

Flask deployment report (Week 4)

Diego Martínez Echevarría

January 14, 2022

Abstract

For Week 4's assignment, I have deployed a toy model with Flask, as a web app. This document describes the process step by step with a technical view.

1 Environment setup

To facilitate replicability, the first thing to do is to setup an environment, so that others can also run the app in the intended conditions. To do so, we run the following shell commands:

```
$ python -m venv environment
$ source environment/bin/activate
$ pip install flask
$ pip freeze > requirements.txt
```

Command by command, this has the following effects:

1. A virtual environment of the name “environment” is created, with a the corresponding “environment” folder.
2. The bash script “activate” is executed. This script sets the newly created environment as the current one. Since it was just created, this environment is empty and hasn't got any modules installed.
3. We install Flask with pip.
4. We save the status of the environment (right now Flask and its dependencies) into “requirements.txt”. This file can then be used by other user with `pip install requirements.txt` to replicate our setup.

Once we've executed those commands, the requirement file looks something like this:

```
click==8.0.3
Flask==2.0.2
itsdangerous==2.0.1
Jinja2==3.0.3
MarkupSafe==2.0.1
Werkzeug==2.0.2
```

While coding this assignment other python modules have been used, so in the end this was how our requirement list looked:

```
click==8.0.3
Flask==2.0.2
itsdangerous==2.0.1
Jinja2==3.0.3
joblib==1.1.0
MarkupSafe==2.0.1
numpy==1.22.0
pandas==1.3.5
python-dateutil==2.8.2
pytz==2021.3
scikit-learn==1.0.2
scipy==1.7.3
six==1.16.0
sklearn==0.0
threadpoolctl==3.0.0
Werkzeug==2.0.2
```

2 The Dataset

We have selected a relatively simple dataset from Kaggle about crabs. It contains fields describing the physiology of a crab, using fields such as Sex, Length, Diameter, etc.

The dataset has already been curated, it's really simple and small. The only "modification" that we have made is to select a few simple fields. In the end these fields will be the ones that the user will introduce in the webpage.

The following is the code used to "process" the dataset. Of course, if the quality of the model was important, further EDA and proper analysis should be made, but for the current situation they're not necessary: we just need a toy model.

```
import pandas as pd

df = pd.read_csv('crab_age.csv').dropna()

# Process csv
fields = ["Length", "Diameter", "Height", "Weight"]
X = pd.get_dummies(data=df[fields])
y = df["Age"]
```

To fit the model, the whole dataset has been divided in predictors (X) and predictions (y), since for this model we are not considering training and testing batches.

3 The Model

The “Model” is a Linear Regression, simple as it is. This is the code used to define and train it (X and y are defined in the previous section):

```
import pickle
from sklearn.linear_model import LinearRegression

MODEL_PATH = "model.pkl"

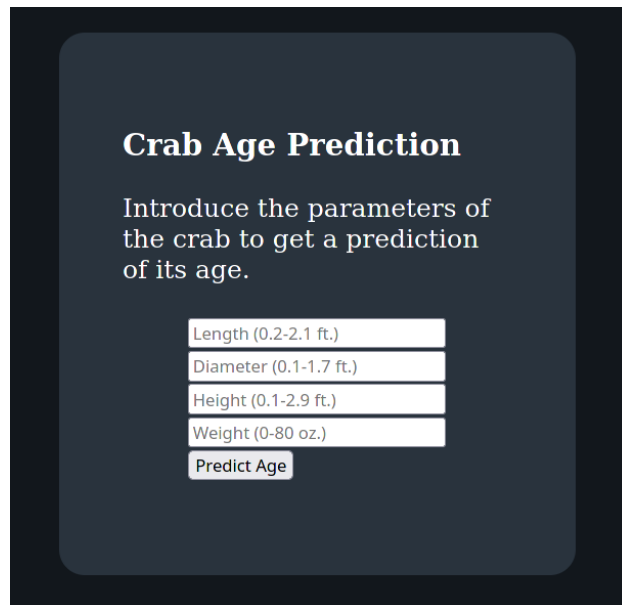
# Build model, fit model
model = LinearRegression()
model.fit(X, y)

# Pickle the object
with open(MODEL_PATH, 'wb') as savefile:
    pickle.dump(model, savefile)
```

As it is stated, we import the model, fit it with the dataset (see previous section) and we save it in a file.

4 The Form

The form and its respective html are very simple, as this is not the focus of the assignment. The following is an image of the form, interpreted by Firefox:



Besides all of the html boilerplate, we also need to connect the form with the application. We do this in two ways:

- **Connecting the form to the function that processes it:** Even if we have a form, button included, we still need to indicate where the information that a user may introduce needs to be sent. To do so, in the line where the form is defined, we write:

```
<form action="{{ url_for('render_prediction')}}" method="post">
```

This looks for the `render_prediction` function (defined in the following section) and gets its assigned url. When the user fills the form, then a request is made, using the verb `post` and the obtained url.

- **Setting a placeholder for the prediction:** The following text is placed after the form:

```
{{ prediction_text }}
```

When the app renders this html, it can substitute this tag with the actual prediction text.

5 The Web App

This is the part where Flask comes into play. Here we define the functions that get called when the user access certain sections of the webpage, or they input some data for a prediction.

Besides that, other necessary tasks are taken care of: loading the necessary modules, creating (and running) the actual app, and loading the model.

```
import pickle
from flask import Flask, request, render_template

app = Flask(__name__)

with open('model.pkl', 'rb') as model_file:
    model = pickle.load(model_file)

@app.route('/')
def render_form():
    """Renders the html containing the webapp form"""
    return render_template('simple_form.html')

@app.route('/predict', methods=['POST'])
def render_prediction():
    """Uses the user input to predict and render the result"""
    X = [float(x) for x in request.form.values()]
    prediction = round(model.predict([X])[0])
    output_text = f"The predicted age of the crab is {prediction} years."
    return render_template('simple_form.html', prediction_text=output_text)

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

6 The End Product

First we run the app (python app.py), after we can open the webpage in our browser. The form looks just like the one showed in section 4.

We are able to introduce some input (image on the left) and then the app returns some predictions using the model (image on the right).

Crab Age Prediction

Introduce the parameters of the crab to get a prediction of its age.

1.0

0.3

2.0

20

Predict Age

Crab Age Prediction

Introduce the parameters of the crab to get a prediction of its age.

Length (0.2-2.1 ft.)

Diameter (0.1-1.7 ft.)

Height (0.1-2.9 ft.)

Weight (0-80 oz.)

Predict Age

The predicted age of the crab is 24 years.

The app works as intended.