TRUSTWISE INDIVIDUAL ASSINGMENT

Submitted by – Devansh(2020B5A72001P)

Submitted on – 06/02/25

Project description

This project implements a REST API(using FAST API) architecture that can handle a string input and return the response determined by two open source models. The response produced by these models are saved in a database and displayed in a web UI. The web UI can display scores in form of a graph and also a table showing the history of received text and results.

Objectives achieved by project

- Implemented a user friendly web UI which allows users to input a text string for evaluation
- The project uses two open source hugging face models - “vectara/hallucination_evaluation_model” and “snlp/roberta_toxicity_classifier”. These models evaluate a string based on hallucination and toxicity respectively
- The response produced by these models is stored in a postgresDB.
- The frontend framework used here is Streamlit and source code is written in python language
- The API calls to server are done in a manner which conforms to the design principles of the representational state transfer (REST) architectural style, implemented using FAST API

Different components of project

The code is briefly divided into 4 files

1. Streamlit.py file – This file contains the code for Streamlit framework which is used to create this project frontend
2. model.py file - This file contains the function which call the required models from hugging face via transformers library
3. api.py file – This file contains the API calls for function, follows RESTful architecture
4. db.py file – contains code for accessing data from postgresDB
5. Docker – I was unable to implement this part of project

Challenges faced during project implementation

- There was a bit of confusion regarding the type of input user would do for the vectara model. The example given in the assignment sheet was “I like you. I love you” which does not have any premise or hypothesis. Whereas the model page on hugging face says it is “designed for detecting hallucinations in LLM”, the code explanation also tells it needs a premise and hypothesis for evaluation.

- On further testing I found the model would run just fine on single string, but to keep the purpose of model, I made the assumption that punctuation mark(.) is used as a splitter between premise and hypothesis. So my function for calling - “vectara/hallucination_evaluation_model” splits the string at punctuation mark(.) and divides them into premise and hypothesis. Edge cases are also considered for same
- There were some challenges faced during certain frontend features, biggest being creating a interactive scatter plot graph for data stored in database. For this I decided to combine the features of plotly, pandas and Streamlit which made it possible to show an cursor interactive graph.

Instruction for running code

- I have uploaded the working code with and without docker on git repository. I have also shared a google drive link for same, which contains the video explaining all the code
- First create virtual environment using
    - python -m venv venv
    - venv/Scripts/activate
    - In one terminal run “uvicorn api:app –reload”
    - In second terminal run “streamlit run streamlit.py”
- For deploying docker simply run virtual environment, keep all files of docker in single folder and run
    - venv/Scripts/activate
    - docker-compose up -d --build

Glossary of code

The model.py file

This file is responsible for calling the models Vectara and Toxicty using Huggingface library of transformers. It contains 3 functions

- def vectara(str):
- def toxic(str):
- def call_llm(string):

I will explain the working of these function one-by-one

def call_llm(string):

This function receives the string which need to be evaluated by models via API call. The string is pushed to function vectara() and toxic(str), output of both these functions are stored in 2 variables and inserted as a query to postgresDB, furthermore the output variables are converted to json format and returned to API call.

```
def call_llm(string):
    toxic_out = toxic(string)
    vectara_out = vectara(string)
    if vectara_out is not None:
        data = {                                #converts the output to json format so it can be easily handeled by API and frontend
            "Received text:": string,
            "Vectara": vectara_out[0],
            "Toxicity": toxic_out
            }
        insert_data(string,toxic_out,vectara_out) #insert data into DB
        return data
    else:
        return {"error": "Invalid input format. please enter in following format 'sentence1,sentence2' where both sentence are separated
```

def vectara(string):

This function receives the string and then breaks it into 2 parts from the first period(.), if period does not exist then None is returned to call_llm, else if it contains the period, the string is split into 2 parts pre and hypo which means premises and hypothesis respectively. Both these values are then sent to model via Huggingface pipeline as a prompt.

The output is then stored and returned to call_llm

```python
def vectara(str):
    try:
        pre, hypo = str.split(".",1)#splits the string based on first encountered "."
    except ValueError:
        return None  #error handling if there are no "," present in provided string

    #creating prompt for model
    prompt = "<pad> Determine if the hypothesis is true given the premise?\n\nPremise: {text1}\n\nHypothesis: {text2}"
    input_pairs = [prompt.format(text1=pre, text2=hypo)]

    full_scores = classifier(input_pairs, top_k=None) # List[List[Dict[str, float]]]

    # Optional: Extract the scores for the 'consistent' label
    simple_scores = [score_dict['score'] for score_for_both_labels in full_scores for score_dict in score_for_both_labels if score_dict

    return simple_scores
```

def toxic(str):

This function also receives the string but no splitting happens here, first the model is called then a tokenizer is created, after this the string is sent to model and response of model is saved. The response is further simplified and returned to call_llm().

```python
def toxic(str):
    model = RobertaForSequenceClassification.from_pretrained('s-nlp/roberta_toxicity_classifier') #calls model from huggning face
    tokenizer = RobertaTokenizer.from_pretrained('s-nlp/roberta_toxicity_classifier') #takes an input and coverts to embedding vector
    batch = tokenizer.encode(str, return_tensors="pt") #how to send an array of string?

    with torch.no_grad():           #disables gradient calculation since we are not training model
        logits = model(batch).logits

    predicted_class_id = logits.argmax().item() #this returns the class with higher value, # idx 0 for neutral, idx 1 for toxic
    return predicted_class_id
```

The api.py file

This file simple contains the code for calling API request following REST API architecture. There are only 2 calls present, a post call to send string as input to model and receive their output.

```python
#post call for string analysis
@app.post("/call_model/{string}")
def call_model(string:str):
    return call_llm(string)
```

And another get call to receive data for creating graph or table, since both these return same type of query, the plotting of graph and displaying of table is handled by Streamlit.

```python
#testing if server is running
@app.get("/")
def root():
    return "Server is running"

#get call for table or graph creation
@app.get("/table_graph")
def table():
    return create_table_graph()
```

The db.py file

This file contains code for saving output of models as well as returning data for plotting graph and tables

```python
#postgres login data
conn = psycopg2.connect(
    host=os.getenv("HOST"),
    port=os.getenv("PORT"),
    user=os.getenv("DB_USER"),
    password=os.getenv("DB_PASSWORD"),
    database=os.getenv("DB_NAME")
)


def create_table():
    create_query = """CREATE TABLE IF NOT EXISTS output (
    id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    string TEXT,
    vectara REAL,
    toxicity REAL
    );"""
    cursor = conn.cursor()
    cursor.execute(create_query)
    conn.commit()
    cursor.close()
```

This snippet contains the code for connecting to postgesdb server, the login data is saved in login.env file, whose values are retrieved using os. This create table query only runs when there are no existing table in dataset.

Next two function just have simple codes of running query to postgresDB using psycopg2 object, either for entering or retrieving query data from database.

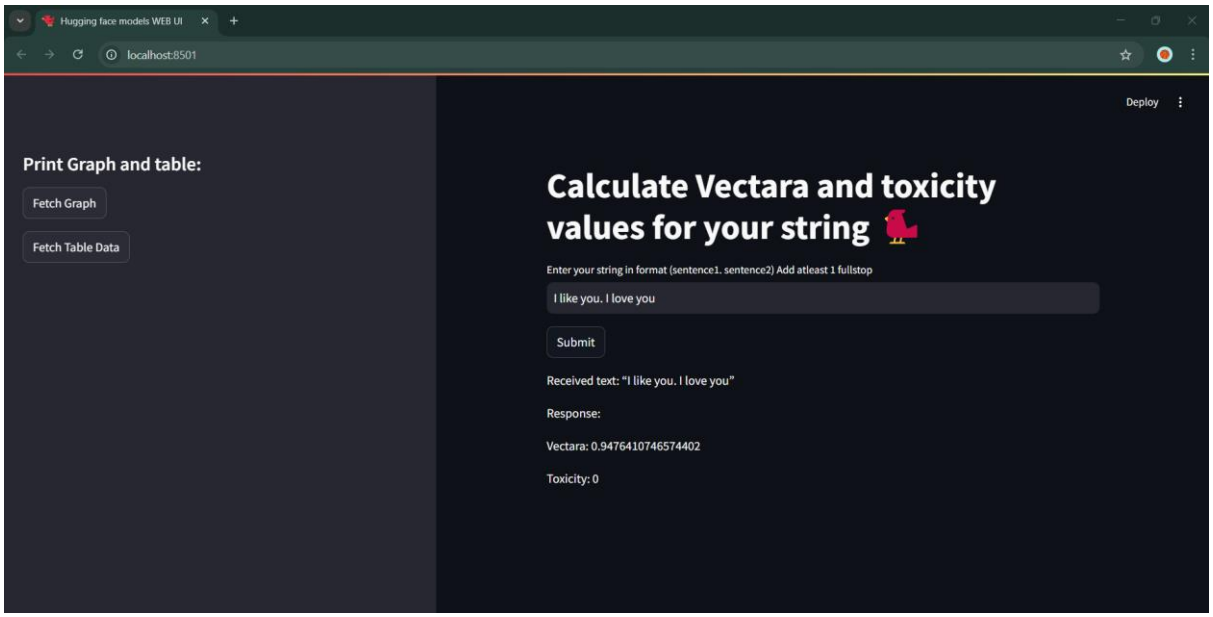
```python
#DB input action
def insert_data(string, toxic_out, vectara_out):
    create_table()
    try:
        cursor = conn.cursor()
        cursor.execute("INSERT INTO output(string, vectara, toxicity) VALUES (%s, %s, %s)", (string, vectara_out[0], toxic_out))
        conn.commit()
        cursor.close()
    except psycopg2.Error as err:
        return {"error" : f"Error occured while executing query : {err}"}

#DB table or graph call
def create_table_graph():
    create_table()
    try:
        df = pd.read_sql("SELECT string, vectara, toxicity FROM output ", conn)
        json_data = df.to_dict(orient='records')
        return json_data
    except psycopg2.Error as err:
        return {"error" : f"Error occured while executing query : {err}"}
```

The streamlit.py file

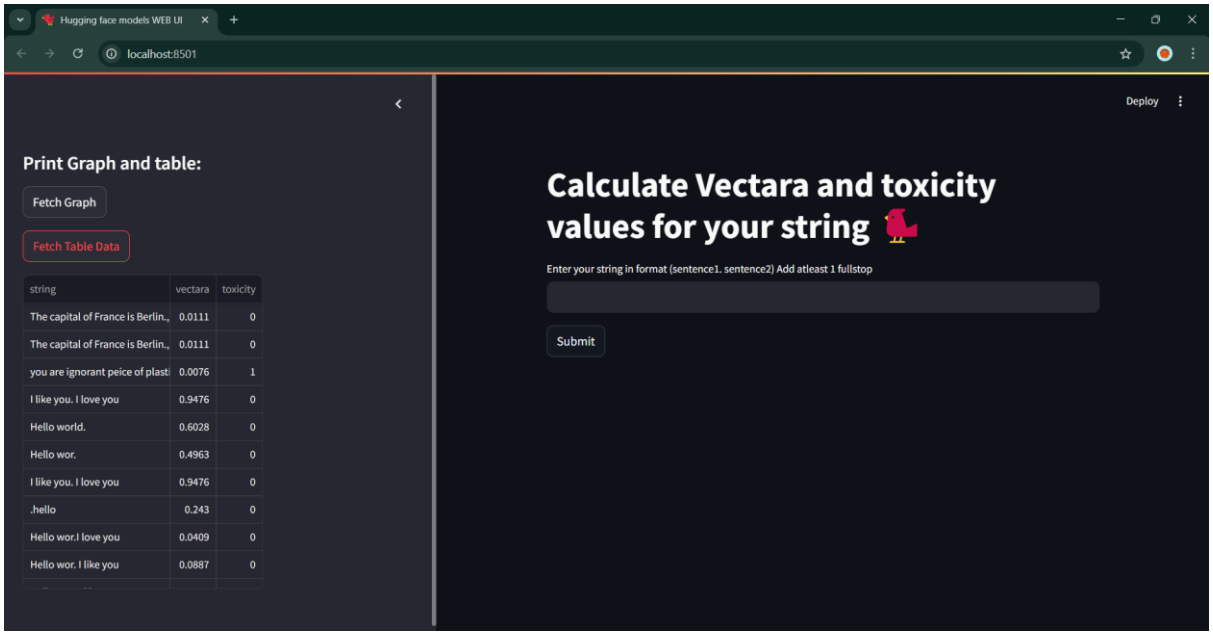
This is the frontend part of code, which is implemented using streamlit

This have total 3 options,

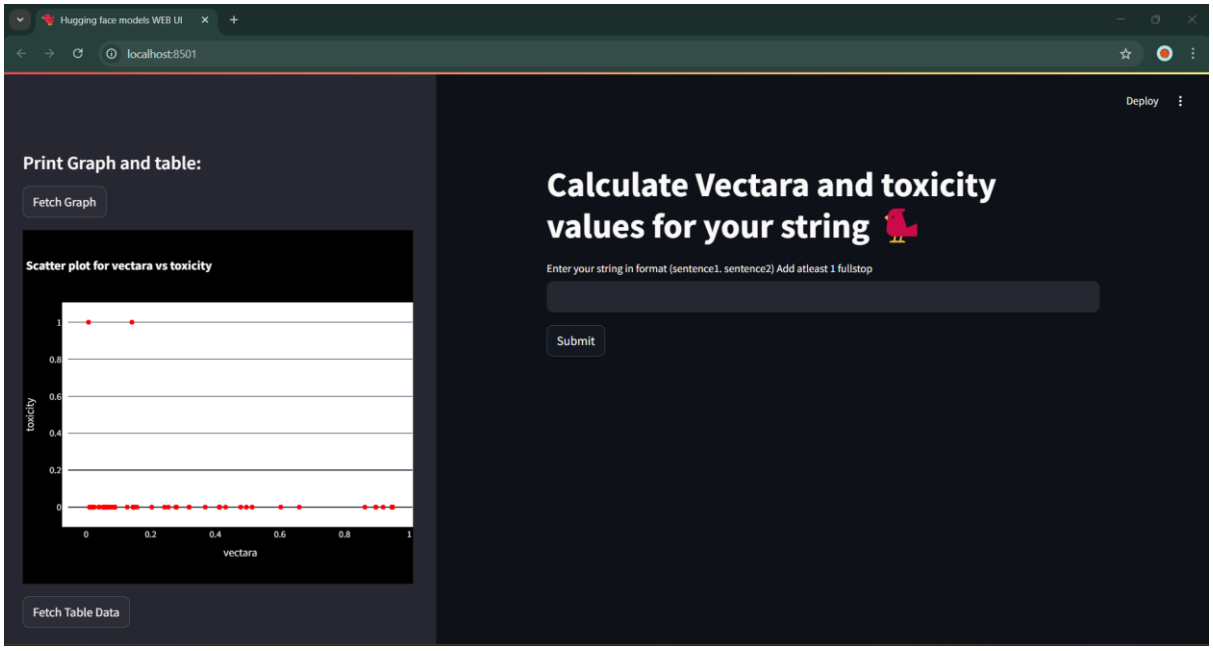
- submitting string for evalution
- Fetching Table of older input from database
- Plotting scatter graph using old database



The string is sent on “Submit” press to API, which further transfers it to call_llm(str) function from which the 2 function for vectara and toxicity evaluation are called. Then the response of both function are returned to both the api.py and db.py for printing on WEB UI and entering in database respectively. The models are correctly able to identify toxic strings as well as hallucination between strings as we can see from fetching table



The graph have all points either at 1 or 0 because toxicity have only integer value, with 0 being neutral and 1 being toxic, points on graph are interactive showing the vectara and toxicity points in (X,Y) format



Now diving into code, main() function is the one which handles everything in frontend, starting from first section which is the middle screen, here the user can input string of any

kind and will get output, all kind of edge cases are covered. Calls are made to API which return the output in json format and displayed on UI.

```
def main():
    #title and index
    st.set_page_config(page_title="Hugging face models WEB UI", page_icon="🐓", layout="centered", initial_sidebar_state="auto", menu_it
    st.title("Calculate Vectara and toxicity values for your string 🐓")


    sent_string=st.text_input("Enter your string in format (sentence1. sentence2) Add atleast 1 fullstop")

    #handling string input
    if st.button("Submit"):
        if sent_string:
            try:
                response = requests.post(url = f"http://127.0.0.1:8000/call_model/{sent_string}")
                json_data = response.json()
                st.write(f'Received text: “{json_data["Received text:"]}”')
                st.write("Response:")
                st.write(f'Vectara: {json_data["Vectara"]}')
                st.write(f'Toxicity: {json_data["Toxicity"]}')
```

The second part is sidebar where 2 buttons are given which are self explanatory, both these graphical objects are created using python library only. The line to line explanation is done briefly in the video which is provided in google drive. I request you to please see that video, which explains everything in-depth.

References

- **Google drive containing all my codes and video explaining in detail how everything is called and works** – https://drive.google.com/drive/folders/1wBSMk4Q5VNi9Vux5aulCio1_pAzym1__?usp=sharing

- s-nlp/roberta_toxicity_classifier · Hugging Face
- vectara/hallucination_evaluation_model · Hugging Face
- https://huggingface.co/docs/transformers/main/en/model_doc/roberta#transformers.RobertaForSequenceClassification