

Model deployment using Flask and Heroku

Christos Christoforou

Data Glacier Internship: Cohort Code LISUM15

December 1, 2022

Step 1: Create a model using a toy dataset. I chose the famous iris dataset and a logistic classifier to predict the species of the flower.

Step 1: Create a model using a toy dataset. I chose the famous iris dataset and a logistic classifier to predict the species of the flower.

```
## import libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
import pickle

## iris dataset
iris = datasets.load_iris()

## convert to pandas dataframe
data = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                    columns= iris['feature_names'] + ['target'])

## take a look at the first rows
print(data.head())

## replace the names of the target variable
data['target'].replace({0:'Setosa',1:'Versicolor',2:'Virginica'},inplace=True)

## X : Data matrix , y : target variable
X = data.drop('target',axis=1)
y = data['target'].copy()

## logistic regressor
clf = LogisticRegression(random_state = 2, solver='lbfgs', multi_class='auto')
clf.fit(X, y)

# Saving model to disk
pickle.dump(clf, open('model.pkl','wb'))

# Loading model to compare the results
model = pickle.load(open('model.pkl','rb'))
print(model.predict(np.array([[5.1,3.5,1.4,.2]])).item())
```

Step 2: Open a text editor of your choice to create the flask app to run our model on, and the HTML code to structure a web page to display the app.

Step 2: Open a text editor of your choice to create the flask app to run our model on, and the HTML code to structure a web page to display the app.

```
app.py
import pickle
from flask import Flask, request, render_template
import numpy as np

app = Flask(__name__)
model = pickle.load(open('model.pkl', 'rb'))

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

@app.route('/predict', methods=['POST'])
def predict():
    input = [float(x) for x in request.form.values()]
    input = np.array(input)

    prediction = model.predict(input).item()

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

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

index.html

```
<!DOCTYPE html>
<html >
<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>Predict Iris Species</h1>

  <!-- Main Input For Receiving Query to our ML -->
  <form action="{{ url_for('predict')}}" method="post">
    <input type="text" name="sepal length (cm)" placeholder="Sepal Length (in cm)" required="required" />
    <input type="text" name="sepal width (cm)" placeholder="Sepal Width (in cm)" required="required" />
    <input type="text" name="petal length (cm)" placeholder="Petal Length (in cm)" required="required" />
    <input type="text" name="petal width (cm)" placeholder="Petal Width (in cm)" required="required" />

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

  <br>
  <br>
  {{ prediction_text }}

</div>

</body>
</html>
```

Step 3:

- Create a requirements text file.

Step 3:

- Create a requirements text file.
- Create a virtual environment that supports the requirements you put in the text file.

Step 3:

- Create a requirements text file.
- Create a virtual environment that supports the requirements you put in the text file.
- Start your command-line interpreter (cmd) and activate the virtual environment you created.

Step 3:

- Create a requirements text file.
- Create a virtual environment that supports the requirements you put in the text file.
- Start your command-line interpreter (cmd) and activate the virtual environment you created.

C:\Windows\System32\cmd.exe

```
C:\Users\chris\Desktop\Data Glacier Internship\Week 4>flaskenv\Scripts\activate
```

```
(flaskenv) C:\Users\chris\Desktop\Data Glacier Internship\Week 4>
```

Step 4:

- On the command-line type the name of your flask app's file (app.py in this case) and hit enter.

Step 4:

- On the command-line type the name of your flask app's file (app.py in this case) and hit enter.
- Copy the URL (<http://127.0.0.1:5000>)

Step 4:

- On the command-line type the name of your flask app's file (app.py in this case) and hit enter.
- Copy the URL (<http://127.0.0.1:5000>)

Select C:\Windows\System32\cmd.exe - app.py

```
C:\Users\chris\Desktop\Data Glacier Internship\Week 4>flaskenv\Scripts\activate
```

```
(flaskenv) C:\Users\chris\Desktop\Data Glacier Internship\Week 4>app.py
```

```
C:\Users\chris\Desktop\Data Glacier Internship\Week 4\flaskenv\Lib\site-packages\sklearn\base.py:329: UserWarning: Trying to unpickle estimator LogisticRegression 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\chris\Desktop\Data Glacier Internship\Week 4\flaskenv\Lib\site-packages\sklearn\base.py:329: UserWarning: Trying to unpickle estimator LogisticRegression 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
```

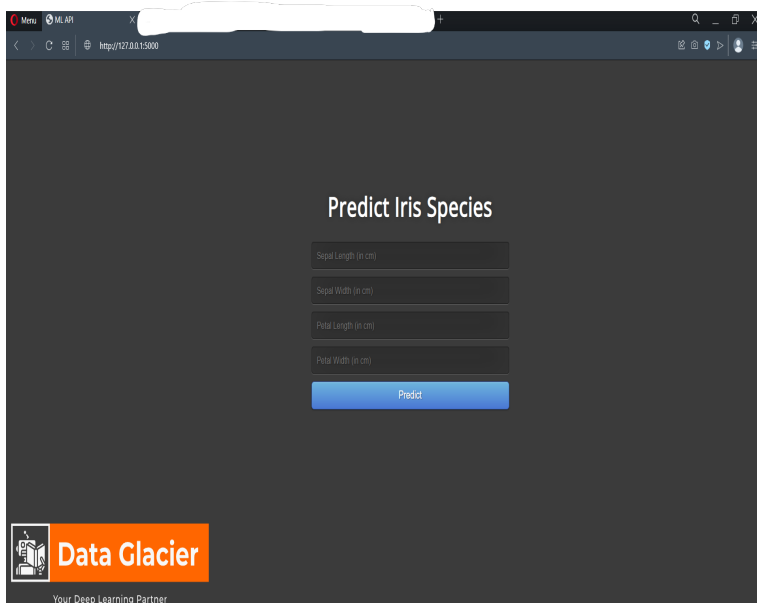
```
warnings.warn(
```

```
* Debugger is active!
```

```
* Debugger PIN: 427-089-755
```

Step 5: Paste the URL on your browser to get to the web page you created.

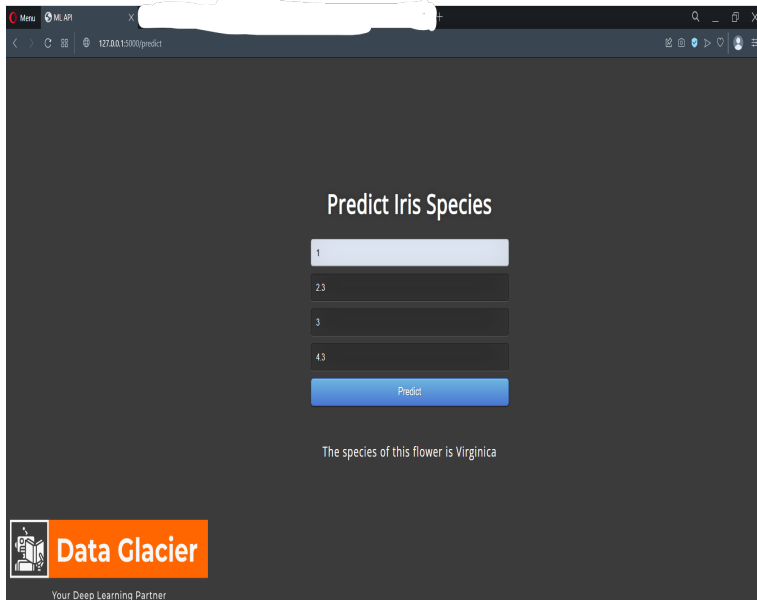
Step 5: Paste the URL on your browser to get to the web page you created.



The screenshot shows a web browser window with a single tab titled 'ML API'. The address bar displays the URL 'http://127.0.0.1:5000'. The webpage has a dark gray background and features the title 'Predict Iris Species' in white text. Below the title, there are four input fields for user data: 'Sepal Length (in cm)', 'Sepal Width (in cm)', 'Petal Length (in cm)', and 'Petal Width (in cm)'. Each field is a dark gray rectangle with light gray placeholder text. Below these fields is a prominent blue button labeled 'Predict' in white text. In the bottom-left corner of the browser window, there is a logo for 'Data Glacier' consisting of a small icon and the text 'Data Glacier' in white on an orange background. Below the logo, the text 'Your Deep Learning Partner' is visible in a smaller font. The browser's top bar includes a 'Menu' button, search, and navigation icons.

Step 6: Enter the Sepal and Petal lengths and widths and hit the predict button to predict the species of the flower.

Step 6: Enter the Sepal and Petal lengths and widths and hit the predict button to predict the species of the flower.



The screenshot shows a web browser window with a single tab titled 'ML API'. The address bar shows the URL '127.0.0.1:5000/predict'. The page has a dark grey background and is titled 'Predict Iris Species' in white text. Below the title, there are four input fields for numerical values: the first field contains '1', the second '23', the third '3', and the fourth '43'. Below these fields is a blue button labeled 'Predict'. Underneath the button, the text 'The species of this flower is Virginica' is displayed. In the bottom left corner, there is a logo for 'Data Glacier' with the tagline 'Your Deep Learning Partner'. The bottom right corner of the browser window shows navigation icons and the page number '8 / 14'.

Predict Iris Species

1


23

3

43

Predict

The species of this flower is Virginica

 **Data Glacier**
Your Deep Learning Partner

8 / 14

- So far we created and trained a model to predict the species of an iris flower, developed a web application that consists of a single web page where we can run our model and test it locally.

- So far we created and trained a model to predict the species of an iris flower, developed a web application that consists of a single web page where we can run our model and test it locally.
- Next, we will deploy our model on Heroku. Heroku is a platform as a service (PaaS) that enables developers to build, run, and operate applications entirely in the cloud. Thus, other users will be able to use our model.

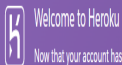
- So far we created and trained a model to predict the species of an iris flower, developed a web application that consists of a single web page where we can run our model and test it locally.
- Next, we will deploy our model on Heroku. Heroku is a platform as a service (PaaS) that enables developers to build, run, and operate applications entirely in the cloud. Thus, other users will be able to use our model.
- The first thing we need to do is to create an account on Heroku. Go to www.heroku.com and press the 'Sign up' button. Fill in the sign up form, go to your email and create a password. Your Heroku account is now good to go.

- So far we created and trained a model to predict the species of an iris flower, developed a web application that consists of a single web page where we can run our model and test it locally.
- Next, we will deploy our model on Heroku. Heroku is a platform as a service (PaaS) that enables developers to build, run, and operate applications entirely in the cloud. Thus, other users will be able to use our model.
- The first thing we need to do is to create an account on Heroku. Go to www.heroku.com and press the 'Sign up' button. Fill in the sign up form, go to your email and create a password. Your Heroku account is now good to go.
- After signing in click the 'create new app' button to create a new app.

- So far we created and trained a model to predict the species of an iris flower, developed a web application that consists of a single web page where we can run our model and test it locally.
- Next, we will deploy our model on Heroku. Heroku is a platform as a service (PaaS) that enables developers to build, run, and operate applications entirely in the cloud. Thus, other users will be able to use our model.
- The first thing we need to do is to create an account on Heroku. Go to www.heroku.com and press the 'Sign up' button. Fill in the sign up form, go to you email and create a password. Your Heroku account is now good to go.
- After signing in click the 'create new app' button to create a new app.

 Salesforce Platform

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

 Personal

Welcome to Heroku

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

[Show next steps](#)

New

 Create new app Create new pipeline

Give a name to your new app and choose a region.

Create New App

App name

app-name

Choose a region



United States



Add to pipeline...

Create app

Connect Heroku to your GitHub repository that contains the relevant code.

Personal > my-heroku-app4

Overview Resources **Deploy** Metrics Activity Access Settings

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
Connect to GitHub

Container Registry
Use Heroku CLI

Deploy using Heroku Git

Use git in the command line or a GUI tool to deploy this app.

Install the Heroku CLI

Download and install the [Heroku CLI](#).

If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.

Search for a repository to connect to

 cchristoforou97

Model-Deployment-using-Heroku

Search

Missing a GitHub organization? [Ensure Heroku Dashboard has team access.](#)

 cchristoforou97/Model-Deployment-using-Heroku

Connect

Next, press the 'deploy branch' button and wait a few minutes to deploy the model to Heroku.

Enable automatic deploys from GitHub

Every push to the branch you specify here will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch is always in a deployable state and any tests have passed before you push. [Learn more.](#)

Choose a branch to deploy

☐ Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

Enable Automatic Deploys

Deploy a GitHub branch

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

Choose a branch to deploy

Deploy Branch

Finally, press the 'view' button to view the 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

Receive code from GitHub



Build **main** 3371efbb



Release phase



Deploy to Heroku



Your app was successfully deployed.

 View