

Week 5: Cloud and API deployment

Report date: 05th June 2024

Internship Batch: LISUM33

Data intake by: Gunjan Varyani

Data intake reviewer : Data Glacier

1.Introduction:

In this project, we are going to deploy machine learning model using the Flask Framework. As a demonstration, our model helps to predict the salary based on the years of experience.

We will focus on both: building a machine learning model for Predicting the Salary expectations, then creating an API for the model, using Flask, the Python micro-framework for building web applications. This API allows us to utilize predictive capabilities through HTTP requests.

The table below lists the datasets, number of years of experience and based on the salary expectation is provided.

Years of Experience	Salary
1.1	39343
1.3	46205
1.5	37731
2	43525
2.2	39891
2.9	56642
3	60150
3.2	54445
3.2	64445
3.7	57189
3.9	63218
4	55794
4	56957
4.1	57081
4.5	61111
4.9	67938
5.1	66029
5.3	83088
5.9	81363
6	93940
6.8	91738
7.1	98273
7.9	101302
8.2	113812
8.7	109431
9	105582
9.5	116969
9.6	112635
10.3	122391
10.5	121872

2. Building a model :

Importing required libraries and dataset:

In this part, we import libraires and datasets which contain the information of the dataset.

```
#In this file, we will use the flask web framework to handle the POST requests that we
will get from the request.py and from HTML file

# Simple Linear Regression

This model predicts the salary of the employ based on experience using a simple linear
regression model.

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pickle

# Importing the dataset
dataset = pd.read_csv('Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
```

2.1 Next we will Build a model:

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3,
random_state = 0)

# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
# Predicting the Test set results
y_pred = regressor.predict(X_test)
```

2.2 After that we will save the model and run the model

```
# Saving model using pickle
pickle.dump(regressor, open('model.pkl','wb'))

# Loading model to compare the results
model = pickle.load( open('model.pkl','rb'))
print(model.predict([[1.8]]))
```

3. Turning model into web application:

We developed a web application that consists of a simple web page with a form field that lets us enter a message. After submitting the message to the web application, it will render it on a new page which gives us a result of Salary predictions.

3.1 App.py

The *app.py* file contains the main code that will be executed by the Python interpreter to run the Flask web application, it included the ML code for classifying SD.

```
#In this file, we will use the flask web framework to handle the POST requests that we
will get from the request.py and from HTML file

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

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():
    '''
    For rendering results on HTML GUI
    '''
    int_features = [int(x) for x in request.form.values()]
    final_features = [np.array(int_features)]
    prediction = model.predict(final_features)

    output = round(prediction[0], 2)

    return render_template('index.html', prediction_text='Salary is
{}'.format(output))

@app.route('/predict_api',methods=['POST'])
def predict_api():
    '''
    For direct API calls through request
    '''
```

```

data = request.get_json(force=True)
prediction = model.predict([np.array(list(data.values()))])

output = prediction[0]
return jsonify(output)

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

```

- 1) First, we ran our application as a single module; thus, we initialized a new Flask instance with the argument `__name__` to let Flask know that it can find the HTML template folder (*templates*) in the same directory where it is located.
- 2) Next, we used the route decorator (`@app.route('/')`) to specify the URL that should trigger the execution of the home function.
- 3) Our `render_template` function simply rendered the *index.html* HTML file, which is in the *templates* folder.
- 4) Inside the `predict` function, we access the salary data set, pre-process the text, and make predictions, then store the model. We access the new message entered by the user and use our model to make a prediction for its label.
- 5) We used the *POST* method to transport the form data to the server in the message body. Finally, by setting the `debug=True` argument inside the `app.run` method, we further activated Flask's debugger.
- 6) Lastly, we used the `run` function to only run the application on the server when this script is directly executed by the Python interpreter, which we ensured using the *if* statement with `__name__ == '__main__'`.

4. Index.html

The following are the contents of the *index.html* file that will render a text form where a user can enter a message.

we create a *index.html* file that will be rendered via the `render_template` line return inside the `predict` function, which we defined in the *app.py* script to display the text that a user-submitted via the text field.

```

<!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">
<div>

</div>

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

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

    <br>
    <br>
    {{ prediction_text }}

</div>

</body>
</html>

```

5. Running Procedure

Once we have done all the above steps, we can start running the API by either double click *app.py*, or executing the command from the Terminal:

```
(.venv) D:\Data Glacier Intern\week 4>python app.py
* 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
* Debugger is active!
* Debugger PIN: 685-308-960
```

Now we could open a web browser and navigate to <http://127.0.0.1:5000/>, we should see a simple website with the content like so

Salary is 45508.08

Salary is 120275.62

Salary is 167005.33

6. Model deployment using GCP

We are ready to start our GCP deployment now that our model has been trained, the machine learning pipeline has been set up, and the application has been tested locally.

Requirement.txt

It is a text file containing the python packages required to execute the application.

6.1 Steps for Model Deployment Using GCP

We are now ready to start deploying our model in GCP, below are the following steps:

1. Login to our GCP account and click on create a project and give the name of the project.

Google Cloud

Search (/) for resources, docs, products, and more

New Project

You have 19 projects remaining in your quota. Request an increase or delete projects. [Learn more](#)

[MANAGE QUOTAS](#)

Project name *
Salary Prediction

Project ID: salary-prediction-425420. It cannot be changed later. [EDIT](#)

Location *
No organization [BROWSE](#)

Parent organization or folder

[CREATE](#) [CANCEL](#)

Once we click on create project and give the name of the project, our project has been created successfully in GCP.

Manage resources

+

CREATE PROJECT

+

CREATE FOLDER

▼

MOVE

DELETE

▶

TAGS

Resources

≡

Filter

Filter

?

⋮

<input type="checkbox"/>	Name	ID	Last accessed	<div>≡</div> <div>↓</div>	Status	Charges	Carbon emissions	Unattended projects	
<input type="checkbox"/>	<div>▼</div> <div> <div>🏠</div> <div>No organization</div> </div>		June 4, 2024					—	⋮
<input type="checkbox"/>	<div>••</div> <div>Salary Prediction</div>		—				—	—	⋮
<input type="checkbox"/>	<div>••</div> <div>My First Project</div>	calm-t...	June 4, 2024			\$0.00	—	—	⋮

RESOURCES PENDING DELETION

2. Once our project has been created in GCP we will open the command prompt in the same folder for our local file saved, we will run the command `gcloud init`

```
Microsoft Windows [Version 10.0.22631.3672]
(c) Microsoft Corporation. All rights reserved.

D:\Data Glacier Intern\week 5>gcloud init
Welcome! This command will take you through the configuration of gcloud.

Settings from your current configuration [default] are:
accessibility:
  screen_reader: 'False'
core:
  account: gunjanvaryani916@gmail.com
  disable_usage_reporting: 'True'
  project: salary-prediction-425420

Pick configuration to use:
[1] Re-initialize this configuration [default] with new settings
[2] Create a new configuration
Please enter your numeric choice: |
```

After running the `gcloud init` we will choose configuration as option 1 reinitialize this configuration, Once we will choose the configuration, we can choose our email account to perform the operations for this configuration; after choosing email id we need to pick our cloud project, I have chosen here salary prediction as mentioned above

```
Pick configuration to use:
[1] Re-initialize this configuration [default] with new settings
[2] Create a new configuration
Please enter your numeric choice: 1

Your current configuration has been set to: [default]

You can skip diagnostics next time by using the following flag:
gcloud init --skip-diagnostics

Network diagnostic detects and fixes local network connection issues.
Checking network connection...done.
Reachability Check passed.
Network diagnostic passed (1/1 checks passed).

Choose the account you would like to use to perform operations for this configuration:
[1] gunjanvaryani916@gmail.com
[2] gvary248@gmail.com
[3] Log in with a new account
Please enter your numeric choice: 2

You are logged in as: [gvary248@gmail.com].

Pick cloud project to use:
[1] salary-prediction-425516
[2] Enter a project ID
[3] Create a new project
Please enter numeric choice or text value (must exactly match list item): 1|
```

Once we select our project name, it will appear as current project has been set to salary prediction project name, as below:

```
Choose the account you would like to use to perform operations for this configuration:
[1] gunjanvaryani916@gmail.com
[2] gvary248@gmail.com
[3] Log in with a new account
Please enter your numeric choice: 2

You are logged in as: [gvary248@gmail.com].

Pick cloud project to use:
[1] salary-prediction-425516
[2] Enter a project ID
[3] Create a new project
Please enter numeric choice or text value (must exactly match list item): 1

Your current project has been set to: [salary-prediction-425516].

Not setting default zone/region (this feature makes it easier to use
[gcloud compute] by setting an appropriate default value for the
--zone and --region flag).
See https://cloud.google.com/compute/docs/gcloud-compute section on how to set
default compute region and zone manually. If you would like [gcloud init] to be
able to do this for you the next time you run it, make sure the
Compute Engine API is enabled for your project on the
https://console.developers.google.com/apis page.

Your Google Cloud SDK is configured and ready to use!

* Commands that require authentication will use gvary248@gmail.com by default
* Commands will reference project 'salary-prediction-425516' by default
Run 'gcloud help config' to learn how to change individual settings

This gcloud configuration is called [default]. You can create additional configurations if you work with multiple accounts and/or projects.
Run 'gcloud topic configurations' to learn more.

Some things to try next:

* Run 'gcloud --help' to see the Cloud Platform services you can interact with. And run 'gcloud help COMMAND' to get help on any gcloud command.
* Run 'gcloud topic --help' to learn about advanced features of the SDK like arg files and output formatting
* Run 'gcloud cheat-sheet' to see a roster of go-to 'gcloud' commands.
```

Once our project has been set, we will run the command as `gcloud app deploy app.yaml --project salary-prediction-425516` for creating an APP engine application in project, for the same we need to choose the App engine location as well.

```
D:\Data Glacier Intern\week 5>gcloud app deploy app.yaml --project salary-prediction-425516
You are creating an app for project [salary-prediction-425516].
WARNING: Creating an App Engine application for a project is irreversible and the region
cannot be changed. More information about regions is at
<https://cloud.google.com/appengine/docs/locations>.

Please choose the region where you want your App Engine application located:

[1] asia-east1 (supports standard and flexible)
[2] asia-east2 (supports standard and flexible and search_api)
[3] asia-northeast1 (supports standard and flexible and search_api)
[4] asia-northeast2 (supports standard and flexible and search_api)
[5] asia-northeast3 (supports standard and flexible and search_api)
[6] asia-south1 (supports standard and flexible and search_api)
[7] asia-southeast1 (supports standard and flexible)
[8] asia-southeast2 (supports standard and flexible and search_api)
[9] australia-southeast1 (supports standard and flexible and search_api)
[10] europe-central2 (supports standard and flexible)
[11] europe-west (supports standard and flexible and search_api)
[12] europe-west2 (supports standard and flexible and search_api)
[13] europe-west3 (supports standard and flexible and search_api)
[14] europe-west6 (supports standard and flexible and search_api)
[15] northamerica-northeast1 (supports standard and flexible and search_api)
[16] southamerica-east1 (supports standard and flexible and search_api)
[17] us-central (supports standard and flexible and search_api)
[18] us-east1 (supports standard and flexible and search_api)
[19] us-east4 (supports standard and flexible and search_api)
[20] us-west1 (supports standard and flexible)
[21] us-west2 (supports standard and flexible and search_api)
[22] us-west3 (supports standard and flexible and search_api)
[23] us-west4 (supports standard and flexible and search_api)
[24] cancel
Please enter your numeric choice: 17

Creating App Engine application in project [salary-prediction-425516] and region [us-central]....done.
Services to deploy:
```

Once we will run the command we will select yes to continue as below, after selecting yes, Our deployment will begin and the files will be uploaded successfully in GCP as we can see below:

```
Run 'gcloud topic configurations' to learn more.

Some things to try next:

* Run 'gcloud --help' to see the Cloud Platform services you can interact with. And run 'gcloud help COMMAND' to get help on any gcloud command.
* Run 'gcloud topic --help' to learn about advanced features of the SDK like arg files and output formatting
* Run 'gcloud cheat-sheet' to see a roster of go-to 'gcloud' commands.

D:\Data Glacier Intern\week 5>gcloud app deploy app.yaml --project salary-prediction-425516
Services to deploy:

descriptor:      [D:\Data Glacier Intern\week 5\app.yaml]
source:          [D:\Data Glacier Intern\week 5]
target project:  [salary-prediction-425516]
target service:  [default]
target version:  [20240605t130249]
target url:      [https://salary-prediction-425516.uc.r.appspot.com]
target service account: [salary-prediction-425516@appspot.gserviceaccount.com]

Do you want to continue (Y/n)? y

Beginning deployment of service [default]...
#=====#
#- Uploading 1 file to Google Cloud Storage      =#
#=====#
File upload done.
Updating service [default]...done.
Setting traffic split for service [default]...done.
Deployed service [default] to [https://salary-prediction-425516.uc.r.appspot.com]

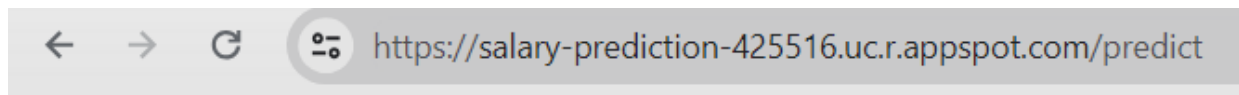
You can stream logs from the command line by running:
$ gcloud app logs tail -s default

To view your application in the web browser run:
$ gcloud app browse


D:\Data Glacier Intern\week 5>gcloud app browse
Opening [https://salary-prediction-425516.uc.r.appspot.com] in a new tab in your default browser.

D:\Data Glacier Intern\week 5>
```


Once our deployment has been completed successfully to GCP we will view our API application by running the command as gcloud app browse




Salary is 73545.9

← → ↻  <https://salary-prediction-425516.uc.r.appspot.com/predict>

Salary is 110929.67

← → ↻  <https://salary-prediction-425516.uc.r.appspot.com/predict>

Salary is 167005.33

← → ↻  <https://salary-prediction-425516.uc.r.appspot.com/predict>

Salary is 213735.04

So from the above output we can see our API is running successfully through Google cloud platform.