?family=Open+Sans);
.btn { display: inline-block; *display: inline; *zoom: 1; padding: 4px 10px 4px; margin-bottom: 0; font-size: 13px; line-height: 18px; color: #333333;
.btn:hover, .btn:active, .btn.active, .btn.disabled, .btn[disabled] { background-color: #e6e6e6; }
.btn-large { padding: 9px 14px; font-size: 15px; line-height: normal; -webkit-border-radius: 5px; -moz-border-radius: 5px; border-radius: 5px; }
.btn:hover { color: #333333; text-decoration: none; background-color: #e6e6e6; background-position: 0 -15px; -webkit-transition: background-position 0
.btn-primary, .btn-primary:hover { text-shadow: 0 -1px 0 rgba(0, 0, 0, 0.25); color: #ffffff; }
.btn-primary.active { color: rgba(255, 255, 255, 0.75); }
.btn-primary { background-color: #4a77d4; background-image: -moz-linear-gradient(top, #6eb6de, #4a77d4); background-image: -ms-linear-gradient(top, #6
.btn-primary:hover, .btn-primary:active, .btn-primary.active, .btn-primary.disabled, .btn-primary[disabled] { filter: none; background-color: #4a77d4;
.btn-block { width: 100%; display:block; }

* { -webkit-box-sizing:border-box; -moz-box-sizing:border-box; -ms-box-sizing:border-box; -o-box-sizing:border-box; box-sizing:border-box; }

html { width: 100%; height:100%; overflow:hidden; }

body {
    width: 100%;
    height:100%;
    font-family: 'Open Sans', sans-serif;
    color: #fff;
    font-size: 18px;
    text-align:center;
    letter-spacing:1.2px;
    background: #3B3B3B !important;
    filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#3E1D6D', endColorstr='#092756',GradientType=1 );

}
.login {
    position: absolute;
    top: 40%;
    left: 50%;
    margin: -150px 0 0 -150px;
    width:400px;
    height:400px;
}

.login h1 { color: #fff; text-shadow: 0 0 10px rgba(0,0,0,0.3); letter-spacing:1px; text-align:center; }

input {
    width: 100%;
    margin-bottom: 10px;
    background: rgba(0,0,0,0.3);
    border: none;
    outline: none;
    padding: 10px;
    font-size: 13px;
    color: #fff;
    text-shadow: 1px 1px 1px rgba(0,0,0,0.3);
    border: 1px solid rgba(0,0,0,0.3);
```

style.css (not full script)

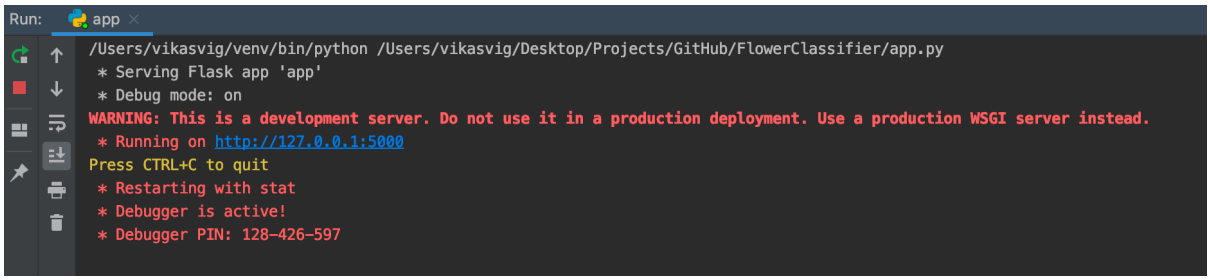
- The scripts above are used to build the webpage where the application is displayed
- The HTML is used to make the webpage title, the form fields where users can input the sizes of different flower features, and the “Predict” button that users click to obtain the classification result
- The CSS is used to add a little style to the webpage, such as setting a background color and displaying the user input form in the middle of the page, which makes the application a little more aesthetic

Step 4: Building the Flask App

```python
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle

# Create flask app
app = Flask(__name__)

# Load model
model = pickle.load(open("model.pkl", "rb"))

# Home page
@app.route("/")
def Home():
    return render_template("index.html")

# Prediction page
@app.route("/predict", methods = ["POST"])
def predict():
    float_features = [float(x) for x in request.form.values()]
    features = [np.array(float_features)]
    prediction = model.predict(features)



    return render_template("index.html", prediction_text = "The flower species is {}".format(prediction))



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

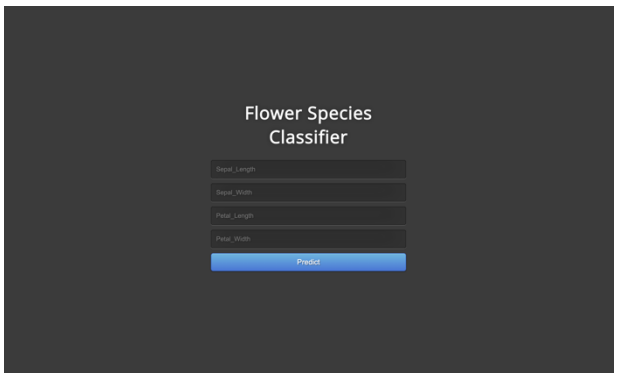
- This script is where the Flask app is actually made and deployed
- The model pickle file is loaded in
- Two pages are made: The Home page and the predict page
- The Home page is what the user sees when they first open the app; it is rendered using the HTML index file
- The Predict page is what the user sees after entering their inputs and clicking the “Predict” button
- After clicking “Predict”, the application runs the model on the user inputs to classify the flower and displays the result to the user

Step 5: Running the App script

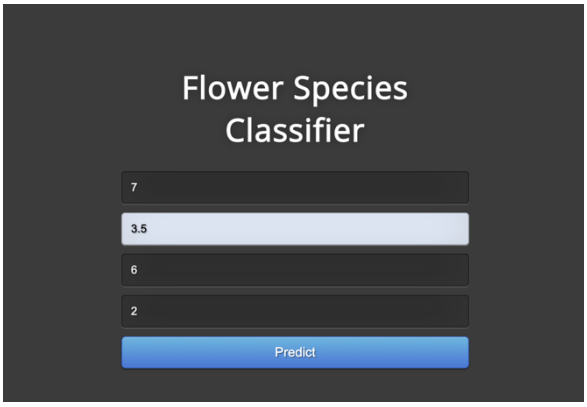


- Running the app script produces this result
- Clicking the link pulls up the webpage in the browser

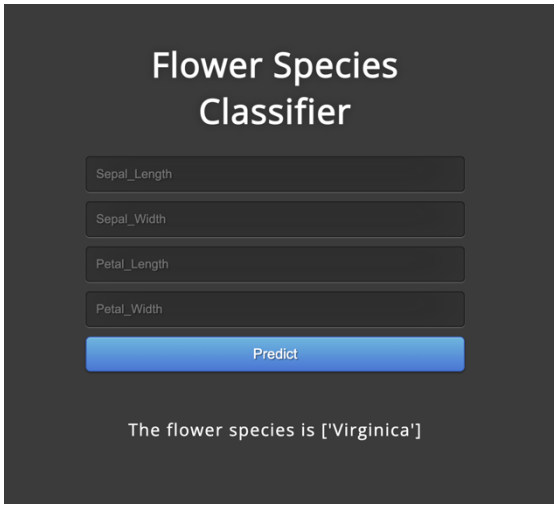
Step 6: The App in Action



The Home page



User inputs feature parameters



Flower species is displayed to the user

- The Home page is what the user sees when they first open the app
- Th user inputs their flower feature parameters in the form fields
- After the “Predict” button is clicked, the model’s prediction of the flower species is displayed to the user