Divyansh Agrawal (2115500052) – 3R

IBM Assignment

Problem Statement:

Social media platforms have become battlegrounds for information, where misinformation and harmful content proliferate, eroding trust and sometimes leading to real-world harm. In response to this pressing challenge, there is a need for an AI-powered solution, referred to as the "Truth Detector," to combat misinformation and promote a healthier online space.

The mission is to design a neural network-based Truth Detector capable of:

- Spotting Fakes: Analyzing text and various forms of media (such as images and videos) to identify potential misinformation. This includes detecting common tactics and formats used by creators to spread false information.
- Critical Thinking: Going beyond keyword analysis, the Truth Detector must delve deeper into the context, sentiment, and credibility of information sources to avoid mistakenly flagging legitimate content as false.
- Fairness and Unbias: It is imperative to mitigate biases in both data and algorithms to ensure that the Truth Detector operates impartially. The goal is to uphold freedom of expression while safeguarding users from harm caused by misinformation.

Additionally, the design should incorporate mechanisms for users to provide feedback and appeal flagged content. This fosters transparency and trust in the system, empowering users to participate in the moderation process and contribute to a more reliable online environment.

The ultimate aim of the Truth Detector is to serve as a champion in the fight against misinformation on social media, promoting trust, credibility, and safety in online interactions. Are you ready to unleash the power of AI and build a healthier digital space?

Approach:

(Text -based Fact-Checker)

The code begins by importing necessary libraries such as NLTK, requests, numpy, and sklearn, which are essential for various functionalities. Next, it loads pre-trained GloVe word embeddings from a file (glove.6B.50d.txt). These embeddings capture semantic meanings of words in a high-dimensional space. Subsequently, two preprocessing functions are defined to prepare input text for analysis. The preprocess(sentence) function tokenizes input sentences, converts words to lowercase, and removes stopwords. Meanwhile, the get_sentence_embedding(sentence) function calculates the embedding for a given sentence using GloVe word embeddings, computing the mean of word embeddings for words in the sentence. Following this, the code defines a function, semantic_similarity(sentence1, sentence2), to measure the semantic similarity between two sentences using cosine similarity. It leverages the cosine_similarity function from sklearn to accomplish this task.

Moving on, the code utilizes the Google Generative AI (Gemini) through the ChatGoogleGenerativeAI class from the langchain_google_genai module. It initializes an instance of the class (llm) with the desired model and Google API key, enabling interactions with Gemini for generating responses to queries. Synonyms for "false" and "true" are retrieved from WordNet and stored in lists (synonyms and synonymsT), including additional terms such as speaker pronouns and affirmative terms in synonymsT. Subsequently, two functions for fact-checking are defined: fact_checker(q) and Combine_Fact_Checker(q).

The former checks the validity of a statement using synonyms of "false" and "true", and if necessary, invokes Gemini AI for clarification. The latter combines multiple fact-checking methods, including an external fact-checking API and the fact_checker() function, to determine the truthfulness of the provided information. The code queries the Google Fact Check Tools API to fetch fact-checking information based on user queries, handling cases where the API response is insufficient by falling back to internal fact-checking methods. Lastly, exception handling is implemented to catch errors that may occur during API calls or processing of input statements, ensuring robustness and graceful handling of errors throughout the code execution.

Declarations:

(Text -based Fact-Checker)

1. Libraries Imported:

- nltk: Natural Language Toolkit library for text processing.
- requests: Library for making HTTP requests.
- numpy: Library for numerical computations.
- sklearn.metrics.pairwise: Module for computing pairwise similarity scores.
- langchain_google_genai: Custom module for interacting with the Google Generative AI.

2. Function Definitions:

a. preprocess(sentence)

- Description: Preprocesses the input sentence by tokenizing, converting to lowercase, removing stopwords, and joining the remaining words.
- Input: sentence (string) - The input sentence to be preprocessed.
- Output: Preprocessed string.

b. get_sentence_embedding(sentence)

- Description: Computes the sentence embedding using pre-trained GloVe word embeddings.

    - Input: sentence (string) - The input sentence.
    - Output: Sentence embedding as a numpy array.

    c. semantic_similarity(sentence1, sentence2)

    - Description: Calculates the cosine similarity between the embeddings of two sentences.
    - Inputs: sentence1, sentence2 (strings) - The two input sentences.
    - Output: Similarity score (float) between 0 and 1.

    d. gemini(q)

    - Description: Utilizes the Google Generative AI to generate responses for the given query.
    - Input: q (string) - The query.
    - Output: Generated response as a string.

    e. fact_checker(q)

    - Description: Checks the validity of a statement by using synonyms of "false" and "true", and if needed, invokes the Gemini AI for clarification.
    - Input: q (string) - The statement or question to be fact-checked.
    - Output: Prints whether the information is true or false.

    f. Combine_Fact_Checker(q)

    - Description: Combines multiple fact-checking methods including an external fact-checking API and the fact_checker() function.
    - Input: q (string) - The statement or question to be fact-checked.
    - Output: Prints whether the information is true or false.

3. Variables and Data:

    - word_embeddings: Dictionary containing pre-trained GloVe word embeddings.
    - stop_words: Set of English stopwords.
    - synonyms, synonymsT: Lists containing synonyms of "false" and "true" respectively.
    - speaker_pronouns: List containing speaker pronouns.
    - negative_marker: List containing markers for negative ratings.

4. External APIs:

    - Google Fact Check Tools API: Used for fetching fact-checking information based on user queries.

5. Main Functionality:

    - The Combine_Fact_Checker() function serves as the main interface for fact-checking statements or questions. It combines multiple fact-checking methods to determine the validity of the provided information.
    - If the external Fact-Checker API doesn't return a satisfactory response, it falls back to the fact_checker() function which leverages synonyms and Gemini AI to assess the truthfulness of the information.

6. Error Handling:

- Exception handling is implemented to capture errors that may occur during API calls or processing of input statements.

(DeepFake Detection)

gradio: Gradio is a library that allows you to quickly create UIs for your machine learning models. It's particularly useful for building web-based interfaces.

torch: PyTorch is an open-source machine learning library used for tasks such as natural language processing and computer vision.

facenet_pytorch: This library provides pre-trained face detection models based on deep learning architectures. MTCNN (Multi-task Cascaded Convolutional Networks) is a face detection algorithm.

numpy: NumPy is a library for numerical computing with Python, providing support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

PIL (Python Imaging Library): PIL is a library for opening, manipulating, and saving many different image file formats.

cv2 (OpenCV): OpenCV is a library of programming functions mainly aimed at real-time computer vision. It provides tools for image processing and computer vision.

pytorch_grad_cam: PyTorch-Grad-CAM is a library for visualizing the regions of an image that a CNN focuses on while making a decision.

Model Used:

The code utilizes the InceptionResnetV1 model pre-trained on the VGGFace2 dataset for face recognition tasks.

Functions Defined:

predict(input_image:Image.Image): This function takes an input image, detects faces using MTCNN, preprocesses the detected face, generates class activation maps (CAM) using GradCAM, performs inference using the pre-trained InceptionResnetV1 model, and returns the predicted class ("real" or "fake") along with the confidence scores and the face with explainability (highlighted regions indicating decision-making areas).

User Interface (Gradio):

The gr.Interface class is used to create a simple web-based UI for the predict function. It takes the function (predict), defines the input and output components of the UI (input image, predicted class label, and the image with explainability), and launches the interface.

Launching the Interface:

The launch() method is called on the interface object, which launches the Gradio interface in a new browser window.

Step-by-Step Explaination:

(Text -based Fact-Checker)

Importing Libraries:

- The code begins by importing necessary libraries such as NLTK, requests, numpy, and sklearn. These libraries provide functionalities for natural language processing, HTTP requests, numerical computations, and similarity calculations.

Loading Word Embeddings:

- The code loads pre-trained GloVe word embeddings from a file (glove.6B.50d.txt). These embeddings represent words as dense vectors in a high-dimensional space, capturing semantic meanings.

Preprocessing Functions:

Two functions are defined for preprocessing text:

- preprocess(sentence): Tokenizes the input sentence, converts words to lowercase, removes stopwords, and returns the processed string.
- get_sentence_embedding(sentence): Computes the embedding for a given sentence using GloVe word embeddings. It calculates the mean of word embeddings for words in the sentence.

Semantic Similarity Calculation:

- The semantic_similarity(sentence1, sentence2) function calculates the cosine similarity between embeddings of two input sentences. It utilizes the cosine_similarity function from sklearn.

Utilizing Google Generative AI (Gemini):

- The code uses the ChatGoogleGenerativeAI class from the langchain_google_genai module to interact with the Google Generative AI (Gemini). It initializes an instance of the class (llm) with the desired model and Google API key.

Synonym Retrieval:

- Synonyms for "false" and "true" are obtained using WordNet and stored in lists (synonyms and synonymsT respectively). Additionally, speaker pronouns and affirmative terms are included in the synonymsT list.

Fact Checking Functions:

Two functions are defined for fact-checking:

- fact_checker(q): Checks the validity of a statement using synonyms of "false" and "true", and if necessary, invokes Gemini AI for clarification.
- Combine_Fact_Checker(q): Combines multiple fact-checking methods including an external fact-checking API and the fact_checker() function. It checks the truthfulness of the provided information.

External Fact-Checking API:

- The code queries the Google Fact Check Tools API to fetch fact-checking information based on user queries. It handles cases where the API response is insufficient by falling back to internal fact-checking methods.

Main Functionality Execution:

- The Combine_Fact_Checker() function serves as the main interface for fact-checking. It takes a statement or question as input, combines various fact-checking methods, and prints whether the information is true or false.

Error Handling:

- Exception handling is implemented to catch errors that may occur during API calls or processing of input statements. This ensures graceful handling of errors and prevents program crashes.

Code:

(Text -based Fact-Checker)

```
import nltk
from nltk.corpus import wordnet, stopwords
import requests
from nltk.tokenize import word_tokenize
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from langchain_google_genai import ChatGoogleGenerativeAI

word_embeddings = {}
with open('glove.6B.50d.txt', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        embedding = np.array(values[1:], dtype='float32')
        word_embeddings[word] = embedding

stop_words = set(stopwords.words('english'))

def preprocess(sentence):
    tokens = nltk.word_tokenize(sentence)
    words = [word.lower() for word in tokens if word.isalpha() and word.lower() not in stop_words]
    words = ' '.join(words)
    return words

def get_sentence_embedding(sentence):
    words = preprocess(sentence)
    word_vectors = [word_embeddings[word] for word in words if word in word_embeddings]
    if len(word_vectors) > 0:
        sentence_embedding = np.mean(word_vectors, axis=0)

        return sentence_embedding

    else:

        return None


def semantic_similarity(sentence1, sentence2):

    embedding1 = get_sentence_embedding(sentence1)

    embedding2 = get_sentence_embedding(sentence2)

    if embedding1 is not None and embedding2 is not None:

        similarity_score = cosine_similarity([embedding1], [embedding2])[0][0]

        return similarity_score

    else:

        return 0


llm = ChatGoogleGenerativeAI(model="gemini-pro", google_api_key="AIzaSyDqwCJlVvtYjNsq-
59W5FA-H4tQO7gBxmc")


def gemini(q):

    response = llm.invoke([q])

    return response.content


synonyms = []


for syn in wordnet.synsets("false"):

    for i in syn.lemmas():

        synonyms.append(i.name())


synonyms = list(set(synonyms))

synonyms += ['not', 'not true',  'lies', 'no', 'not correct', 'incorrect']

print(synonyms)
```

```
synonymsT = []

for syn in wordnet.synsets("true"):
    for i in syn.lemmas():
        synonymsT.append(i.name())

synonymsT = list(set(synonymsT))
speaker_pronouns = ['i', 'me', 'myself', 'we', 'us', 'ourselves', "i'm", "i've", "i'd", "i'll", "my", "mine"]
synonymsT += speaker_pronouns
synonymsT.append("yes")
print(synonymsT)

def fact_checker(q):
    try:
        q_token = word_tokenize(q)
        q_token = [word for word in q_token]
        if any(token in synonymsT for token in q_token):
            print("True Information")
        else:
            response = gemini(q + ", simple yes or no and explain")
            print("Gemini Response:", response)
            q_token = [word for word in response.lower().replace('\n', ' ').replace(',', '').split(" ")]
            if any(token in synonyms for token in q_token):
                print("False Information")
            elif any(token in synonymsT for token in q_token):
                print("True Information - 2")
            else:
                print("False Information - 2")

    except Exception as e:
        print("Gemini couldn't resolve your request:", e)
```

```
negative_marker = ['Pants on Fire', 'False', 'Incorrect Information', 'Incorrect', 'Fake']


def Combine_Fact_Checker(q):
    if (len(q) == 0):
        print("No query provided")
        return
    try:
        query = "%20".join(q.lower().split(" "))
        resp =
requests.get(f"https://factchecktools.googleapis.com/v1alpha1/claims:search?pageSize=3&query={q
uery}&key=AIzaSyAfbNl74qJ7S1iMVQwtrxqFNu-SB6DOPwc").json()
        if (len(resp) == 0 or len(resp['claims']) != 3):
            fact_checker(q.lower())
        else :
            textual_ratings = []
            for claim in resp['claims']:
                for review in claim['claimReview']:
                    textual_ratings.append(review['textualRating'])
            if (set(textual_ratings).issubset(set(negative_marker))):
                print("False Information")
            else:
                print("True Information")
    except Exception as e:
        print("Fact-Checker API Error:", e)


Combine_Fact_Checker("Is Joe Biden dead?")


import gradio as gr
import torch
import torch.nn.functional as F
from facenet_pytorch import MTCNN, InceptionResnetV1
```

```
import numpy as np
from PIL import Image
import cv2
from pytorch_grad_cam import GradCAM
from pytorch_grad_cam.utils.model_targets import ClassifierOutputTarget
from pytorch_grad_cam.utils.image import show_cam_on_image
import warnings
warnings.filterwarnings("ignore")

DEVICE = 'cpu'

mtcnn = MTCNN(
   select_largest=False,
   post_process=False,
   device=DEVICE
).to(DEVICE).eval()

model = InceptionResnetV1(
   pretrained="vggface2",
   classify=True,
   num_classes=1,
   device=DEVICE
)

checkpoint = torch.load("resnetinceptionv1_epoch_32.pth", map_location=torch.device('cpu'))
model.load_state_dict(checkpoint['model_state_dict'])
model.to(DEVICE)
model.eval()

def predict(input_image:Image.Image):
   try:
```

```
"""Predict the label of the input_image"""
face = mtcnn(input_image)
if face is None:
    raise Exception('No face detected')
face = face.unsqueeze(0) # add the batch dimension
face = F.interpolate(face, size=(256, 256), mode='bilinear', align_corners=False)

# convert the face into a numpy array to be able to plot it
prev_face = face.squeeze(0).permute(1, 2, 0).cpu().detach().int().numpy()
prev_face = prev_face.astype('uint8')

face = face.to(DEVICE)
face = face.to(torch.float32)
face = face / 255.0
face_image_to_plot = face.squeeze(0).permute(1, 2, 0).cpu().detach().int().numpy()

target_layers=[model.block8.branch1[-1]]
cam = GradCAM(model=model, target_layers=target_layers)
targets = [ClassifierOutputTarget(0)]

grayscale_cam = cam(input_tensor=face, targets=targets, eigen_smooth=True)
grayscale_cam = grayscale_cam[0, :]
visualization = show_cam_on_image(face_image_to_plot, grayscale_cam, use_rgb=True)
face_with_mask = cv2.addWeighted(prev_face, 1, visualization, 0.5, 0)

with torch.no_grad():
    output = torch.sigmoid(model(face).squeeze(0))
    prediction = "real" if output.item() < 0.5 else "fake"

    real_prediction = 1 - output.item()
    fake_prediction = output.item()
```

```
            confidences = {
                'real': real_prediction,
                'fake': fake_prediction
            }
    except Exception as e:
        print(e)
    return confidences, face_with_mask


interface = gr.Interface(
    fn=predict,
    inputs=[
        gr.Image(label="Input Image", type="pil")
    ],
    outputs=[
        gr.Label(label="Class"),
        gr.Image(label="Face with Explainability", type="pil")
    ],
).launch()
```