IBM AI Engineering Final Project Report

Emotion Detection Web Application using Watson NLP, Flask, and Python

Kumar Hemant

July 11, 2025

Contents

1 Task 1: Fork and Clone the Project Repository

1.1 Why

To begin development, you must first fork and clone the starter repository into your own GitHub account and local Cloud IDE environment.

1.2 Steps

```
# Step 1: Fork the repo on GitHub from:
https://github.com/ibm-developer-skills-network/oaqjp-final-project-
    emb-ai.git

# Step 2: In Cloud IDE terminal
mkdir final_project
cd final_project

# Step 3: Clone your forked repository
git clone https://github.com/YOUR_USERNAME/oaqjp-final-project-emb-
    ai.git .
```

1.3 Optional

Verify structure with:

```
ls -R
```

2 Task 2: Create the Emotion Detection Application

2.1 Why

To detect emotions from text using IBM Watson NLP API by sending a POST request.

2.2 Steps

1. Create emotion_detection.py in the final_project folder.
2. Write the function emotion_detector.

2.3 Code Snippet

```
import requests

def emotion_detector(text_to_analyze):
    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"
    }
    payload = {"raw_document": {"text": text_to_analyze}}

    response = requests.post(url, json=payload, headers=headers)
    return response.text
```

2.4 Optional

Install ‘requests‘:

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

3 Task 3: Format the Output of the Application

3.1 Why

The raw JSON response must be converted into a dictionary and formatted for structured output.

3.2 Steps

1. Use json.loads() to parse the response.
2. Extract keys: anger, disgust, fear, joy, sadness.
3. Compute dominant_emotion.

3.3 Formatted Function

```
import requests
import json
```

```
def emotion_detector(text_to_analyze):
    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"
    }
    payload = {"raw_document": {"text": text_to_analyze}}

    response = requests.post(url, json=payload, headers=headers)
    if response.status_code != 200:
        return {
            'anger': None, 'disgust': None, 'fear': None,
            'joy': None, 'sadness': None, 'dominant_emotion': None
        }

    emotions = json.loads(response.text)['emotionPredictions'][0]['
        emotion']
    dominant = max(emotions, key=emotions.get)
    emotions['dominant_emotion'] = dominant
    return emotions
```

4 Task 4: Package the Application

4.1 Why

To convert the module into a reusable and importable Python package.

4.2 Steps

1. Create folder: EmotionDetection
2. Move emotion_detection.py into it.
3. Add ___init___.py with the following:

```
from .emotion_detection import emotion_detector
```

4.3 Test Import

```
from EmotionDetection import emotion_detector
print(emotion_detector("I hate working long hours"))
```

5 Task 5: Run Unit Tests

5.1 Why

Unit tests verify application correctness for known test cases.

5.2 Steps

Create test_emotion_detection.py with the following:

```
import unittest
from EmotionDetection import emotion_detector

class TestEmotionDetection(unittest.TestCase):
    def test_joy(self):
        self.assertEqual(emotion_detector("I am glad this happened")
            ["dominant_emotion"], "joy")
    def test_anger(self):
        self.assertEqual(emotion_detector("I am really mad about
            this")["dominant_emotion"], "anger")
    def test_disgust(self):
        self.assertEqual(emotion_detector("I feel disgusted just
            hearing about this")["dominant_emotion"], "disgust")
    def test_sadness(self):
        self.assertEqual(emotion_detector("I am so sad about this")[
            "dominant_emotion"], "sadness")
    def test_fear(self):
        self.assertEqual(emotion_detector("I am really afraid that
            this will happen")["dominant_emotion"], "fear")

if __name__ == "__main__":
    unittest.main()
```

6 Task 6: Web Deployment using Flask 🌐

6.1 Why

To provide a user-friendly web interface to access your app.

6.2 Steps

1. Create server.py in final_project.
2. Use Flask to create a web server with a route /emotionDetector.

6.3 Minimal Flask Code

```
from flask import Flask, request, render_template
from EmotionDetection import emotion_detector

app = Flask(__name__)

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

@app.route('/emotionDetector', methods=['GET'])
def emotion_route():
    text = request.args.get("textToAnalyze")
    result = emotion_detector(text)

    if result["dominant_emotion"] is None:
        return "Invalid text! Please try again!"

    return (f"For the given statement, the system response is 'anger
        ': {result['anger']}, "
            f"'disgust': {result['disgust']}, 'fear': {result['fear
                ']}, "
            f"'joy': {result['joy']} and 'sadness': {result['sadness
                ']}. "
            f"The dominant emotion is {result['dominant_emotion']}."
                )

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

7 Task 7: Error Handling

7.1 Why

Gracefully handle blank input and unexpected API issues.

7.2 In emotion_detector.py

Return None for all emotions if response status is not 200: (Already shown in Task 3)

7.3 In server.py

Return a user-friendly message:

```
if result["dominant_emotion"] is None:
    return "Invalid text! Please try again!"
```

8 Task 8: Static Code Analysis

8.1 Why

PEP8 compliance ensures clean, readable, and professional code.

8.2 Run PyLint

```
python3 -m pip install pylint
pylint server.py
```

8.3 Fix Suggestions

- Add docstrings to all functions
- Fix spacing, naming, and unused imports

8.4 Goal

Achieve:

```
Your code has been rated at 10.00/10
```

9 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 screenshots:

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 the project guide. This ensures you can easily review your work whenever needed.