Final Project: AI-Based Web Application Development and Deployment

Estimated Time: 1 hour 45 minutes

Scenario

You have been hired as a software engineer by an e-commerce company to create an AI-based web app that performs analytics on customer feedback for their signature products. To accomplish this requirement, you will create an Emotion Detection system that processes feedback provided by the customer in text format and deciphers the associated emotion expressed.

Introduction

In this final project, you will be assessed on the knowledge gained on all aspects of app creation and its web deployment throughout this course. You will be required to save screenshots of your results from time to time, with specific nomenclature. These screenshots will have to be uploaded in the peer graded assignment that follows.

In this project, we use the embeddable Watson AI libraries to create an emotion detection application.

Emotion detection extends the concept of sentiment analysis by extracting the finer emotions, like joy, sadness, anger, and so on, from statements rather than the simple polarity that sentiment analysis provides. This makes emotion detection a very important branch of study and businesses use such systems widely for their AI based recommendation systems, automated chat bots, and so on.

Prerequisite

Before you begin this project, make sure you have an active GitHub account. You'll be using GitHub to create and manage the repository for this lab. If you don't have an account yet, follow the steps in this lab GitHub Sign Up and Create Repo. It walks you through creating a GitHub account and setting up a repository, step by step.

Project guidelines

For the completion of this project, you'll have to complete the following 8 tasks, based on the knowledge you have gained through the course.

Note: This platform is not persistent. It is recommended that you keep a copy of your code on your local machines and save changes from time to time. In case you revisit the lab, you will need to recreate the files in this lab environment, using the saved copies from your machines.

Tasks and objectives:

- Task 1: Fork and Clone the project repository
- Task 2: Create an emotion detection application using Watson NLP library
- Task 3: Format the output of the application
- Task 4: Package the application
- Task 5: Run Unit tests on your application
- Task 6: Deploy as web application using Flask
- Task 7: Incorporate Error handling
- Task 8: Run static code analysis

Let's get started!

Task 1: Fork and Clone the project repository

Note: *Please run this lab in the Skills Network Theia Lab environment itself, not in your local IDE (like VS Code). The API used here is hosted on the Skills Network platform and is only accessible within the Theia Lab.*

The GitHub repository of the project is available on the URL mentioned below.
As a first step you need to Fork this repository, click the Fork button at the top-right corner of the repository page.

```
https://github.com/ibm-developer-skills-network/oaqjp-final-project-emb-ai.git
```

You can find instructions on how to fork the repository by visiting exercise 2 at the following link.

`Note:` Ensure your forked repository is Public.This action will create a copy of the forked repository within your GitHub account.

1. Open a new Terminal and make the directory `final_project` using mkdir command.
2. Clone this forked GitHub repository using the Cloud IDE terminal to your project to a folder named `final_project`.
   You can find instructions on how to get the clone URL of the repository by visiting exercise 3 at the following link.
3. After the cloning is complete, use the terminal to change the current directory `final_project`.
4. Take the **screenshot** of your folder structure. Save this image as `1_folder_structure.png`.

Task 2: Create an emotion detection application using the Watson NLP library

The Watson NLP libraries are embedded. Therefore, there is no need of importing them to your code. You only need to send a post request to the correct function in the library and receive the output.

1. For this project, you'll use the **Emotion Predict** function of the Watson NLP Library. For accessing this function, the URL, the headers, and the input json format is as follows.

   ```
   URL: 'https://sn-watson-emotion.labs.skills.network/v1/watson.runtime.nlp.v1/NlpService/EmotionPredict'
   Headers: {"grpc-metadata-mm-model-id": "emotion_aggregated-workflow_lang_en_stock"}
   Input json: { "raw_document": { "text": text_to_analyse } }
   ```

   Note that the `text_to_analyze` is being used as a variable that holds the actual written text that needs to be analyzed.

2. Create a file named `emotion_detection.py` in `final_project` folder.
3. In the `emotion_detection.py` file, write the function to run emotion detection using the appropriate Emotion Detection function. Name this function `emotion_detector`.

   Note: Assume that that text to be analyzed is passed to the function as an argument and is stored in the variable `text_to_analyze`. The value being returned must be the `text` attribute of the response object as received from the Emotion Detection function.

4. Take a **screenshot** of the code you write and save it as `2a_emotion_detection.png`.
5. The application should now be importable using Python shell. Open a `python3` shell.

6. Import the application.
7. After you successfully import the application, test your application with the text: **"I love this new technology."**
8. Take a **screenshot** of the terminal with all three steps included in it along with the final output. Name this file `2b_application_creation.png`.

Note: In case the python shell shows an error `ModuleNotFoundError: No module named 'requests'`, you may install the `requests` library to your IDE using the following command on the terminal.

```
python3 -m pip install requests
```

Optional:

At any point of time if you want to push your code to your forked GitHub repository, you can do so by following the instructions provided in this link.

Task 3: Format the output of the application

1. Convert the response text into a dictionary using the `json` library functions.

   Note the content formatting in this dictionary.
2. Extract the required set of emotions, including anger, disgust, fear, joy and sadness, along with their scores.
3. Write the code logic to find the dominant emotion, which is the emotion with the highest score.
4. Then, modify the `emotion_detector` function to return the following output format.

```
{
'anger': anger_score,
'disgust': disgust_score,
'fear': fear_score,
'joy': joy_score,
'sadness': sadness_score,
'dominant_emotion': '<name of the dominant emotion>'
}
```

5. Take a **screenshot** of the modified function code and save it as `3a_output_formatting.png`.
6. Test the application in the `python3` shell again, with the statement **I am so happy I am doing this**.
7. Verify that the response received has the dominant emotion is **joy**.
8. Take a **screenshot** of the output from the terminal shell and save it as `3b_formatted_output_test.png`.

Optional

At any point of time if you want to push your code to your forked GitHub repository, you can do so by following the instructions provided in this link.

Task 4: Package the application

In this task, you will package the application created in the previous steps.

1. Perform the relevant steps in creation of the package. Set the name of the package to `EmotionDetection`.
2. Once completed, take a **screenshot** of the contents of the `init` file along with the final folder structure of the final_project directory.

   Make sure to fit both of the images into a single screenshot and name it `4a_packaging.png`.
3. To make sure that the `EmotionDetection` is a valid package, you need to test the package. To do this:
    1. Run a python shell in the terminal.
    2. Import the `emotion_detector` function from the package.
    3. If the package is created accurately, you will not receive any error messages.
    4. Test run the function again with the statement **I hate working long hours**.
    5. Verify that the output displays the dominant emotion as **anger**.
4. Take a **screenshot** of this terminal output and save it as `4b_packaging_test.png`.

Optional:

At any point of time if you want to push your code to your forked GitHub repository, you can do so by following the instructions provided in this link.

Task 5: Run unit tests on your application

With a functional application being available, you now need to run unit tests to validate the application output.

1. To run unit tests, create a new file, `test_emotion_detection.py` that calls the required application function from the package and tests it for the following statements and dominant emotions.

Statement	**Dominant Emotion**
I am glad this happened	joy
I am really mad about this	anger
I feel disgusted just hearing about this	disgust
I am so sad about this	sadness
I am really afraid that this will happen	fear

2. Once complete, take a **screenshot** of the code and save it as `5a_unit_testing.png`.
3. Execute the `test_emotion_detection.py` file on terminal.
4. Check the unit test output to verify that the unit tests have passed.
5. Take a **screenshot** of the terminal output of this execution and save it as `5b_unit_testing_result.png`.

Optional:

At any point of time if you want to push your code to your forked GitHub repository, you can do so by following the instructions provided in this link.

Task 6: Web deployment of the application using Flask

After all the tests are successfully completed, you need to deploy the application on the web. This will enable the customer to access and use the application.

Note: *The `index.html` file in the `templates` folder and `mywebscript.js` file in the `static` folder have been provided as a part of the repo. These files do not need to be updated in this project.*

1. You have to create the `server.py` file from scratch. Note the following requirements of this task.

   Note: `server.py` is a part of `final_project` folder.

2. Make sure that the Flask decorator for the application calling function is `\emotionDetector`.

3. The customer has requested the output to be displayed in the format as displayed in the example below.

   Example Output
   Let's say that you want to evaluate the statement `I love my life`. The statement is processed as follows:

   ```
   {
   "anger": 0.006274985,
   "disgust": 0.0025598293,
   "fear": 0.009251528,
   "joy": 0.9680386,
   "sadness": 0.049744144,
   "dominant_emotion":"joy"
   }
   ```

   The response is displayed as:

   ```
   For the given statement, the system response is 'anger': 0.006274985,
   'disgust': 0.0025598293, 'fear': 0.009251528, 'joy': 0.9680386 and 'sadness':
   0.049744144. The dominant emotion is joy.
   ```

4. Application needs to be deployed on `localhost:5000`

For completing this task of the project, you have to first take a snapshot of the final contents of `server.py` and name it `6a_server.png`. Secondly, you have to take a snapshot of the final deployed application, testing it for the statement “I think I am having fun”. Name this image as `6b_deployment_test.png`.

Optional:

At any point of time if you want to push your code to your forked GitHub repository, you can do so by following the instructions provided in this link.

Task 7: Incorporate Error handling

Incorporate the error handling capability in your function `emotion_detector` to manage blank entries from users, i.e. running the application without any input.

1. Access the `status_code` attribute of the server response to correctly display the system response for blank entries.

   For `status_code = 400`, make the function return the same dictionary, but with values for all keys being `None`.

2. Take a **screenshot** of the modified function and name it `7a_error_handling_function.png`.

3. Modify `server.py` to incorporate error handling when the `dominant_emotion` is `None`. In this scenario, the response should display a message **Invalid text! Please try again!**.

4. Take a **screenshot** of this edited file and save it as `7b_error_handling_server.png`.

5. Deploy the application and test it for blank entries.

6. The output should display the error message, **Invalid text! Please try again!**.

7. Take a **screenshot** of the output displaying the error message and name it `7c_error_handling_interface.png`

Optional:

At any point of time if you want to push your code to your forked GitHub repository, you can do so by following the instructions provided in this link.

Task 8: Run static code analysis

Finally, its time to check code compliance.

1. Run static code analysis on the code you created. Use PyLint library on `server.py` file and try to generate a 10/10 score. You will have to modify the `server.py` file to get this score. Take a snapshot of the modified file and name it as `8a_server_modified.png`.

   Note: The difference between a 9/10 score and a 10/10 score may be use of Docstrings. Include Docstrings in all functions of your codes.

2. Take a **screenshot** of a 10/10 output on the terminal after running static code analysis. Name the screenshot as `8b_static_code_analysis.png`.

Checklist for the images

Before you submit the project for evaluation, make sure to verify that you have captured all the images as instructed during the course of the project. Here is a quick summary of the required.

Task 1: Clone the project repository

```
1_folder_structure.png
```

Task 2: Create an emotion detection application using Watson NLP library

```
2a_emotion_detection.png
2b_application_creation.png
```

Task 3: Format the output of the application

```
3a_output_formatting.png
3b_formatted_output_test.png
```

Task 4: Package the application

```
4a_packaging.png
4b_packaging_test.png
```

Task 5: Run Unit tests on your application

```
5a_unit_testing.png
5b_unit_testing_result.png
```

Task 6: Deploy as web application using Flask

```
6a_server.png
6b_deployment_test.png
```

Task 7: Incorporate Error handling

```
7a_error_handling_function.png
7b_error_handling_server.png
7c_error_handling_interface.png
```

Task 8: Run static code analysis

```
8a_server_modified.png
8b_static_code_analysis.png
```

Optional:

If you want to push your code to your forked GitHub repository, you can do so by following the instructions provided in this link. This ensures you can easily review your work whenever needed.

Conclusion

Congratulations on completing the project.

By completing this project, you have:

1. Created an Emotion Detection application using the functions from embeddable AI libraries
2. Extracted relevant information from the output received from the function
3. Tested and packaged the application created using the Emotion Detection function
4. Completed web deployment of your application using Flask
5. Incorporated error handling in your application to account for invalid input to your application
6. Written codes that are in perfect compliance with PEP8 guidelines, getting 10/10 score in static code analysis

Next Steps

Now that you have completed the project, please submit the project for peer review.
Upload all the images, which you captured when building the project, as instructed in the **Peer Graded Review Submission**.

Author(s)

Abhishek Gagneja