Entwicklung und Bereitstellung von KI-basierten Webanwendungen

Geschätzte Zeit: 60 Minuten

Übersicht

In diesem Projekt nutzen wir die eingebetteten Watson KI-Bibliotheken, um eine Anwendung zu erstellen, die eine Sentiment-Analyse eines bereitgestellten Textes durchführt. Anschließend stellen wir die Anwendung über das Web mit dem Flask-Framework bereit.

Projektleitlinien

Für den Abschluss dieses Projekts musst du die folgenden 8 Aufgaben basierend auf dem Wissen, das du im Kurs erworben hast, abschließen.

Aufgaben und Ziele:

- Aufgabe 1: Klone das Projekt-Repository
- Aufgabe 2: Erstelle eine Sentiment-Analyse-Anwendung mit der Watson NLP-Bibliothek
- Aufgabe 3: Formatiere die Ausgabe der Anwendung
- Aufgabe 4: Verpacke die Anwendung
- Aufgabe 5: Führe Unit-Tests für deine Anwendung durch
- Aufgabe 6: Stelle die Anwendung als Webanwendung mit Flask bereit
- Aufgabe 7: Integriere Fehlerbehandlung
- Aufgabe 8: Führe eine statische Codeanalyse durch

Lass uns anfangen!

Über einbettbare Watson AI-Bibliotheken

In diesem Projekt werden Sie einbettbare Bibliotheken verwenden, um eine KI-gestützte Python-Anwendung zu erstellen.

Einbettbare Watson AI-Bibliotheken umfassen die NLP-Bibliothek, die Text-zu-Sprache-Bibliothek und die Sprache-zu-Text-Bibliothek. Diese Bibliotheken können als Teil Ihrer Anwendung eingebettet und verteilt werden. Zu Ihrer Bequemlichkeit sind diese Bibliotheken bereits auf Skills Network Labs Cloud IDE vorinstalliert und können in diesem Projekt verwendet werden.

Die NLP-Bibliothek enthält Funktionen zur Sentimentanalyse, Emotionserkennung, Textklassifizierung, Sprachenerkennung usw. unter anderem. Die Sprache-zu-Text-Bibliothek enthält Funktionen, die den Transkriptionsdienst durchführen und geschriebenen Text aus gesprochener Audio generieren. Die Text-zu-Sprache-Bibliothek erzeugt natürlich klingende Audioausgaben aus geschriebenem Text. Alle verfügbaren Funktionen in jeder dieser Bibliotheken rufen vortrainierte KI-Modelle auf, die alle auf den Cloud IDE-Servern verfügbar sind und allen Nutzern kostenlos zur Verfügung stehen.

Diese Bibliotheken können auch über Ihre persönlichen Systeme aufgerufen werden. Die Richtlinien dafür sind auf der Watson AI-Bibliotheksseite verfügbar.

Aufgabe 1: Klone das Projekt-Repository

Das Github-Repository des Projekts ist unter der unten genannten URL verfügbar.

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

- Öffne ein neues Terminal und erstelle das Verzeichnis `practice_project` mit dem Befehl `mkdir` und wechsle mit dem Befehl `cd` in das aktuelle Verzeichnis `practice_project`.

  ```
  mkdir practice_project
  cd practice_project
  ```

- Klone dieses GitHub-Repo über das Cloud-IDE-Terminal in dein Projekt in einen Ordner namens `practice_project`.

  ▼ Klicke hier für einen Hinweis
  git clone <Paste_URL_here> folder
  ▼ Klicke hier für die Lösung

  git clone https://github.com/ibm-developer-skills-network/zzrjt-practice-project-emb-ai.git practice_project
  cd practice_project

- Stelle sicher, dass Python 3.11 und die erforderlichen Bibliotheken verfügbar sind:

```
python3.11 -V
pip3.11 show requests flask pylint
```

- Installiere fehlende Bibliotheken:

```
python3.11 -m pip install requests flask pylint
```

- Nach Abschluss sollte der Projekt-Tab die Ordnerstruktur wie im Bild gezeigt haben.

Aufgabe 2: Erstellen einer Sentiment-Analyse-Anwendung mit der Watson NLP-Bibliothek

Die Sentiment-Analyse in der NLP ist die Praxis, Computer zu nutzen, um die in einem Text ausgedrückte Stimmung oder Emotion zu erkennen. Durch NLP kategorisiert die Sentiment-Analyse Wörter als positiv, negativ oder neutral.

Die Sentiment-Analyse wird häufig auf Textdaten durchgeführt, um Unternehmen dabei zu helfen, die Stimmung zu Marken und Produkten in Kundenfeedback zu überwachen und die Bedürfnisse der Kunden zu verstehen. Sie hilft dabei, die Einstellung und Stimmung der breiten Öffentlichkeit zu erfassen, was dann wertvolle Informationen über den Kontext liefern kann.

Für die Erstellung der Sentiment-Analyse-Anwendung werden wir die Watson Embedded AI Libraries nutzen. Da die Funktionen dieser Bibliotheken bereits auf dem Cloud IDE-Server bereitgestellt sind, ist es nicht notwendig, diese Bibliotheken in unseren Code zu importieren. Stattdessen müssen wir eine POST-Anfrage an das entsprechende Modell mit dem erforderlichen Text senden, und das Modell wird die passende Antwort senden.

Ein Beispielcode für eine solche Anwendung könnte sein

```
import requests
def <function_name>(<input_args>):
    url = '<relevant_url>'
    headers = {<header_dictionary>}
    myobj = {<input_dictionary_to_the_function>}
    response = requests.post(url, json = myobj, headers=header)
    return response.text
```

Hinweis: Die Antwort der Watson NLP-Funktionen erfolgt in Form eines Objekts. Um auf die Details der Antwort zuzugreifen, können wir das Attribut `text` des Objekts verwenden, indem wir `response.text` aufrufen und die Funktion die Antwort als einfachen Text zurückgeben lassen.

Für dieses Projekt verwenden Sie die auf BERT basierende Sentiment-Analysefunktion der Watson NLP-Bibliothek. Um auf diese Funktion zuzugreifen, sind die URL, die Header und das Eingabe-JSON-Format wie folgt.

```
URL: 'https://sn-watson-sentiment-bert.labs.skills.network/v1/watson.runtime.nlp.v1/NlpService/SentimentPredict'
Headers: {"grpc-metadata-mm-model-id": "sentiment_aggregated-bert-workflow_lang_multi_stock"}
Input json: { "raw_document": { "text": text_to_analyse } }
```

Hier wird `text_to_analyze` als Variable verwendet, die den tatsächlichen geschriebenen Text enthält, der analysiert werden soll.

In dieser Aufgabe müssen Sie eine neue Datei mit dem Namen `sentiment_analysis.py` im Ordner `practice_project` erstellen. In dieser Datei schreiben Sie die Funktion zur Durchführung der Sentiment-Analyse mithilfe der Watson NLP BERT Sentiment-Analysefunktion, wie oben besprochen. Nennen wir diese Funktion

`sentiment_analyzer`. Gehen Sie davon aus, dass der zu analysierende Text als Argument an die Funktion übergeben wird und in der Variable `text_to_analyse` gespeichert ist.

▼ Klicken Sie hier für die Lösung
sentiment_analysis.py

```
import requests  # Importieren Sie die requests-Bibliothek, um HTTP-Anfragen zu bearbeiten
def sentiment_analyzer(text_to_analyse):  # Definieren Sie eine Funktion namens sentiment_analyzer, die einen String-Input (text_to_analyse) ent
    url = 'https://sn-watson-sentiment-bert.labs.skills.network/v1/watson.runtime.nlp.v1/NlpService/SentimentPredict'  # URL des Sentiment-Analy
    myobj = { "raw_document": { "text": text_to_analyse } }  # Erstellen Sie ein Dictionary mit dem zu analysierenden Text
    header = {"grpc-metadata-mm-model-id": "sentiment_aggregated-bert-workflow_lang_multi_stock"}  # Setzen Sie die erforderlichen Header für di
    response = requests.post(url, json = myobj, headers=header)  # Senden Sie eine POST-Anfrage an die API mit dem Text und den Headern
    return response.text  # Geben Sie den Antworttext von der API zurück
```

Diese Anwendung kann jetzt über die Python-Shell aufgerufen werden. Um die Anwendung zu testen, öffnen Sie eine Python-Shell mit python3.11, um die Python-Shell im aktuellen Verzeichnis, d.h. practice_project, zu öffnen. *Stellen Sie sicher, dass das aktuelle Verzeichnis `practice_project` ist.*

```
python3.11
```

Im Python-Shell importiere die Funktion `sentiment_analyzer`.

▼ Klicke hier für einen Hinweis
Syntax:

```
von dateiname import funktion_name
```

▼ Klicke hier für die Lösung

```
von sentiment_analysis import sentiment_analyzer
```

Teste deine Anwendung nach dem erfolgreichen Import mit dem Text “Ich liebe diese neue Technologie.”

```
sentiment_analyzer("I love this new technology")
```

Das erwartete Ergebnis ist wie im Bild unten gezeigt. Um die Python-Shell zu verlassen, drücken Sie `Ctrl+Z` oder geben Sie `exit()` ein.

Damit ist die Aufgabe 2 abgeschlossen. Beachten Sie, dass in der Ausgabe nur die für uns relevanten Informationen das `label` und der `score` sind. In der folgenden Aufgabe werden Sie diese Informationen aus dieser Ausgabe extrahieren.

Aufgabe 3: Formatieren Sie die Ausgabe der Anwendung

Die Ausgabe der erstellten Anwendung liegt in Form eines Wörterbuchs vor, wurde jedoch als Text formatiert. Um relevante Informationen aus dieser Ausgabe zu extrahieren, müssen wir diesen Text zuerst in ein Wörterbuch umwandeln. Da Wörterbücher das Standardformatierungssystem für JSON-Dateien sind, nutzen wir die integrierte Python-Bibliothek `json`.

Lassen Sie uns sehen, wie das funktioniert.

Zuerst importieren Sie in einer Python-Shell die json-Bibliothek.

```
import json
```

Führen Sie als Nächstes die Funktion sentiment_analyzer für den Text “I love this new technology” aus, genau wie in Aufgabe 2, und speichern Sie die Ausgabe in einer Variablen namens `response`.

```
from sentiment_analysis import sentiment_analyzer
response = sentiment_analyzer("I love this new technology")
```

Jetzt übergib die `response`-Variable als Argument an die Funktion json.loads und speichere die Ausgabe in `formatted_response`. Drucke `formatted_response` aus, um den Unterschied in der Formatierung zu sehen.

```
formatted_response = json.loads(response)
print(formatted_response)
```

Die erwartete Ausgabe der oben genannten Schritte ist im Bild unten dargestellt.

Beachten Sie, dass das Fehlen von einfachen Anführungszeichen auf beiden Seiten der Antwort darauf hinweist, dass dies kein Text mehr ist, sondern stattdessen ein Wörterbuch. Um die richtigen Informationen aus diesem Wörterbuch abzurufen, müssen wir die Schlüssel entsprechend ansprechen. Da es sich um eine verschachtelte Wörterbuchstruktur handelt, d. h. ein Wörterbuch von Wörterbüchern, müssen die folgenden Anweisungen verwendet werden, um die Ausgaben für das Label und den Score aus dieser Antwort zu erhalten.

```
label = formatted_response['documentSentiment']['label']
score = formatted_response['documentSentiment']['score']
```

Überprüfen Sie den Inhalt von `label` und `score`, um die Ausgabe zu verifizieren.

Für Aufgabe 3 integrieren Sie die oben erwähnte Technik und nehmen Sie Änderungen an der Datei `sentiment_analysis.py` vor. Die erwartete Ausgabe beim Aufruf der Funktion `sentiment_analyzer` sollte nun ein Wörterbuch mit 2 Schlüsseln, label und score, sein, wobei jeder den entsprechenden Wert enthält, der aus der Antwort der Watson NLP-Funktion extrahiert wurde. Überprüfen Sie Ihre Änderungen, indem Sie die modifizierte Funktion in einer Python-Shell testen.

▼ Hier klicken für die Lösung

```
import requests
import json
def sentiment_analyzer(text_to_analyse):
    # URL des Sentiment-Analyse-Dienstes
    url = 'https://sn-watson-sentiment-bert.labs.skills.network/v1/watson.runtime.nlp.v1/NlpService/SentimentPredict'
    # Konstruktion der Anfrage-Payload im erwarteten Format
    myobj = { "raw_document": { "text": text_to_analyse } }
    # Benutzerdefinierter Header, der die Modell-ID für den Sentiment-Analyse-Dienst angibt
    header = {"grpc-metadata-mm-model-id": "sentiment_aggregated-bert-workflow_lang_multi_stock"}
    # Senden einer POST-Anfrage an die Sentiment-Analyse-API
    response = requests.post(url, json=myobj, headers=header)
    # Parsen der JSON-Antwort von der API
    formatted_response = json.loads(response.text)
    # Extrahieren des Sentiment-Labels und des Scores aus der Antwort
    label = formatted_response['documentSentiment']['label']
```

```
    score = formatted_response['documentSentiment']['score']
    # Rückgabe eines Wörterbuchs mit den Ergebnissen der Sentiment-Analyse
    return {'label': label, 'score': score}
```

Am Ende sollte die erwartete Ausgabe der Funktion im Bild unten gezeigt werden.

Um die Python-Shell zu verlassen, geben Sie `exit()` ein oder drücken Sie `Ctrl+Z`.

Aufgabe 4: Verpacken der Anwendung

In dieser Aufgabe musst du die finale Anwendung, die du in den Aufgaben 2 und 3 erstellt hast, verpacken.

Lass uns den Namen des Pakets als `SentimentAnalysis` beibehalten. Die Schritte, die beim Verpacken erforderlich sind, sind:

1. Erstelle einen Ordner im Arbeitsverzeichnis mit dem Namen des Pakets.

▼ Hier klicken für einen Hinweis
mkdir <package_name>
▼ Hier klicken für die Lösung

```
mkdir SentimentAnalysis
```

2. Verschiebe den Anwendungscode (d.h. das Modul) in den Paketordner.

▼ Hier klicken für einen Hinweis
Du kannst einen Terminalbefehl oder die Cloud-IDE-Konsole verwenden, um die Datei `sentiment_analysis.py` in den Ordner `SentimentAnalysis` zu verschieben.
▼ Hier klicken für die Lösung

```
mv ./sentiment_analysis.py ./SentimentAnalysis
```

3. Erstelle die neue Datei als **init**.py-Datei im Paketordner, um das Modul zu referenzieren.

▼ Hier klicken für einen Hinweis
Importiere das Modul/die Funktion aus dem aktuellen Ordner in die Init-Datei.
▼ Hier klicken für die Lösung
Füge diese folgende Zeile zu __init__.py ein:

```
from . import sentiment_analysis
```

Die endgültige Ordnerstruktur sollte wie im Bild unten aussehen.

`SentimentAnalysis` ist jetzt ein gültiges Paket und kann in jede Datei dieses Projekts importiert werden.

Um dies zu testen, starte eine Python-Shell im Terminal und versuche, die Funktion `sentiment_analyzer` aus dem Paket zu importieren.

▼ Hier klicken für den Hinweis
Die Syntax für diesen Import ist

```
from package_name.module_name import function_name
```

▼ Hier klicken für die Lösung

```
from SentimentAnalysis.sentiment_analysis import sentiment_analyzer
```

Wenn nach der Importanweisung keine Fehlermeldung angezeigt wird, bedeutet dies, dass das Paket jetzt einsatzbereit ist. Teste die Funktion, indem du die folgende Anweisung in der Shell ausführst.

```
sentiment_analyzer("This is fun.")
```

Die erhaltene Ausgabe würde wie unten gezeigt aussehen.

Um die Python-Shell zu verlassen, geben Sie `exit()` ein oder drücken Sie `Ctrl+Z`.

Aufgabe 5: Führen Sie Unit-Tests für Ihre Anwendung aus

Da wir nun eine funktionale Anwendung haben, ist es erforderlich, dass wir Unit-Tests für einige Testfälle durchführen, um die Gültigkeit ihrer Ausgaben zu überprüfen.

Um Unit-Tests auszuführen, müssen wir eine neue Datei erstellen, die die erforderliche Anwendungsfunktion aus dem Paket aufruft und diese für ein bekanntes Text- und Ausgabepaar testet.

Dazu führen Sie die folgenden Schritte aus.

1. Erstellen Sie eine neue Datei im Ordner `practice_project`, die `test_sentiment_analysis.py` heißt.

2. Importieren Sie in dieser Datei die Funktion `sentiment_analyzer` aus dem Paket `SentimentAnalysis`. Importieren Sie auch die `unittest`-Bibliothek.

▼ Klicken Sie hier für die Lösung

```
from SentimentAnalysis.sentiment_analysis import sentiment_analyzer
import unittest
```

3. Erstellen Sie die Unit-Test-Klasse. Nennen wir sie TestSentimentAnalyzer. Definieren Sie `test_sentiment_analyzer` als die Funktion, um die Unit-Tests auszuführen.

▼ Klicken Sie hier für die Lösung

```
class TestSentimentAnalyzer(unittest.TestCase):
    def test_sentiment_analyzer(self):
```

4. Definieren Sie 3 Unit-Tests in der genannten Funktion und überprüfen Sie die Gültigkeit der folgenden Aussage - Label-Paare.
   “I love working with Python”: “SENT_POSITIVE”
   “I hate working with Python”: “SENT_NEGATIVE”
   “I am neutral on Python”: “SENT_NEUTRAL”

▼ Klicken Sie hier für einen Hinweis
Verwenden Sie die Funktion `assertEqual`, um das `label` der Ausgabe mit dem erwarteten Label zu vergleichen.
▼ Klicken Sie hier für die Lösung

```
class TestSentimentAnalyzer(unittest.TestCase):
    def test_sentiment_analyzer(self):
        # Testfall für positives Sentiment
        result_1 = sentiment_analyzer('I love working with Python')
        self.assertEqual(result_1['label'], 'SENT_POSITIVE')
        # Testfall für negatives Sentiment
        result_2 = sentiment_analyzer('I hate working with Python')
        self.assertEqual(result_2['label'], 'SENT_NEGATIVE')
        # Testfall für neutrales Sentiment
        result_3 = sentiment_analyzer('I am neutral on Python')
        self.assertEqual(result_3['label'], 'SENT_NEUTRAL')
```

5. Rufen Sie die Unit-Tests auf.

▼ Klicken Sie hier für die Lösung
Fügen Sie die folgende Zeile am Ende der Datei hinzu.

```
unittest.main()
```

Jetzt, da die Datei bereit ist, führen Sie die Datei aus, um die Unit-Tests durchzuführen. Nach erfolgreicher Ausführung sollte die Ausgabe dieser Datei wie im Bild unten gezeigt aussehen.

Aufgabe 6: Bereitstellung als Webanwendung mit Flask

Jetzt, da die Anwendung bereit ist, ist es an der Zeit, sie über eine Weboberfläche bereitzustellen. Um den Bereitstellungsprozess zu erleichtern, wurden Ihnen 3 Dateien zur Verfügung gestellt, die für diese Aufgabe verwendet werden sollen.

- Überprüfen Sie die Verzeichnisstruktur:

```
practice_project/
├── SentimentAnalysis/
│   ├── __init__.py
│   ├── sentiment_analysis.py
├── templates/
│   ├── index.html
├── static/
│   ├── mywebscript.js
├── server.py
```

- Diese `index.html` im Ordner `templates` enthält den Code für die Weboberfläche, die für dieses Labor entworfen wurde. Diese wird Ihnen in vollständiger Form bereitgestellt und ist so zu verwenden, wie sie ist. Sie müssen keine Änderungen an dieser Datei vornehmen.

Die Benutzeroberfläche sieht wie auf dem Bild gezeigt aus.

- Durch Klicken auf die Schaltfläche `Run Sentiment Analysis` im HTML-Interface wird die JavaScript-Datei mywebscript.js im Ordner `static` aufgerufen, die eine GET-Anfrage ausführt und den vom Benutzer bereitgestellten Text als Eingabe nimmt. Dieser Text, der in einer Variablen namens `textToAnalyze` gespeichert ist, wird dann an die Serverdatei weitergeleitet, um an die Anwendung gesendet zu werden. Diese Datei wird Ihnen ebenfalls in vollständiger Form bereitgestellt und ist so zu verwenden, wie sie ist. Sie müssen keine Änderungen an dieser Datei vornehmen.
- Öffnen Sie `server.py` im Ordner `practice_project`. Diese Aufgabe dreht sich um den Abschluss dieser Datei. Sie können diese Datei abschließen, indem Sie die folgenden 5 Schritte ausführen.

a. *Importieren Sie die relevanten Bibliotheken und Funktionen*

In dieser Datei benötigen Sie die Flask-Bibliothek sowie die Funktion `render_template` (zum Bereitstellen der HTML-Datei) und die Funktion `request` (um die GET-Anfrage von der Webseite zu initiieren).

Sie müssen auch die Funktion `sentiment_analyzer` aus dem Paket `SentimentAnalysis` importieren.

Fügen Sie die relevanten Codezeilen zum Importieren der genannten Funktionen in `server.py` hinzu.

▼ Klicken Sie hier für die Lösung

```
from flask import Flask, render_template, request
from SentimentAnalysis.sentiment_analysis import sentiment_analyzer
```

b. *Initiieren Sie die Flask-Anwendung mit dem Namen `Sentiment Analyzer`*

Nutzen Sie das Wissen aus Modul 2 dieses Kurses und fügen Sie die Anweisung zu server.py hinzu, die die Anwendung initiiert und sie `Sentiment Analyzer` nennt.

▼ Klicken Sie hier für die Lösung

```
app = Flask("Sentiment Analyzer")
```

c. *Definieren Sie die Funktion `sent_analyzer`*

Zweck dieser Funktion ist es, eine GET-Anfrage an die HTML-Oberfläche zu senden, um den Eingabetext zu erhalten. Beachten Sie, dass die GET-Anfrage die Variable `textToAnalyze` referenzieren sollte, wie sie in der Datei `mywebscript.js` definiert ist. Speichern Sie den eingehenden Text in einer Variablen `text_to_analyze`. Als zweite Funktion rufen Sie Ihre Anwendung `sentiment_analyzer` mit `text_to_analyze` als Argument auf.

Formatieren Sie auch die Rückgabe der Funktion in einem formellen Text. Zum Beispiel:
`Der gegebene Text wurde als POSITIV mit einem Wert von 0.99765 identifiziert.`

▼ Klicken Sie hier für einen Hinweis

1. Verwenden Sie request.args.get, um die GET-Anfrage zu initiieren.
2. Das Label, das als “SENT_CLASS” empfangen wird (wobei die Klasse POSITIV, NEGATIV oder NEUTRAL sein kann), muss an ‘_’ aufgeteilt werden, um den Klassennamen einzeln zuzugreifen.

▼ Klicken Sie hier für die Lösung
Die Funktion sollte folgendermaßen aussehen.

```
@app.route("/sentimentAnalyzer")
def sent_analyzer():
    # Text zum Analysieren aus den Anfrageargumenten abrufen
    text_to_analyze = request.args.get('textToAnalyze')
    # Den Text an die Funktion sentiment_analyzer übergeben und die Antwort speichern
    response = sentiment_analyzer(text_to_analyze)
    # Das Label und den Wert aus der Antwort extrahieren
    label = response['label']
    score = response['score']
    # Eine formatierte Zeichenkette mit dem Sentiment-Label und dem Wert zurückgeben
    return "Der gegebene Text wurde als {} mit einem Wert von {} identifiziert.".format(label.split('_')[1], score)
```

Hinweis: Die Funktion verwendet den Flask-Dekorator `@app.route("/sentimentAnalyzer")`, wie im `mywebscript.js`-Datei referenziert.

d. *Rendern Sie die HTML-Vorlage mit `render_index_page`*

Diese Funktion sollte einfach die Funktion render_template für die HTML-Vorlage `index.html` ausführen.

▼ Klicken Sie hier für die Lösung

```
@app.route("/")
def render_index_page():
    return render_template('index.html')
```

e. *Führen Sie die Anwendung auf `localhost:5000` aus*

Schließlich, beim Ausführen der Datei, führen Sie die Anwendung auf dem Host: `0.0.0.0` (oder localhost) auf Portnummer 5000 aus.

▼ Klicken Sie hier für die Lösung

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

Um die Anwendung bereitzustellen, führen Sie die Datei *server.py* aus dem Terminal aus.

```
python3.11 server.py
```

Die Ausgabe würde so aussehen.

Die App läuft jetzt auf localhost:5000. Um auf die Anwendung zuzugreifen, gehen Sie zum Tab Skills Network Toolbox und klicken Sie auf Anwendung starten. Geben Sie den Anwendungsport als 5000 ein und klicken Sie auf Ihre Anwendung.

Die Benutzeroberfläche der Anwendung wird geöffnet. Nutzen Sie die Benutzeroberfläche, um Ihre Anwendung zu testen.

Um die Anwendung zu stoppen, drücken Sie `Ctrl+C`.

Aufgabe 7: Fehlerbehandlung einfügen

Um die Fehlerbehandlung einzufügen, müssen wir die verschiedenen Formen von Fehlercodes identifizieren, die als Antwort auf die GET-Anfrage, die von der `sent_analyzer`-Funktion in `server.py` initiiert wurde, empfangen werden können.

Dies ist bereits Teil der Funktionen der Watson NLP-Bibliothek und kann auf der Terminal-Konsole beobachtet werden, auf der der Code ausgeführt wird.

Betrachten Sie das unten gezeigte Bild.

Die Codes zeigen an, dass die ursprüngliche GET-Anfrage erfolgreich war (200), die Anfrage dann erfolgreich an die Watson-Bibliothek übertragen wurde (304) und die GET-Anfrage zur Generierung der Antwort ebenfalls erfolgreich durchgeführt wurde.

Im Falle ungültiger Eingaben antwortet das System mit dem Fehlercode 500, was darauf hinweist, dass etwas auf der Serverseite nicht stimmt. Eine ungültige Eingabe könnte alles sein, was das Modell nicht interpretieren kann. In dieser Fehlersituation wird die Ausgabe dieser Anwendung jedoch nicht aktualisiert.

Beachten Sie, dass die Ausgabe auf der Schnittstelle dieselbe ist wie zuvor, der zu analysierende Text ein zufälliger Text ist und die Watson AI-Bibliotheken einen 500-Fehler ausgeben, was bestätigt, dass das Modell die Anfrage nicht verarbeiten konnte.

Um diesen Fehler in unserer Anwendung zu beheben, müssen wir die Antwort studieren, die von der Funktion der Watson AI-Bibliothek empfangen wird, wenn der Server einen 500-Fehler generiert. Um dies zu testen, müssen wir die Schritte aus Aufgabe 2 zurückverfolgen und die Watson AI-Bibliothek mit einer ungültigen Zeichenfolgen-Eingabe testen.

Öffnen Sie eine Python-Shell im Terminal und führen Sie die folgenden Befehle aus, um die erforderliche Ausgabe zu überprüfen, nachdem Sie die Datei sentiment_analysis.py mit dem Folgenden aktualisiert haben.

```
import requests
# Define the URL for the sentiment analysis API
url = "https://sn-watson-sentiment-bert.labs.skills.network/v1/watson.runtime.nlp.v1/NlpService/SentimentPredict"
# Set the headers with the required model ID for the API
headers = {"grpc-metadata-mm-model-id": "sentiment_aggregated-bert-workflow_lang_multi_stock"}
# Define the first payload with nonsensical text to test the API
myobj = { "raw_document": { "text": "as987da-6s2d aweadsa" } }
# Make a POST request to the API with the first payload and headers
response = requests.post(url, json=myobj, headers=headers)
# Print the status code of the first response
print(response.status_code)
# Define the second payload with a meaningful text to test the API
myobj = { "raw_document": { "text": "Testing this application for error handling" } }
# Make a POST request to the API with the second payload and headers
response = requests.post(url, json=myobj, headers=headers)
# Print the status code of the second response
print(response.status_code)
```

Die Konsolenausgabe sieht wie im Bild unten dargestellt aus. Die roten Kästchen zeigen den ungültigen Text und den erhaltenen Statuscode an, während die gelben Kästchen den gültigen Text und den erhaltenen Statuscode anzeigen.

Dies ermöglicht es Ihnen, die Anwendung so zu modifizieren, dass wir unterschiedliche Ausgaben für verschiedene Statuscodes senden können.

Im ersten Teil dieser Aufgabe müssen Sie die `sentiment_analyzer()`-Funktion so ändern, dass sowohl `label` als auch `score` auf `None` gesetzt werden, falls ein ungültiger Texteingang vorliegt.

▼ Hier klicken für einen Hinweis
Erstellen Sie eine if-else-Bedingung in der Funktion sentiment_analyzer, um die erforderliche Funktionalität hinzuzufügen.
▼ Hier klicken für die Lösung
sentiment_analysis.py

```
import requests
import json
def sentiment_analyzer(text_to_analyse):
    # Definieren Sie die URL für die API zur Sentiment-Analyse
    url = 'https://sn-watson-sentiment-bert.labs.skills.network/v1/watson.runtime.nlp.v1/NlpService/SentimentPredict'
    # Erstellen Sie das Payload mit dem zu analysierenden Text
    myobj = { "raw_document": { "text": text_to_analyse } }
    # Setzen Sie die Header mit der erforderlichen Modell-ID für die API
    header = {"grpc-metadata-mm-model-id": "sentiment_aggregated-bert-workflow_lang_multi_stock"}
    # Führen Sie eine POST-Anfrage an die API mit dem Payload und den Headern durch
    response = requests.post(url, json=myobj, headers=header)
    # Analysieren Sie die Antwort von der API
    formatted_response = json.loads(response.text)
    # Wenn der Statuscode der Antwort 200 ist, extrahieren Sie das Label und den Score aus der Antwort
    if response.status_code == 200:
        label = formatted_response['documentSentiment']['label']
        score = formatted_response['documentSentiment']['score']
    # Wenn der Statuscode der Antwort 500 ist, setzen Sie Label und Score auf None
    elif response.status_code == 500:
        label = None
        score = None
    # Geben Sie das Label und den Score in einem Dictionary zurück
    return {'label': label, 'score': score}
```

Jetzt sollte die Antwort, die an die Konsole gesendet wird, auch für gültige und ungültige Eingabetypen unterschiedlich sein.
Für ungültige Eingaben lassen Sie die Konsole `Ungültige Eingabe! Versuchen Sie es erneut.` ausgeben.

▼ Hier klicken für einen Hinweis
Erstellen Sie eine if-else-Bedingung in der Funktion sent_analyzer von server.py, um zu überprüfen, ob "label" None ist oder nicht.
▼ Hier klicken für die Lösung

```
def sent_analyzer():
    # Abrufen des zu analysierenden Textes aus den Anfrageparametern
    text_to_analyze = request.args.get('textToAnalyze')
    # Übergeben Sie den Text an die Funktion sentiment_analyzer und speichern Sie die Antwort
    response = sentiment_analyzer(text_to_analyze)
    # Extrahieren Sie das Label und den Score aus der Antwort
    label = response['label']
    score = response['score']
    # Überprüfen Sie, ob das Label None ist, was auf einen Fehler oder ungültige Eingaben hinweist
    if label is None:
        return "Ungültige Eingabe! Versuchen Sie es erneut."
    else:
        # Geben Sie einen formatierten String mit dem Sentiment-Label und dem Score zurück
        return "Der gegebene Text wurde als {} mit einem Score von {} identifiziert.".format(label.split('_')[1], score)
```

Jetzt ist Ihre Anwendung in der Lage, angemessen auf jegliche Eingabeformen zu reagieren.

Aufgabe 8: Führen Sie eine statische Codeanalyse durch

Schließlich überprüfen wir in Aufgabe 8 die Qualität Ihrer Programmierfähigkeiten gemäß den PEP8-Richtlinien, indem wir eine statische Codeanalyse durchführen.

Normalerweise geschieht dies während der Paketierung und des Unit-Tests der Anwendung. Wir haben diesen Schritt jedoch ans Ende dieses Projekts verschoben, da die Codes in allen vorherigen Aufgaben aktualisiert wurden. Sobald Ihre Dateien für dieses Projekt bereit sind, lassen Sie uns testen, ob sie den PEP8-Richtlinien entsprechen.

Der erste Schritt in diesem Prozess besteht darin, die `PyLint`-Bibliothek über das Terminal zu installieren.

▼ Hier klicken für die Lösung

```
python3.11 -m pip install pylint
```

Verwenden Sie als Nächstes pylint, um eine statische Codeanalyse auf `server.py` durchzuführen.

▼ Hier klicken für die Lösung
Führen Sie im Terminal-Bash den folgenden Befehl aus.

```
pylint server.py
```

Wenn alle Aspekte des PEP8-Leitfadens in Ihrem Code berücksichtigt wurden, sollte die generierte Punktzahl 10/10 betragen. Falls dies nicht der Fall ist, folgen Sie den Anweisungen der pylint-Bibliothek, um den Code entsprechend zu korrigieren.

Damit ist das Übungsprojekt abgeschlossen.

(Optional) Zusätzliche Übungen

Interessierte Lernende können die folgenden Übungen eigenständig ausprobieren, um das Verständnis der im Rahmen dieses Projekts erlernten Konzepte zu vertiefen. Für diese Übungen wird keine Lösung bereitgestellt. Fühlen Sie sich jedoch frei, Ihre Lösungen mit Ihren Kommilitonen in den Diskussionsforen des Kurses zu besprechen und zu teilen.

1. Führen Sie eine statische Code-Analyse auf `sentiment_analysis.py` durch. Versuchen Sie, eine Punktzahl von 10/10 zu erreichen. Hinweis: *Docstrings*
2. Testen Sie die Fähigkeit Ihrer Anwendung, Sätze in anderen Sprachen als Englisch zu verarbeiten, z. B. Französisch, Deutsch usw. Überprüfen Sie, ob die Anwendung mit einer Fehlermeldung für ungültigen Text reagiert.
3. Derzeit, wenn die Anwendung OHNE Eingabe ausgeführt wird, d. h. wenn der Text leer gelassen wird, gibt das Modell dennoch denselben Fehler für ungültigen Text aus. Versuchen Sie, einen Sonderfall einzuführen, bei dem eine leere Eingabe eine andere Fehlermeldung erhält.

Fazit

Herzlichen Glückwunsch zum Abschluss dieses Projekts.

Mit dem Abschluss dieses Projekts haben Sie:

1. Eine KI-basierte Sentiment-Analyse-Anwendung mit eingebetteten Watson NLP-Bibliotheken erstellt.
2. Die Ausgabe, die von der Funktion der Watson NLP-Bibliothek erhalten wurde, formatiert, um relevante Informationen daraus zu extrahieren.
3. Die Anwendung verpackt und importierbar für jede Python-Code-Nutzung gemacht.
4. Unit-Tests für die Anwendung durchgeführt und die Gültigkeit ihrer Ausgaben für verschiedene Eingaben überprüft.
5. Die Anwendung mit dem Flask-Framework bereitgestellt.
6. Eine Fehlerbehandlungsfunktion in die Anwendung integriert, sodass ein Antwortcode von 500 eine angemessene Antwort von der Anwendung erhält.
7. Eine statische Code-Analyse der Code-Dateien durchgeführt, um deren Einhaltung der PEP8-Richtlinien zu bestätigen.