CS 816 - Software Production Engineering

MAJOR PROJECT REPORT

CENSOR Quotient - UNCOVERING HARSH TEXTS

Team Members

N V Sai Likhith - IMT2020118

Tarun Kumar Gupta - IMT2020061

Introduction

This document reports the work done as part of the Software Production Engineering Course Project. The project involves development of a website that detects the sentiment of a X(formerly Twitter) user's tweet. The project was built using DevOps practices such as SCM, Continuous Integration, Continuous Deployment and Continuous Monitoring. The website and practices are explained in detail in the following sections.

Important Links

Link to GitHub repository that is used for SCM and contains the source code for all the files.

Link to the Docker Hub containing the images.

Link to the kaggle notebook that contains the scripts used to fine-tune the LLM BERT Model.

Link to the Hugging face repository that contain the fine-tuned models for predicting harsh, vulgar and threatening sentiments.

Censor Quotient - Uncovering Harsh Texts

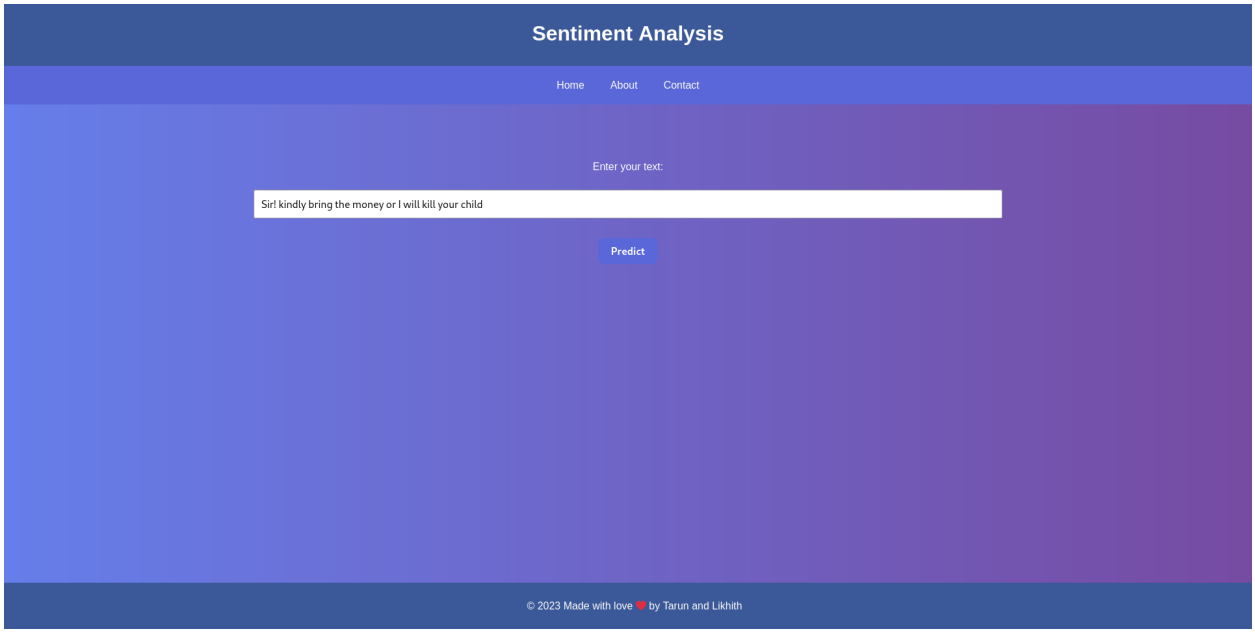
The Website

The website has two windows that the end-user can access to perform the main task of detecting the sentiment of a tweet. The Home page accepts input from the user either in text format or link to X(formerly Twitter). Users need not worry about specifying if it is text or link as the website automatically detects if the link belongs to the X platform.

The second page is the 'results' page where users can get to know the sentiment of the input he provided. The input is checked for three kinds of sentiments - harsh, vulgar and

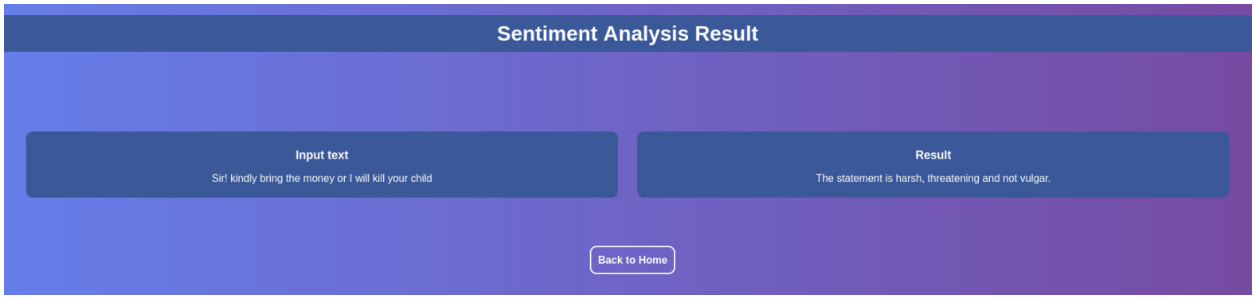
threatening. The output specifies if the text belongs or doesn't belong to each of the three categories.

Here, I will provide a few screenshots of the website.



Above statement is : **Sir! kindly bring the money or I will kill your child**

We can see that the statement is harsh and threatening but not vulgar and it predicted the same.



Included one more example of a statement which is vulgar but not threatening. More examples are provided in the viva.

Source Code

Here, I will give a brief explanation of the source code for developing the website in a module-wise manner

Flask

Flask is a lightweight and versatile web framework for Python used to build web applications. It can handle HTTP requests and provide mechanisms to access data.

POST

A post request is made to URL /sentiment-analysis once the user hits predict button(submits the form). Which then calls predict_sentiment function. This function fetches the text that the user has inputted and send the request to backend for predictions. On successful prediction, the result is displayed on the website.

Extracting Content from Twitter

```python
# Function to extract text from a Twitter link using Apify
def extract_text_from_twitter_link(tweet_url):
    # Initialize the ApifyClient with your API token
    client = ApifyClient("apify_api_2QGQH8DkI54nayPr5grpIauvPmWqFL3NwQ4M")

    run_input = {
        "startUrls": [{"url": tweet_url}],
        "tweetsDesired": 1,
        "addUserInfo": False,
    }

    run = client.actor("KVJr35xjTw2XyvMeK").call(run_input=run_input)

    # Fetch and return the full_text of the tweet
    for item in client.dataset(run["defaultDatasetId"]).iterate_items():
        return item.get("full_text", None)
```

ApifyClient is an API provider for languages like Javascript and Python. It can be integrated in the codes to perform API calls to websites like X. Here, we leveraged it to perform API

calls to X and get the content of the tweet. As a side note, using Apify Client we can extract even more data just rather than the text of the tweet. Visit their website for more details.

Input Processing

```python
@app.route('/sentiment-analyzer', methods=['POST'])
def predict_sentiment():
    try:
        if request.method == 'POST':
            text_input = request.form.get('text_input', '')

            # Check if the input is a Twitter link
            if text_input.startswith('https://twitter.com/') or text_input.startswith('https://x.com/
                # Extract text from the Twitter link
                tweet_text = extract_text_from_twitter_link(text_input)
                if tweet_text:
                    text_input = tweet_text
                else:
                    raise Exception("Error extracting text from Twitter URL")

            # Perform sentiment analysis using the backend API
            response = requests.post("http://backend:6000/", files={'user_text': text_input})

            label = 'The statement is '

            if response.status_code == 200:
                predictions = response.json().get('predictions')

                n = len(predictions)

                for i, key in enumerate(predictions):
                    key = str(key)
                    if key.startswith('NOT_'): key = 'not ' + key.split('_')[-1]
                    label += str(key)
                    if i < n-2: label += ', '
                    elif i == n-2: label += ' and '
                    elif i == n-1: label += '.'

                app.logger.info("Prediction for input text: %s is  %s",str(text_input), str(label))
```

BERT

BERT (Bidirectional Encoder Representations from Transformers) is a natural language processing (NLP) model developed by Google. BERT has significantly advanced the field of machine learning by excelling in tasks such as language understanding and contextual

reasoning. Unlike traditional models, BERT employs a bidirectional transformer architecture, enabling it to consider the entire context of a word by looking at both left and right contexts simultaneously. This bidirectional approach results in a more nuanced understanding of language semantics and has led to remarkable improvements in tasks like question answering, sentiment analysis, and language translation. BERT's pre-training on vast amounts of diverse textual data allows it to capture intricate linguistic nuances and context-dependent meanings, making it a cornerstone in the development of state-of-the-art NLP models and applications.

Predicting the sentiment

On running python3 app.py in the frontend folder, we can see our website on http://127.0.0.1:5000/. Index.html page is served for this url. On inputting the text and hitting the predict button, the URL changes to http://127.0.0.1:5000/sentiment-analysis which serves the result.html page. predict_sentiment function runs on clicking the predict button which sends a request to the backend for making the predictions. On successful prediction, results can be seen on the website.

Techstack and DevOps

Frontend - HTML, CSS, Javascript

Backend - Python for making ML models

Flask - To connect frontend with ML models.

Git: Git is a distributed version control system that facilitates collaborative software development. It helps track changes in the codebase, manage different versions of the projects, and collaborate seamlessly with others.

Github: GitHub is a web-based platform built on top of Git. It enhances Git's functionality by providing a centralized hub for hosting and collaborating on projects. GitHub offers a user-friendly interface, issue tracking, pull requests, and a social component that fosters community and collaboration.

Jenkins: Jenkins is an open-source automation server that plays a pivotal role in supporting Continuous Integration (CI) and Continuous Delivery (CD) in software development. It helps to automate a wide variety of tasks in our project like model training, testing, deployment, and monitoring. This increases the efficiency, consistency and manageability. The automated nature of these workflows significantly diminishes the probability of errors, expedites deployment cycles, and elevates the overall dependability of our machine learning application.

Docker: As a containerization platform, Docker allows developers to encapsulate applications and their dependencies into lightweight, portable containers. These containers ensure consistency across various environments, from development to production, by encapsulating everything needed to run the application. Docker's efficiency lies in its ability to isolate applications, making them highly portable, scalable, and easily reproducible. This approach simplifies deployment processes, accelerates development cycles, and enhances collaboration by providing a standardized environment.

Docker Compose: Docker Compose is a powerful tool that simplifies the orchestration of multi-container Docker applications. Serving as a companion to Docker, Compose enables developers to define and manage complex, multi-service applications in a single, declarative configuration file. This file outlines the services, networks, and volumes required for an application, streamlining the process of setting up and connecting multiple containers. Docker Compose not only enhances collaboration by ensuring consistency across development, testing, and production environments but also facilitates the scaling and deployment of applications with a straightforward command. Its user-friendly approach allows developers to define, configure, and run multi-container Docker applications seamlessly, contributing to the efficiency and reproducibility of the entire development lifecycle.

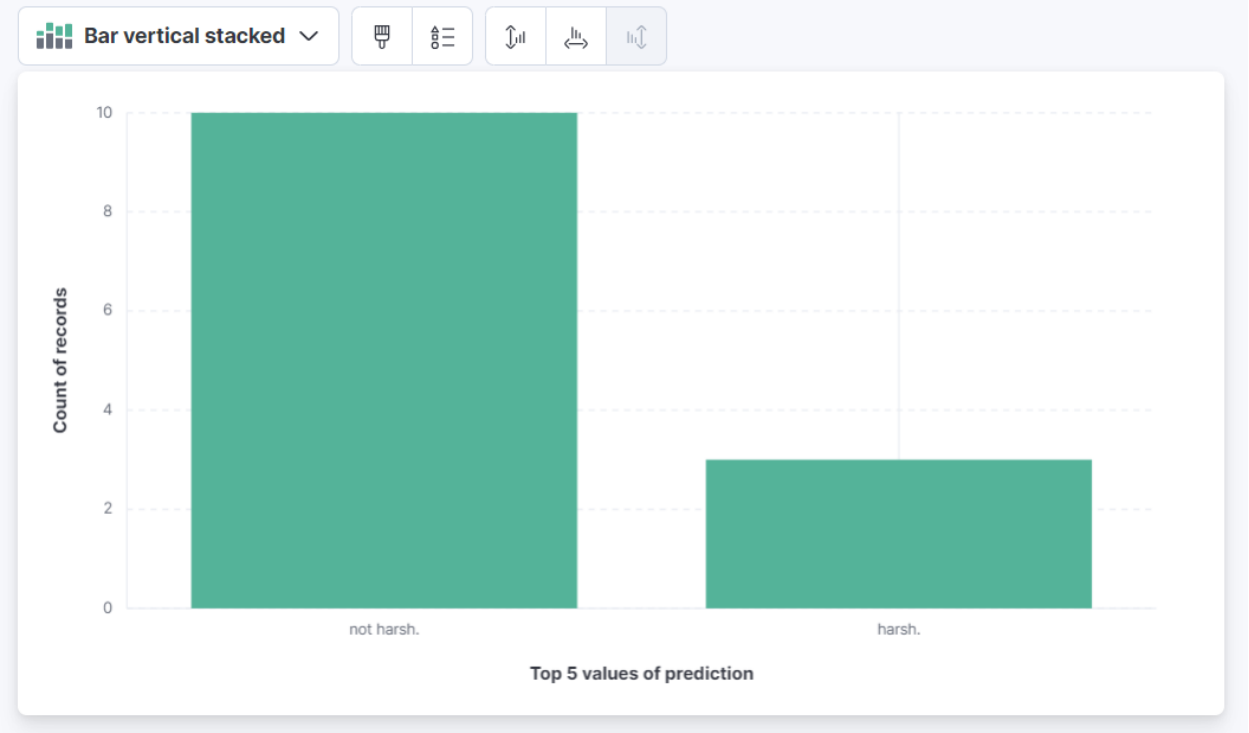
```yaml
services:
  frontend:
    image: nvsailikhith/sentiment_analyzer_frontend:latest
    container_name: sentiment_analysis_frontend
    stdin_open: true
    ports:
      - "5000:5000"

  backend:
    image: nvsailikhith/sentiment_analyzer_backend:latest
    container_name: sentiment_analysis_backend
    stdin_open: true
    ports:
      - "6000:6000"
```

Ansible: Ansible is a powerful automation and configuration management tool that streamlines the orchestration of IT infrastructure. It allows users to define the desired state of their systems, and then automates the processes needed to achieve that state. With its agentless architecture, Ansible communicates with remote servers using SSH, making it easy to set up and manage. Ansible playbooks, written in YAML, describe the tasks to be executed on remote machines, enabling tasks such as software deployment, configuration management, and system provisioning to be carried out efficiently and consistently. Ansible promotes infrastructure as code, offering a scalable and flexible solution for managing and automating IT infrastructure across diverse environments.

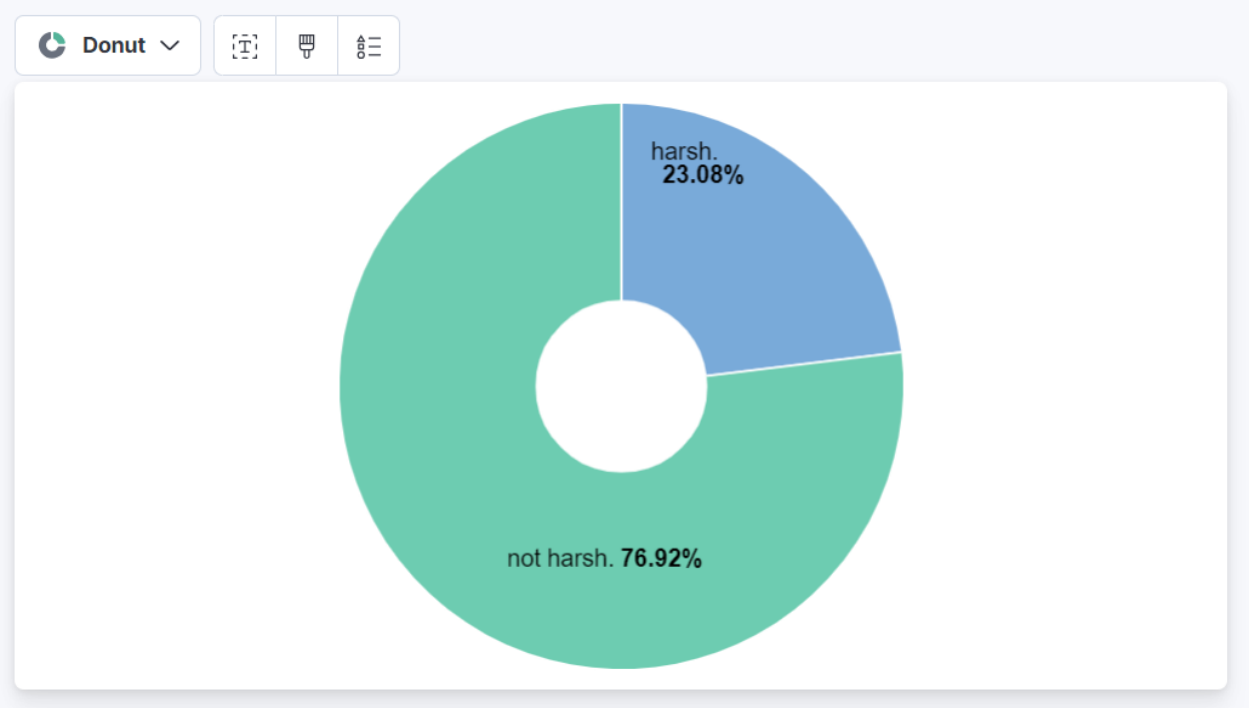
ngrok: ngrok is a tool that helps in hosting our localhost ports in global scope. We use it here to make our jenkins (port 8080) link up with github using webhooks. This helps in automatically triggering the Jenkins to build the application whenever there is a commit to GitHub (SCM tool).

ELK Stack: The Elastic Stack is widely adopted for log and event data analysis, application performance monitoring, and security information and event management (SIEM).

Bar graph obtained:



Pie Chart Obtained:



Jenkins Pipeline

Stage I: Git SCM pull

```
stage('Stage 1: Git pull') {
    steps {
        git url: 'https://github.com/Likhith-2914/Sentiment-Analysis.git', branch: 'master'
    }
}
```

In stage 1, we specify the github repository to pull from, and the branch.

Stage II: Build

```
stage('Stage 2: Build') {
    steps {
        sh 'pip3 install --upgrade pip'
        sh 'pip install -r requirements.txt'
        sh 'pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu'
    }
}
```

We install a few packages that are required in the testing phase

Stage III: Test

```
stage('Stage 3: Test') {
    steps {

        sh '''cd tester
            python3 tester.py'''

    }
}
```

Testing of application -> Run tester.py in the tester folder. The tester.py creates sub-processes to test two key features in the application - extracting the twitter link and predicting the correct output. The testing is done by the 'unittest' module of the python.

Stage IV: Docker image - Frontend

```
stage('Stage 4: Docker image - frontend') {

    steps {

        script {
            frontend_docker_image = docker.build("nvsailikhith/sentiment_analysis_frontend:latest", "./frontend")
        }
    }
}
```

Build docker image for frontend

Stage V: Docker image - Backend

```
stage('Stage 5: Docker image - backend') {
    steps {
        script {
            backend_docker_image = docker.build("nvsailikhith/sentiment_analysis_backend:latest", "./backend")
        }
    }
}
```

Build docker image for backend

Stage VI: Push frontend docker image

```
stage('Stage 6: Push frontend docker image') {

    steps {

        script {
            docker.withRegistry('', 'DockerHubCred') {
                frontend_docker_image.push()
            }
        }
    }
}
```

Pushing the frontend docker image to docker hub. DockerHubCred is the ID that contains Docker Hub's username and password.

Stage VII: Push backend docker image

```
stage('Stage 7: Push backend docker image') {

    steps {

        script {
            docker.withRegistry('', 'DockerHubCred') {
                backend_docker_image.push()
            }
        }
    }
}
```

Similarly, we push the backend docker image to docker hub.

Stage VIII: Clear cache

```
stage('Stage 8: Clear Cache') {
    steps {

        sh 'docker container prune -f'
        sh 'docker image prune -f'
    }
}
```

Clears the dangling images, i.e., docker images that don't have a tag. This arises as we are pushing new images with 'latest' tag in every run, which takes away the 'latest' tag from the images of the preceding run.

Stage IX: Ansible Deploy

```
stage('Stage 9: Ansible Deploy') {
    steps {
        ansiblePlaybook becomeUser: null,
        colorized: true,
        credentialsId: 'localhost',
        disableHostKeyChecking: true,
        installation: 'Ansible',
        inventory: 'inventory',
        playbook: 'ansible-playbook.yml',
        sudoUser: null

        input message: 'Finished using the web site? (Click "Proceed" to continue)'
        sh 'docker-compose down' //Stopping and removing the containers and networks linked to the project
    }
}
```

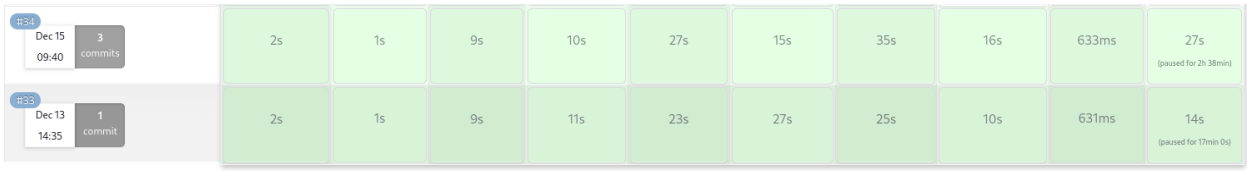
In this stage, we run the ansible-playbook.

```
---
- name: Deploy Docker Images
  hosts: all

  vars:
    ansible_python_interpreter: /usr/bin/python3

  tasks:

    - name: Start docker service
      service:
        name: docker
        state: started

    - name: Pull the docker images
      command: docker-compose pull

    - name: Run the docker images (detached mode)
      command: docker-compose up -d

```

It mainly performs two tasks,

1) Pull the docker images: In this task, the command `docker-compose pull` command pulls the docker images specified in docker-compose.yaml.

2) Run the containers: `docker-compose up -d` command runs the containers that have the images that were pulled.



Above image shows the complete working of the pipeline. In the demo, we have shown the pipeline starts to execute after a commit automatically.