Sentiment Analysis Model Documentation

Project Overview

OBJECTIVE:

The goal of this project is to develop a sentiment analysis model that categorizes review sentences into positive, neutral, and negative classes. Each sentence in a review inherits the review's star rating.

APPROACH:

The project uses a combination of machine learning techniques, including TF-IDF, Doc2Vec, and CatBoost, to train a classifier. The model is deployed locally with a focus on achieving a p99 latency of 300ms for predictions.

Model Description

MODEL CHOICE:

CatBoost Classifier: Chosen for its efficiency and ability to handle categorical data without extensive preprocessing.
TF-IDF Vectorizer: Used to convert text data into numerical form by capturing the importance of words.
Doc2Vec: Provides vector representations of sentences, capturing semantic meaning.
Reference: https://medium.com/data-science-lab-spring-2021/amazon-review-rating-prediction-with-nlp-28a4acdd4352
My first choice was BERT, and I also designed the model. You can check the other branch (`new_branch`) on GitHub. However, due to Starlink's limitations, building a Docker image was challenging, especially when trying to install PyTorch (both full and CPU versions), as it frequently timed out. The trained BERT model is saved in https://storage.googleapis.com/my-sentiment-models/bert_sentiment_model.pt. I have to say, BERT outperforms CatBoost.

TRAINING:

Dataset: The model is trained on a dataset of book reviews, where each review is split into sentences.
Preprocessing: Includes tokenization, stopword removal, and sentiment analysis using TextBlob.
Performance Metrics: Evaluated using accuracy, precision, recall, and F1-score.

Performance:
The model achieves a mean accuracy of approximately 0.85 across cross-validation folds. Confusion matrix and classification reports are generated to assess performance.

System Design

ARCHITECTURE:

The system consists of a Flask-based API server (`model_server.py`) that handles prediction requests.
The model is loaded into memory at startup, and predictions are served via HTTP endpoints.

DEPLOYMENT:

The model is containerized using Docker, allowing for easy deployment and scaling.
A CI/CD pipeline is set up using GitHub Actions to automate testing and deployment.

CLOUD READINESS:

The system is designed to be easily deployable to cloud environments, such as Google Cloud Run, using Docker images.

Logging and Monitoring

LOGGING:

Logs are captured using Python's `logging` module, recording important events and errors.

MONITORING:

1. The `/metrics` endpoint provides real-time metrics, including request count and latency statistics.

2. Google cloud online monitoring, please to check below Figure GCP

Code Structure

FILE DESCRIPTIONS:

`review_prediction.py`: Contains the main logic for training the sentiment analysis model.
`model_server.py`: Implements the Flask server for serving predictions.
`test_model_server.py`: Contains tests for the model server, including latency measurements.
`integration_test.py` and `test_sentiment_model.py`: Unit and integration tests for the model and server.
load_test.py:
simulates multiple concurrent requests to evaluate how well the server handles high traffic and to measure the latency of responses under load.
README.md:
`Dockerfile`: Defines the Docker image for the server.

`.github/workflows/ci-cd.yml`: CI/CD pipeline configuration.

Key Functions and Classes:

```
### Installation
git clone https://github.com/Tofu0142/TP.git
cd TP


2. Install dependencies:
pip install -r requirements.txt


3. Download NLTK resources (automatically handled by the scripts, but can be done manually):

import nltk
nltk.download(['punkt', 'stopwords', 'wordnet'])
```

Usage

Training a new model
python review_prediction.py

Running the API server
python model_server.py

Making predictions
import requests
import json

# Single prediction
response = requests.post(
    "http://localhost:8080/predict",
    json={"text": "This book was absolutely fantastic! I couldn't put it down."}
)
print(response.json())

# Batch prediction
response = requests.post(
    "http://localhost:8080/predict_batch",
    json={"texts": [
        "This book was absolutely fantastic! I couldn't put it down.",
        "The characters were poorly developed and the plot was predictable.",
        "It was an okay read, nothing special but not terrible either."
    ]}
)
print(response.json())

Cloud Deployment

The API is deployed and available on Google Cloud Run at:
[https://sentiment-analysis-438649044905.us-central1.run.app](https://sentiment-analysis-438649044905.us-
central1.run.app)

Testing the Cloud API

You can test the deployed API using curl:

Health Check
curl https://sentiment-analysis-438649044905.us-central1.run.app/health

Single Prediction
curl -X POST \
  https://sentiment-analysis-438649044905.us-central1.run.app/predict \
  -H "Content-Type: application/json" \
  -d '{"text": "This book was absolutely fantastic! I could not put it down."}'

Batch Prediction
curl -X POST \
  https://sentiment-analysis-438649044905.us-central1.run.app/predict_batch \
  -H "Content-Type: application/json" \
  -d '{
    "texts": [
      "This book was absolutely fantastic! I could not put it down.",
      "The characters were poorly developed and the plot was predictable.",
      "It was an okay read, nothing special but not terrible either."
    ]
  }'

Docker Deployment
1. Build the Docker image:
docker build -t sentiment-analysis .

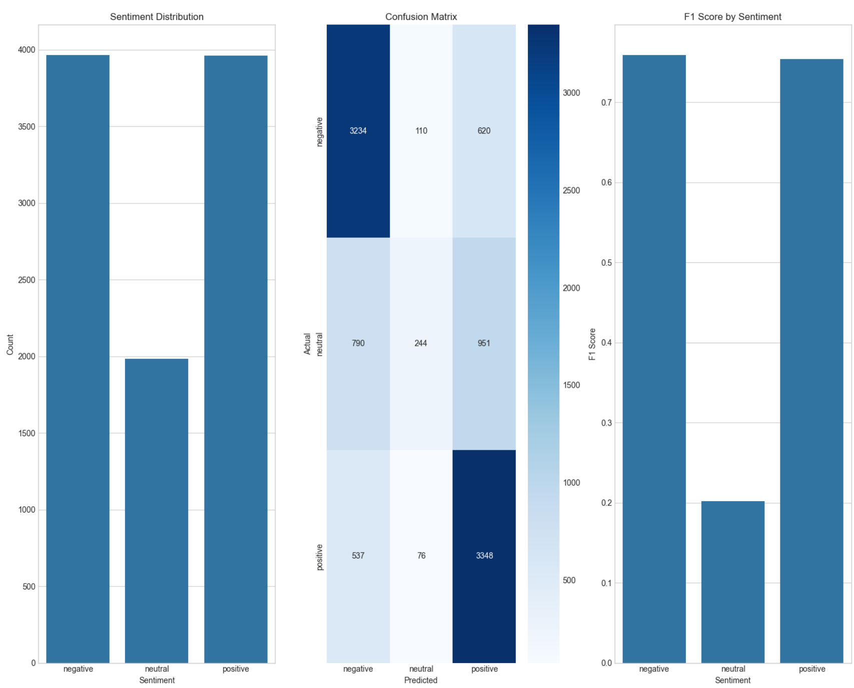
2. Run the container:

docker run -p 8080:8080 sentiment-analysis
```
```

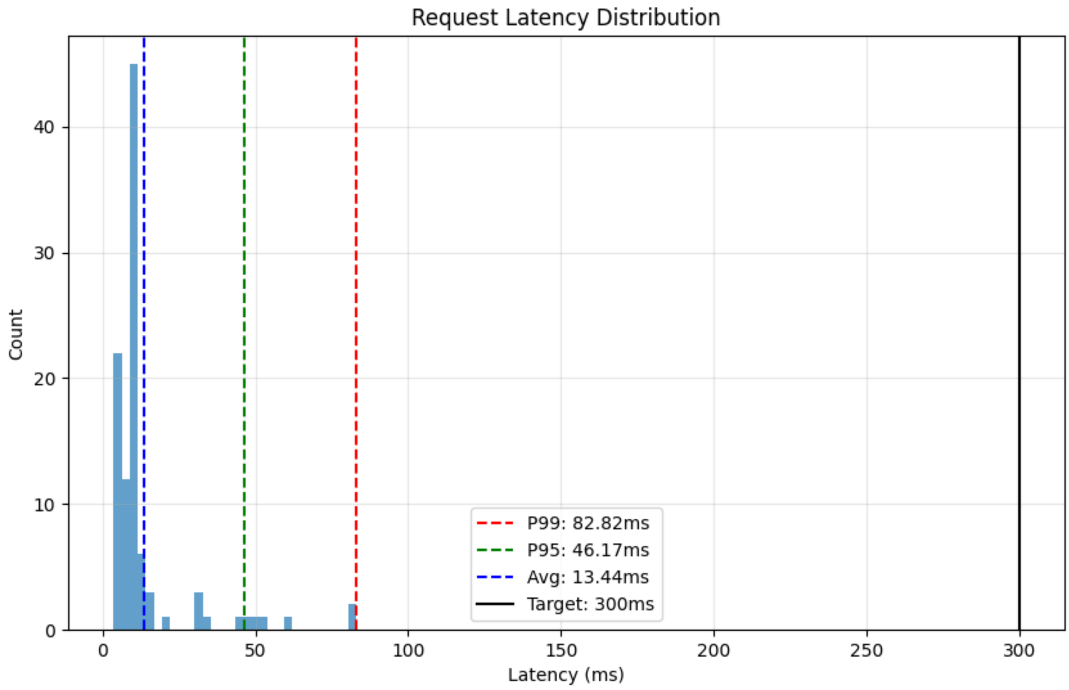
Figures and Diagrams

Data Analysis:

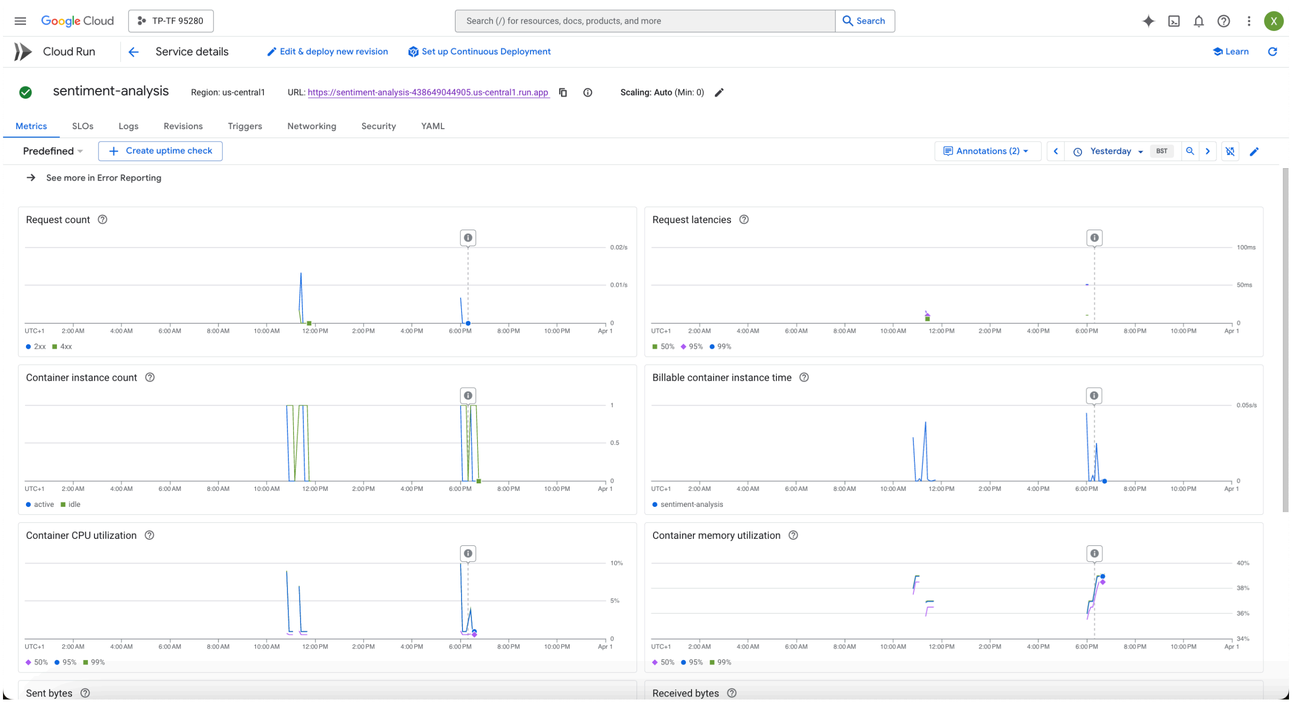
Confusion Matrix: Visualizes the model's performance on test data.



Latency Distribution Plot:
Shows the distribution of request latencies during load testing - 100 requests.



GCP :



Conclusion

The sentiment analysis model effectively categorizes review sentences with high accuracy and low latency. The system is robust, scalable, and ready for deployment in both local and cloud environments.