

NWEN 243 – 2024T2

Project 3 – Part B (2 of 2 parts): Deploying a Sentiment Analysis Model on AWS EC2

Note: Part **B** is new in 2024. The developer of this lab was Taran John, VUW, along with various edits by Kris.

Due: See the **submission system** for authoritative dates and times, however **both** parts **A** and **B** will be submitted together.

FOR SUBMISSION:

- In terms of supporting evidence, it is up to you to take appropriate **screenshots** and make commentary.
 - You must put together a folio of evidence supporting your work in this lab.
 - In your screenshots ensure any IP addresses and/or instance names are visible.
 - Collect your screen shots and organise into a PDF with a narrative explaining what is going on, what each screen shot reveals and how they line up to the sections in this Project.
 - Label this narrative, **Project3partB.pdf**
 - You've seen what we're after, by example from Project 2, you need to provide a similar level of evidence here and this is reflected in the marking criteria. Handing in some working code and the pdf equivalent of crumpled paper won't get you the marks you'd like - there needs to be evidence of introspection and understanding.
 - There are also a couple of questions to answer at the end - you could make some tests or experiments and in general provide an insightful commentary of what is going on.
-

TODO

Students will build a machine learning model for sentiment analysis, package it using Docker and Amazon ECR, deploy it to an AWS EC2 instance, and use a client script to interact with the deployed model.

The sentiment training data used is the conveniently packaged (for python) imdb dataset.

Lab Outline

Part 1: EC2 Setup

- Launching an EC2 instance
- Configuring security groups
- Installing necessary development tools (Docker, Python, Pip, etc.)

Part 2: Introduction to Sentiment Analysis and Model Training

- Brief overview of sentiment analysis
- Explanation of the model architecture (logistic regression)
- Training the sentiment analysis model using provided dataset

Part 3: Packaging the Model

- Writing an inference script with API endpoint
- Creating a Dockerfile
- Building and testing the Docker image locally

Part 4: Creating a Client Script

- Writing a Python script to send requests to the API
- Handling command-line arguments for user input

Part 5: Questions

Part 1: EC2 SETUP

In this section, you'll launch an EC2 instance and set it up for development by installing necessary tools including Docker, Python, and pip.

Step 1: Launch an EC2 Instance

1. Sign in to the AWS Management Console and open the EC2 console.
2. Click "Launch Instance"
3. Choose **Amazon Linux 2023 AMI** (do not choose the OS we used in lab project 2)
4. Select **t2.small** instance type.
 - This will cost USD0.32 per hour
 - so make sure you kill it after you are done.
5. Configure Security Group:
 - Allow SSH (port 22) from your IP
 - Add a rule to allow HTTP (port 80) from anywhere
 - Add a rule to allow Custom TCP (port 32000) from anywhere
8. The default 8GB storage is fine
9. Review and launch
10. Create a new key pair or select an existing one, then launch the instance

Step 2: Update and Install Dependencies

Connected to your EC2 instance, and run the following commands:

```
# Update the package manager and installed packages  
sudo yum update -y
```

The latest version of python 3 already ought be installed but this will install both Python3 and pip3 as needed (pip 3 is definitely not installed by default).

```
# Install Python 3 and pip  
sudo yum install python3 python3-pip -y
```

```
# Verify Python and pip installation  
python3 --version
```

```
Python 3.9.16
```

```
pip3 --version
```

```
pip 21.3.1 from /usr/lib/python3.9/site-packages/pip (python 3.9)
```

Make sure you do the following:

```
# Install required Python Libraries  
pip3 install pandas scikit-learn datasets joblib flask --user 2>&1 | grep -v  
"pip's dependency resolver"
```

Verify installations

```
pip3 list | grep -E "pandas|scikit-learn|datasets|joblib|flask"
```

Mine outputs:

datasets	3.0.0
joblib	1.4.2
pandas	2.2.2
scikit-learn	1.5.2

Step 3: Install and Configure Docker

Install Docker on your EC2 instance:

Install Docker

```
sudo yum install docker -y
```

Start the Docker service

```
sudo systemctl start docker
```

Enable Docker to start on boot

```
sudo systemctl enable docker
```

Add the ec2-user to the docker group so you can execute Docker commands without using sudo

```
sudo usermod -a -G docker ec2-user
```

Verify Docker installation

```
docker --version
```

Mine outputs

Docker version 25.0.5, build 5dc9bcc

Log out and log back in for the group changes to take effect.

Step 4: Set Up Your Project Directory

Create a directory for your project:

```
mkdir ~/sentiment-analysis-project  
cd ~/sentiment-analysis-project
```

You're now ready to start developing your sentiment analysis project directly on the EC2 instance!

Part 2: Introduction to Sentiment Analysis

What is Sentiment Analysis?

Sentiment analysis is a natural language processing (NLP) technique used to determine the emotional tone behind a piece of text. It involves classifying text as positive, negative, or neutral. This technology has numerous applications, including:

- Analysing customer feedback
- Monitoring social media sentiment
- Assessing product reviews
- Gauging public opinion on various topics

In this lab project, as mentioned earlier, we're using the imdb dataset (movies) for training.

How Does It Work?

At its core, sentiment analysis works by:

1. Preprocessing the text (removing punctuation, lowercasing, etc.)
2. Converting text into numerical features (e.g., using techniques like bag-of-words or word embeddings)
3. Using a machine learning model to classify the sentiment based on these features

Sentiment analysis models are typically trained on large datasets of labeled text. In this project, our dataset consists of movie reviews labeled as positive or negative. The model learns to associate certain words or patterns with positive or negative sentiment.

In practice, more advanced techniques like deep learning models or pre-trained language models (e.g., BERT) can be used for improved performance.

Here is some basic Python code that creates a simple model, and outputs it. Create a python file for this called `training_script.py` on your ec2 machine by using:

```
cat > training_script.py
```

and paste the following code:

```
import pandas as pd
from datasets import load_dataset
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
import joblib

# Load the IMDB dataset
dataset = load_dataset("imdb")

# Convert the dataset to a pandas DataFrame
train_df = pd.DataFrame(dataset['train'])
test_df = pd.DataFrame(dataset['test'])
```

```

# Combine train and test for simplicity (you might want to keep them separate
in practice)
df = pd.concat([train_df, test_df], axis=0, ignore_index=True)

# Display the first few rows
print(df.head())

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df['text'], df['label'],
test_size=0.2, random_state=42)

# Convert text to numerical features using bag-of-words
vectorizer = CountVectorizer(max_features=5000) # Limit to top 5000 words
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)

# Train a Logistic regression model
model = LogisticRegression(random_state=42, max_iter=1000)
model.fit(X_train_vectorized, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test_vectorized)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Function to predict sentiment
def predict_sentiment(text):
    # Vectorize the input text
    text_vectorized = vectorizer.transform([text])
    # Predict the sentiment
    prediction = model.predict(text_vectorized)
    # Return the sentiment Label
    return "Positive" if prediction[0] == 1 else "Negative"

# Test the model with some example reviews
print("\nTesting the model:")
print("Sentiment:", predict_sentiment("This movie was fantastic! I really
enjoyed it.))
print("Sentiment:", predict_sentiment("I didn't like this film at all. It was
boring and poorly acted.))

# Save the model and vectorizer
joblib.dump(model, 'sentiment_model.joblib')
joblib.dump(vectorizer, 'vectorizer.joblib')

```

```

print("\nModel and vectorizer saved in the local directory.")

# Demonstrate Loading and using the saved model
loaded_model = joblib.load('sentiment_model.joblib')
loaded_vectorizer = joblib.load('vectorizer.joblib')

print("\nTesting the loaded model:")
def predict_sentiment_loaded(text):
    text_vectorized = loaded_vectorizer.transform([text])
    prediction = loaded_model.predict(text_vectorized)
    return "Positive" if prediction[0] == 1 else "Negative"

print("Sentiment:", predict_sentiment_loaded("A great watch! Highly
recommended."))
print("Sentiment:", predict_sentiment_loaded("Don't waste your time on this
one."))

```

remember to press ^D to signal to cat, that this is the end of the file.

Double check you managed to paste the whole thing (and didn't paste the `` marks) using:

```
less training_script.py
```

I will also put the file on the ECS website separately. If you're having issues pasting it, you can scp it to the instance.

Training a Sentiment Analysis Model

Now, let's go through the process of running the training script.

1. Ensure you're in the correct directory:
`cd ~/sentiment-analysis-project`
2. Run the script:
`python3 training_script.py`

The script will take some time to run as it downloads the dataset, trains the model, and saves the results.

Verifying the Output

After the script finishes running it will output some test results, a report on the models classification accuracy. You should see two new files:

- `sentiment_model.joblib`
- `vectorizer.joblib`

These files contain your trained model and the vectorizer, respectively. They will be used in the next steps of the project for sentiment analysis predictions.

Part 3: Packaging the Model

In this section, you'll create an inference script with an API endpoint, create a Dockerfile, and build and test your Docker image locally.

Step 1: Writing the Inference Script

Create a new file (in the same way as before) named `app.py` with the following content:

```
from flask import Flask, request, jsonify
import joblib
import pandas as pd

app = Flask(__name__)

# Load the model and vectorizer
model = joblib.load('sentiment_model.joblib')
vectorizer = joblib.load('vectorizer.joblib')

@app.route('/predict', methods=['POST'])
def predict():
    data = request.json
    text = data['text']

    # Vectorize the input text
    text_vectorized = vectorizer.transform([text])

    # Make prediction
    prediction = model.predict(text_vectorized)

    # Convert prediction to sentiment
    sentiment = "Positive" if prediction[0] == 1 else "Negative"

    return jsonify({'sentiment': sentiment})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=32000)
```

This script creates a Flask API with a `/predict` endpoint that accepts POST requests with JSON data containing a 'text' field. It uses the pre-trained model and vectorizer to make predictions.

Step 2: Creating the Dockerfile

Create a new file named Dockerfile in the same directory as app.py with the following content:

```
# Use an official Python runtime as the base image
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install the required packages
RUN pip install --no-cache-dir flask gunicorn joblib scikit-learn pandas

# Make port 32000 available to the world outside this container
EXPOSE 32000


# Run app.py when the container launches
CMD ["gunicorn", "--bind", "0.0.0.0:32000", "app:app"]
```

This Dockerfile sets up a Python environment, installs necessary packages, and configures the container to run your Flask app using Gunicorn.

Step 3: Building the Docker Image

In the sentiment-analysis-project directory containing your app.py, Dockerfile, sentiment_model.joblib, and vectorizer.joblib files, then run:

```
docker build -t sentiment-analysis-app .
```



This command builds a Docker image named sentiment-analysis-app based on your Dockerfile. It may take a while, even on the small ec2 instance.

Step 4: Testing the Docker Image Locally

After the build completes, run the container locally:

```
docker run -d -p 32000:32000 sentiment-analysis-app
```

This command runs your Docker container and maps port 32000 of the container to port 32000 on your local machine.

Step 5: Testing the API

Open a new terminal window and use curl to test your API:

```
curl -X POST -H "Content-Type: application/json" -d '{"text":"This movie was fantastic!"}' http://localhost:32000/predict
```

You should receive a JSON response with the predicted sentiment.

```
{"sentiment":"Positive"}
```

Try the alternative as well - just for fun and completeness:

```
curl -X POST -H "Content-Type: application/json" -d '{"text":"This movie was awful!"}' http://localhost:32000/predict
```

Try some other statements, such as average, and see what sentiment the model assigns.

Step 6: Verify the Endpoint from Another Machine

On your local machine (in another terminal window) or any other machine except the ec2 instance itself, use curl to test the endpoint:

```
curl -X POST -H "Content-Type: application/json" -d '{"text":"This movie was fantastic!"}' http://<your-ec2-public-ip>:32000/predict
```

Replace <your-ec2-public-ip> with your EC2 instance's public IP address.

You should receive the same JSON response as you did when testing locally on the EC2 instance.

This is also the time to tell you if you got the Security Group wrong, you don't have to start again. Assume for instance you put the wrong port number. You go to the instances page (refresh), then click on the 'Instance ID'. Scroll down to the security tab and then click on the link under Security groups. You can now edit your SG on this page. Yay.

Troubleshooting

If you encounter any issues:

1. Ensure your EC2 security group allows inbound traffic on port 32000 from anywhere (0.0.0.0/0)
2. Check that Docker is running on your EC2 instance (`sudo service docker status`)
3. Verify that your Docker container is running (`docker ps`)
4. Check the Docker logs for any errors (`docker logs <container_id>`)
5. If you can access the endpoint locally on EC2 but not externally, double-check your security group settings
6. Ensure your EC2 instance is in a public subnet with an Internet Gateway in your VPC

Part 4: Creating a Client Script

In this step, you'll create a client script that takes two command-line arguments: the EC2 instance's IP address and the text to analyse. The script will send the text to your deployed sentiment analysis endpoint and display the result. You can choose either Python or Java. You need to run this client script from your own machine, as you did with curl in step 6, not the EC2 instance. This could also be second ec2 instance, if you so choose.

Option 1: Python Client Script

Create a new file named `sentiment_client.py` with the following content:

```
import requests
import sys

def analyze_sentiment(ip_address, text):
    url = f"http://{ip_address}:32000/predict"
    payload = {"text": text}

    try:
        response = requests.post(url, json=payload)
        response.raise_for_status()
        result = response.json()
        return result['sentiment']
    except requests.exceptions.RequestException as e:
        return f"An error occurred: {e}"

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: python sentiment_client.py <ec2-public-ip> \"<text-to-analyze>\"")
        sys.exit(1)

    ip_address = sys.argv[1]
    text = sys.argv[2]

    sentiment = analyze_sentiment(ip_address, text)
    print(f"Text: {text}")
    print(f"Sentiment: {sentiment}")
```

Run the script:

```
python sentiment_client.py <your-ec2-public-ip> 'some comment'
```

You **might** need to install the requests library on your own python3 install.

```
pip3 install requests
```

This is my output

```
python3 sentiment_client.py 54.89.154.252 "Bilbo should have flown the eagles to Mordor. Silly Hobbit."
```

Text: Bilbo should have flown the eagles to Mordor. Silly Hobbit.
Sentiment: Negative

Option 2: Java Client Script

Create a new file named `SentimentClient.java` with the following content:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.charset.StandardCharsets;

public class SentimentClient {

    public static String analyzeSentiment(String ipAddress, String text) {
        try {
            URL url = new URL("http://" + ipAddress + ":32000/predict");
            HttpURLConnection con = (HttpURLConnection) url.openConnection();
            con.setRequestMethod("POST");
            con.setRequestProperty("Content-Type", "application/json; utf-8");
            con.setRequestProperty("Accept", "application/json");
            con.setDoOutput(true);

            String jsonString = "{\"text\": \"" + text + "\"}";

            try(OutputStream os = con.getOutputStream()) {
                byte[] input =
jsonInputString.getBytes(StandardCharsets.UTF_8);
                os.write(input, 0, input.length);
            }

            try(BufferedReader br = new BufferedReader(new
InputStreamReader(con.getInputStream(), StandardCharsets.UTF_8))) {
                StringBuilder response = new StringBuilder();
                String responseLine = null;
                while ((responseLine = br.readLine()) != null) {
                    response.append(responseLine.trim());
                }
                return response.toString();
            }
        } catch (Exception e) {
```

```

        return "An error occurred: " + e.getMessage();
    }
}

public static void main(String[] args) {
    if (args.length != 2) {
        System.out.println("Usage: java SentimentClient <ec2-public-ip>
\"<text-to-analyze>\"");
        System.exit(1);
    }

    String ipAddress = args[0];
    String text = args[1];

    String sentiment = analyzeSentiment(ipAddress, text);
    System.out.println("Text: " + text);
    System.out.println("Sentiment: " + sentiment);
}
}

```

Compile and run the Java script:

```

javac SentimentClient.java
java SentimentClient <your-ec2-public-ip> 'some comment'

```

This is my output

```

java SentimentClient 54.89.154.252 "Merry and Pippin were very annoying
Hobbits, in my opinion Frodo was the best Hobbit."

```

Text: Merry and Pippin were very annoying Hobbits, in my opinion Frodo was the best Hobbit.

Sentiment: {"sentiment":"Negative"}

Perhaps my points of discussion are a bit too subtle for the simple sentiment model we built in this lab.

Expected Behavior

- The script should take two command-line arguments: the EC2 instance's IP address and the text to analyse.
- It will send the text to your deployed model and display both the input text and the predicted sentiment.
- If there's an error (e.g., cannot connect to the server), it should display an error message.

Troubleshooting

- Ensure your EC2 instance is running and the Docker container is active.
- Double-check that port 32000 is open in your EC2 security group.
- Verify that you're using the correct public IP address of your EC2 instance.
- If using Java, make sure you have JDK installed and properly set up in your system path.
- For Python, ensure you have the requests library installed (`pip install requests`).
- If your text contains spaces or special characters, remember to enclose it in quotes when running the script.

Part 5: Questions

There is only one question.

Given what you know now, from lab projects 2 and 3 - and material covered in the lectures, how would you architect a scalable version of our sentiment analyser.

Think about the pros and cons, think about the requirements, data and processing, bottle necks, type of parallelism we need, and most importantly - draw your marker a diagram of your solution.

This answer is limited to 1 page of PDF 12pt font, normal margins, including any diagrams.