TABLE OF CONTENTS

ABSTRACT

This project aimed to address the issue of toxic comments on Twitter by classifying them into various categories of toxicity using Python, PyTorch, and BERT. The data was collected from Twitter using Selenium to scrape the data which was then pre-processed and analysed. The data used to train the BERT model was taken from the Kaggle Competition for the Toxic Comment Classification Challenge, the model then was implemented through the PyTorch framework. The resulting AUC score of 98.86 demonstrated the effectiveness of BERT and PyTorch in accurately classifying toxic comments on social media platforms. By efficiently identifying and removing toxic content, this project contributes to creating a safer online environment and promoting healthier online communities. Python facilitated the seamless integration of various libraries and frameworks, while PyTorch provided the tools for building and training neural networks. BERT, a pre-trained transformer-based model for natural language processing, played a pivotal role in text classification. Overall, this project marks a significant advancement in tackling toxic content on the internet.

INTRODUCTION

In today's digital age, social media platforms have become powerful tools for communication and interaction. They offer vast opportunities for individuals to express their thoughts, share opinions, and connect with others from across the globe. However, this virtual realm is not without its challenges, one of which is the prevalence of toxic comments. Toxic comments, laden with offensive language, hate speech, and harmful content, can have detrimental effects on both individuals and online communities as a whole.

The problem of toxic comments on social media platforms is multifaceted. For individuals targeted by such content, the experience can be distressing, leading to emotional and psychological harm. Toxic comments can foster an environment of fear and intimidation, limiting free expression and inhibiting healthy discussions. Moreover, these harmful remarks create a negative atmosphere within online communities, affecting user engagement and deteriorating the overall quality of interactions.

Currently, addressing toxic content on social media platforms relies heavily on manual moderation, which can be both time-consuming and inefficient. The sheer volume of user-generated content makes it challenging for moderators to identify and remove all instances of toxicity. As a result, harmful comments often go unnoticed, perpetuating a toxic online environment.

To combat this pervasive issue, this project seeks to develop an automated approach to classify and identify toxic comments on Twitter. Leveraging the power of transformer-based machine learning techniques BERT, our goal is to accurately identify and remove toxic comments, thereby fostering a safer and more positive online environment for all users.

Python, a versatile programming language, provides a robust foundation for implementing machine learning algorithms. Paired with the PyTorch framework, known for its efficiency in training deep neural networks, we have a powerful toolkit at our disposal. From Hugging Face- BERT (Bidirectional Encoder Representations from Transformers), a pre-trained transformer-based model, is utilized for natural language processing tasks. Its ability to understand context and semantics makes it a valuable asset in deciphering the meaning and intent behind user comments.

The methodology involves data collection from Twitter through a custom scraper implemented using Selenium. By gathering relevant tweets under a topic, we create a comprehensive dataset for analysing the tweets for toxicity. Subsequently, the data is pre-processed to remove irrelevant information and prepare it for classification.

By training the BERT model with PyTorch, we equip it to classify toxic comments into various categories, including toxic, severe toxic, obscene, threat, insult, and identity hate. The effectiveness of our approach is measured using the AUC score, indicating the model's ability to accurately classify toxic comments.

The results of this project demonstrate the potential of automated approaches to tackle toxicity on social media platforms. Achieving a high AUC score of 98.86 for the 5 models utilized emphasizes the efficacy of BERT in detecting and addressing harmful content. Ultimately, this project makes a significant contribution to fostering a safer online space and promoting healthy online communities.

In conclusion, the prevalence of toxic comments on social media platforms demands innovative solutions to create a safer and more positive online environment. Leveraging the power of Python, PyTorch, and BERT, this project presents an automated approach to classify and identify toxic comments on Twitter. By accurately identifying and removing harmful content, we endeavour to cultivate a digital landscape that promotes healthy interactions, free expression, and a sense of safety for all users.

SYSTEM REQUIREMENTS

SOFTWARE REQUIREMENTS

The Python project requires you to have basic knowledge of Python programming, deep learning with the PyTorch library and Selenium web scraping.

Install the necessary libraries for this project using this command:
! pip install numpy, pandas, selenium, matplotlib, nltk, seaborn, transformers, pytorch

The Toxic Comment Classification Challenge dataset:

Is a collection with a large number of Wikipedia comments which have been manually labelled for toxic behaviour. The types of toxicity are toxic, severe_toxic, obscene, threat, insult, and identity_hate.

File descriptions:

train.csv - the training set, contains comments with their binary labels

test.csv - the test set, you must predict the toxicity probabilities for these comments. To deter hand labelling, the test set contains some comments which are not included in the scoring.

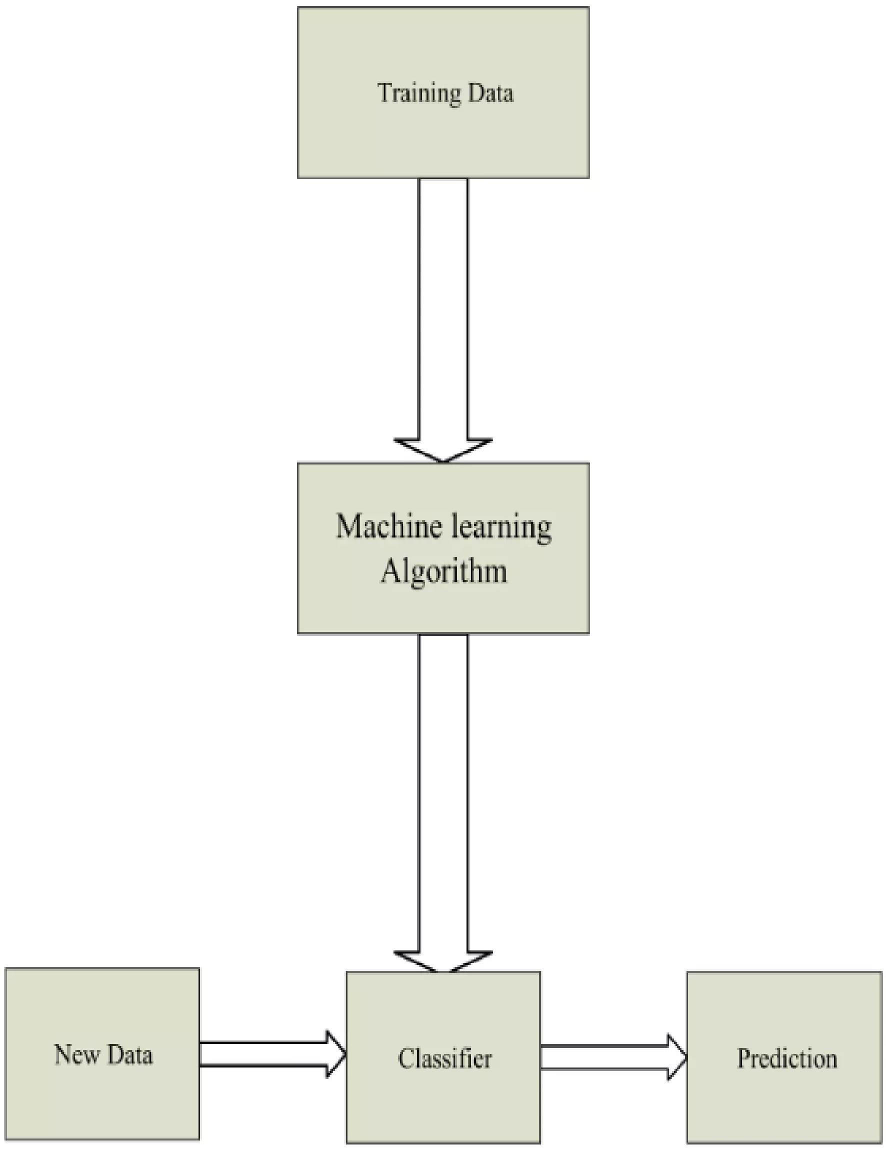
Usage

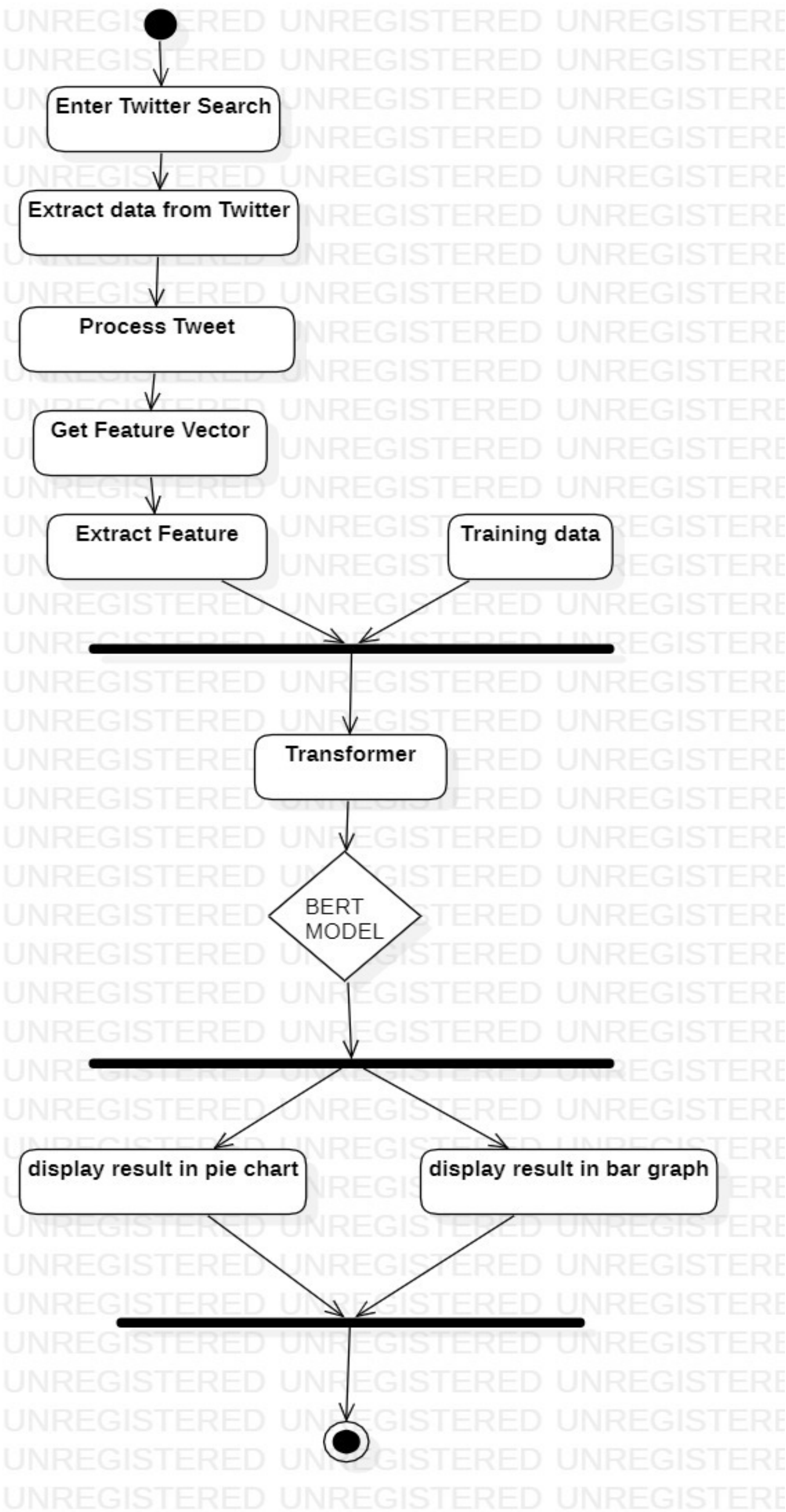
The dataset under CC0, with the underlying comment text being governed by Wikipedia's CC-SA-3.0

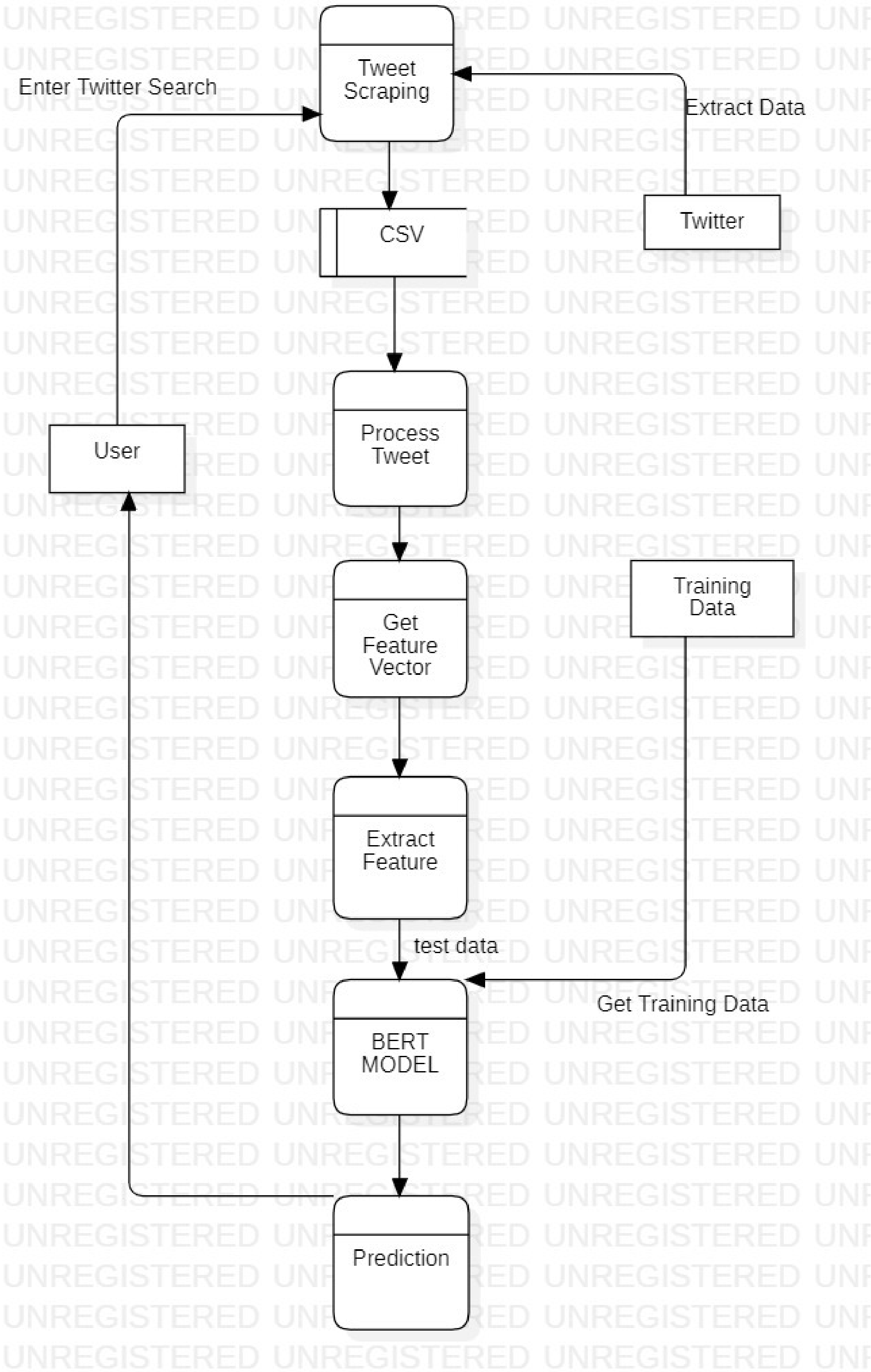
HARDWARE REQUIREMENTS

- ◊ RAM : 16 GB or Higher
- ◊ Processor : Intel i5 or above
- ◊ Accelerator : GPU T4 x2

SYSTEM DESIGN




Enter Twitter Search
Extract data from Twitter
Process Tweet
Get Feature Vector
Extract Feature
Training data
Transformer
BERT MODEL
display result in pie chart
display result in bar graph


Tweet Scraping
Enter Twitter Search
Extract Data
Twitter
CSV
User
Process Tweet
Training Data
Get Feature Vector
Extract Feature
test data
Get Training Data
BERT MODEL
Prediction

IMPLEMENTATION

PART I SCRAPPING

ALGORITHM

1. Set the Twitter login credentials (**get_username**, **get_pass**) and the search topic (**get_topic**).
2. Install the necessary libraries (**selenium**, **getpass**, **csv**) and import the required modules.
3. Create a Chrome WebDriver instance.
4. Navigate to the Twitter login page.
5. Maximize the window for better visibility.
6. Wait for a few seconds to allow the page to load.
7. Enter the username and press Enter.
8. Wait for a few seconds before entering the password.
9. Enter the password and press Enter.
10. Wait for a sufficient time to log in.
11. Check for any popup by Twitter.
12. Enter the search topic in the search input field and press Enter.
13. Click on the "Latest" option to filter the search results.
14. Define the function **get_tweet_data(card)** to extract tweet data from each tweet card.
15. Get all tweet cards on the page and loop through them.
16. For each tweet card, extract the handle, post date, comment, and responding text using the **get_tweet_data()** function.
17. Append the tweet data to the **data** list.

18. Scroll down the page to load more tweets.

19. Repeat steps 15-18 until all relevant tweets are collected.

20. Save the collected tweet data to a CSV file with the given filename.

PSUDOCODE

- SET get_username = 'USERNAME'
- SET get_pass = 'PASSWORD'
- SET get_topic = 'ENTER SEARCH TOPIC'
- INSTALL selenium, getpass, csv
- IMPORT necessary modules
- CREATE driver as Chrome WebDriver instance
- NAVIGATE to 'https://www.twitter.com/login'
- MAXIMIZE window
- WAIT for a few seconds (sleep) to allow the page to load
- FIND username input field
- ENTER get_username
- PRESS Enter
- WAIT for a few seconds (sleep) before entering the password
- FIND password input field
- ENTER get_pass
- PRESS Enter
- WAIT for sufficient time to log in (sleep)
- CHECK for any popup by Twitter
- FIND search input field
- ENTER get_topic
- PRESS Enter
- CLICK on the "Latest" option
- DEFINE function get_tweet_data(card):
- TRY:
- EXTRACT handle from card
- EXCEPT NoSuchElementException:
- RETURN
- TRY:
- EXTRACT postdate from card
- EXCEPT NoSuchElementException:
- RETURN
- EXTRACT comment from card
- EXTRACT responding from card
- CONCATENATE comment and responding to get the full text

- CREATE tweet as a tuple (handle, text)
- RETURN tweet
- CREATE an empty list data to store tweet data
- CREATE an empty set tweet_ids to store unique tweet ids
- SET scrolling = True
- SET last_position = window.pageYOffset
- WHILE scrolling:
- FIND all tweet cards on the page
- LOOP through each tweet card:
- CALL get_tweet_data(card) to get tweet data
- IF tweet data is not None:
    - CALCULATE tweet_id from the handle and text
    - IF tweet_id is not in tweet_ids:
    - ADD tweet_id to tweet_ids
    - APPEND tweet data to data
- SCROLL down the page to load more tweets
- WAIT for a few seconds (sleep)
- CALCULATE current_position = window.pageYOffset
- IF last_position is equal to current_position:
- INCREMENT scroll_attempt
- IF scroll_attempt is greater than or equal to 3:
    - SET scrolling = False
- ELSE:
    - WAIT for a few seconds (sleep) and attempt another scroll
- ELSE:
- UPDATE last_position to current_position
- WRITE data to a CSV file with the given filename
- CLOSE the driver

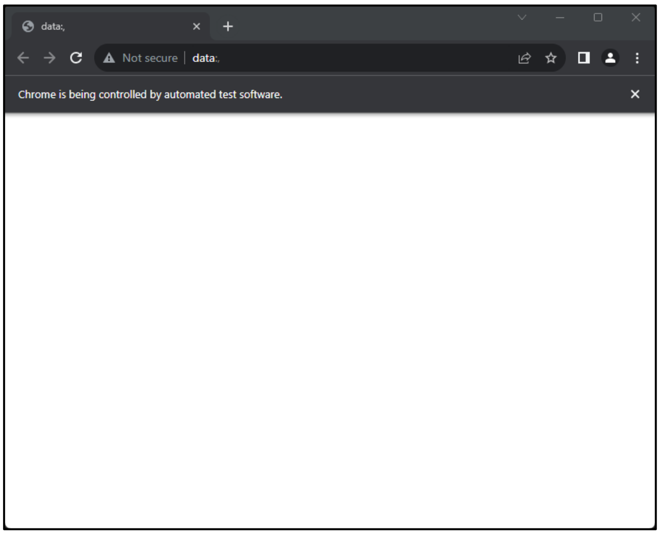
EXECUATABLE CODE

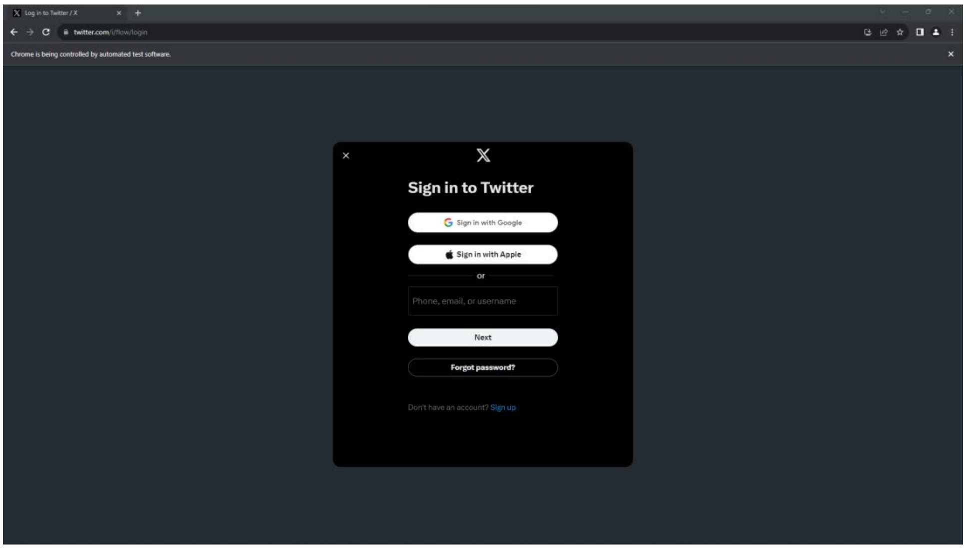
```
get_username = 'USERNAME'

get_pass = 'PASSWORD'

get_topic = 'ENTER SEARCH TOPIC'

!pip install selenium

from selenium.webdriver.common.keys import Keys

from selenium.common.exceptions import NoSuchElementException

from selenium.webdriver import Chrome

import re

from getpass import getpass

from time import sleep

import csv

driver = Chrome()

driver.get('https://www.twitter.com/login')

driver.maximize_window()

#change sleep values according to internet speed

sleep(5)

username = driver.find_element("xpath",'//input[@name="text"]')

username.send_keys(get_username)

username.send_keys(Keys.RETURN)

sleep(3)

password=driver.find_element("xpath",'//input[@name="password"]')

password.send_keys(get_pass)

password.send_keys(Keys.RETURN)

sleep(10)

#check for any popup by twitter

search_input = driver.find_element("xpath",'//input[@aria-label="Search query"]')
```

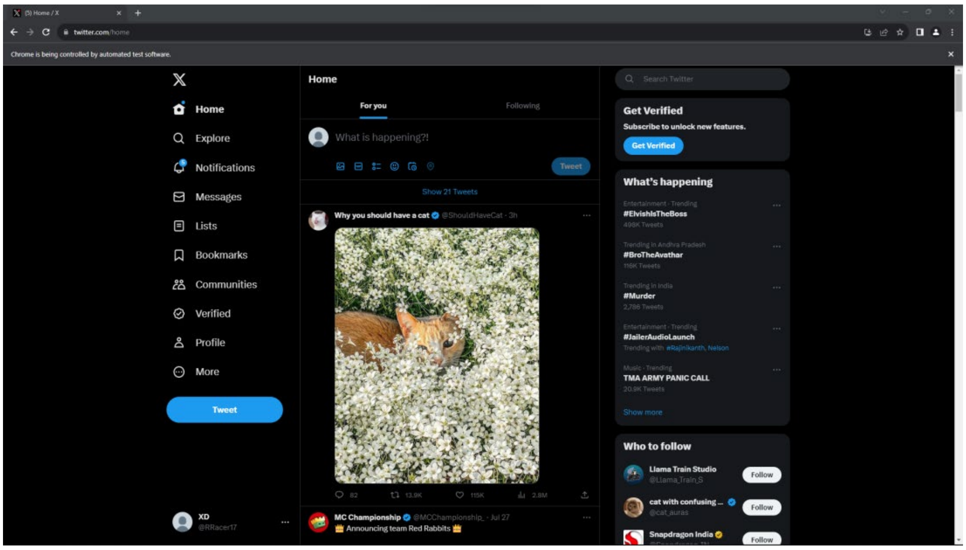
```
search_input.send_keys(get_topic)
search_input.send_keys(Keys.ENTER)
sleep(1)
driver.find_element("link text",'Latest').click()
def get_tweet_data(card):
    """Extract data from tweet card"""

    try:
        handle = card.find_element("xpath",'.//span[contains(text(), "@")]').text
    except NoSuchElementException:
        return

    try:
        postdate = card.find_element("xpath",'.//time').get_attribute('datetime')
    except NoSuchElementException:
        return

    comment = card.find_element("xpath",'.//div[2]/div[2]/div[1]').text
    responding = card.find_element("xpath",'.//div[2]/div[2]/div[2]').text
    text = comment + responding

    tweet = (handle, text)

    return tweet
#get all tweets on the page
data=[]
tweet_ids= set()
```

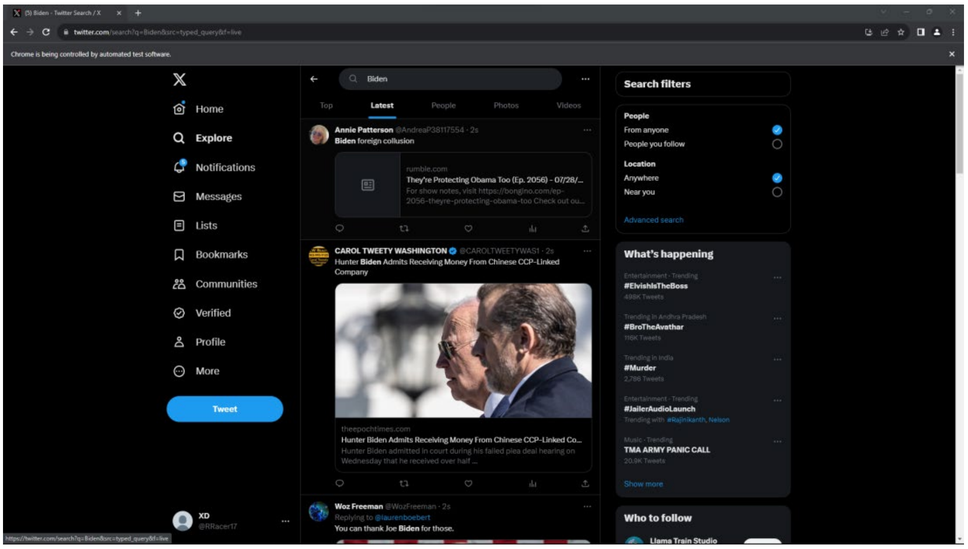
```
last_position = driver.execute_script("return window.pageYOffset;")
scrolling = True

while scrolling:
    page_cards = driver.find_elements("xpath",'//article[@data-testid="tweet"]')
    for card in page_cards[-30:]:
        tweet = get_tweet_data(card)
        if tweet:
            tweet_id = ''.join(tweet)
            if tweet_id not in tweet_ids:
                tweet_ids.add(tweet_id)
                data.append(tweet)


#ENTER A SAVE FILE NAME FOR SAVING THE .csv FILE
#RECOMMENDED TO USE NAME SAME AS TOPIC
        with open('.csv', 'w', newline='', encoding='utf-8') as f:
            header = ['Handle', 'Text']
            writer = csv.writer(f)
            writer.writerow(header)
            writer.writerows(data)


    scroll_attempt = 0
    while True:
        # check scroll position
        driver.execute_script('window.scrollTo(0, document.body.scrollHeight);')
```

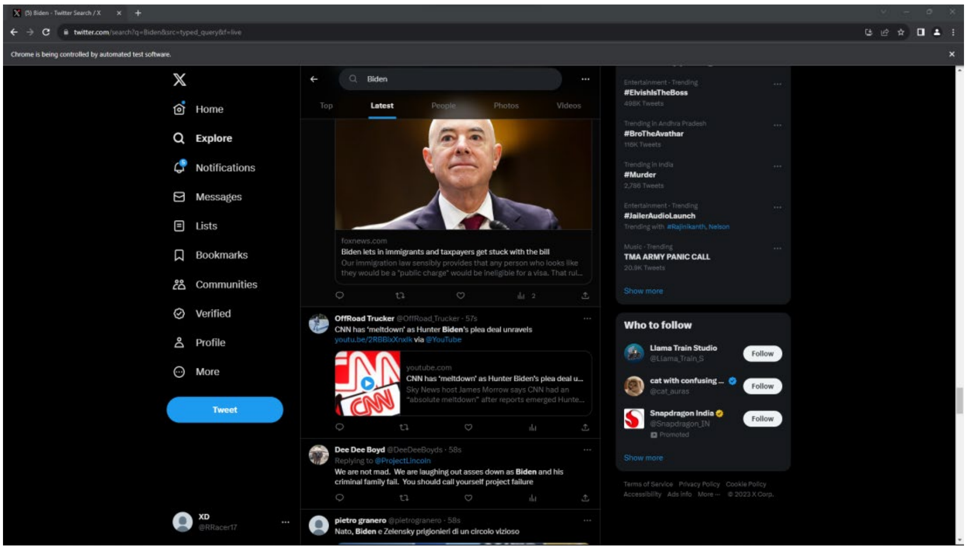
```
        sleep(2)
        curr_position = driver.execute_script("return window.pageYOffset;")
        if last_position == curr_position:
            scroll_attempt += 1

            # end of scroll region
            if scroll_attempt >= 3:
                scrolling = False
                break
            else:
                sleep(2) # attempt another scroll
        else:
            last_position = curr_position
            with open('#ENTER SAME NAME AS ABOVE SAVE.csv', 'w', newline='', encoding='utf-8') as f:
                header = ['Handle', 'Text']
                writer = csv.writer(f)
                writer.writerow(header)
                writer.writerows(data)
            break
    driver.close()
```

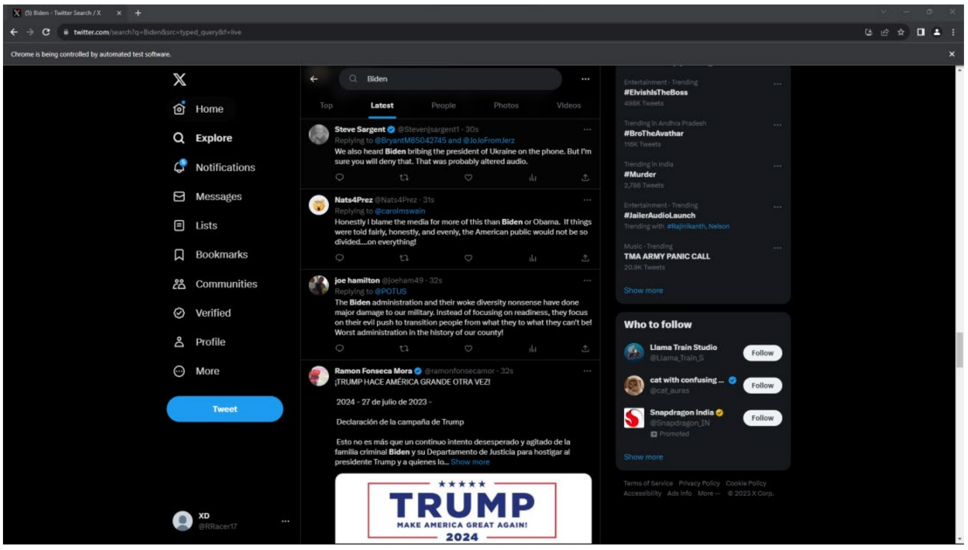

data:
Not secure
data:
Chrome is being controlled by automated test software.

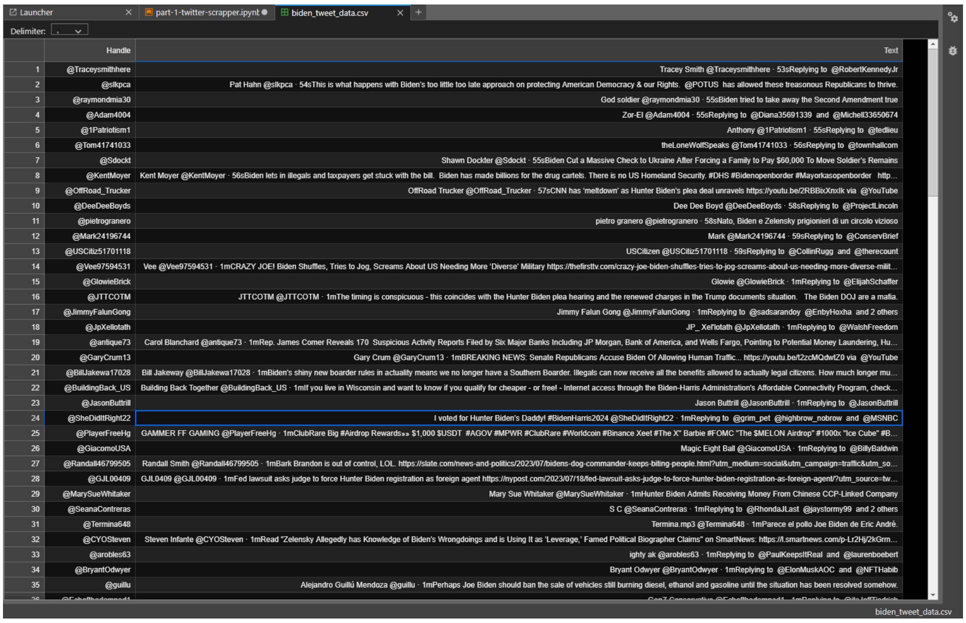

Log in to Twitter / X
twitter.com/i/flow/login
Chrome is being controlled by automated test software.
Sign in to Twitter
Sign in with Google
Sign in with Apple
or
Phone, email, or username
Next
Forgot password?
Don't have an account? Sign up


Home / X
twitter.com/home
Chrome is being controlled by automated test software.
Home
Explore
Notifications
Messages
Lists
Bookmarks
Communities
Verified
Profile
More
Tweet
XD
For you
Following
What is happening?!
Tweet
Show 21 Tweets
Why you should have a cat
MC Championship
Announcing team Red Rabbits
Search Twitter
Get Verified
Subscribe to unlock new features.
Get Verified
What's happening
#ElvishIsTheBoss
#BroTheAvathar
#Murder
#JailerAudioLaunch
TMA ARMY PANIC CALL
Show more
Who to follow
Llama Train Studio
cat with confusing ...
Snapdragon India
Follow
Follow
Follow









	Handle	Text
1	@Traceysmithhere	Tracey Smith @Traceysmithhere · 53sReplying to @RobertKennedyJr
2	@slkpca	Pat Hahn @slkpca · 54sThis is what happens with Biden's too little too late approach on protecting American Democracy & our Rights. @POTUS has allowed these treasonous Republicans to thrive.
3	@raymondmia30	God soldier @raymondmia30 · 55sBiden tried to take away the Second Amendment true
4	@Adam4004	Zor-El @Adam4004 · 55sReplying to @Diana35691339 and @Michell33650674
5	@1Patriotism1	Anthony @1Patriotism1 · 55sReplying to @tedlieu
6	@Tom41741033	theLoneWolfSpeaks @Tom41741033 · 56sReplying to @townhallcom
7	@Sdockt	Shawn Dockter @Sdockt · 55sBiden Cut a Massive Check to Ukraine After Forcing a Family to Pay $60,000 To Move Soldier's Remains
8	@KentMoyer	Kent Moyer @KentMoyer · 56sBiden lets in illegals and taxpayers get stuck with the bill. Biden has made billions for the drug cartels. There is no US Homeland Security. #DHS #Bidenopenborder #Mayorkasopenborder http...
9	@OffRoad_Trucker	OffRoad Trucker @OffRoad_Trucker · 57sCNN has 'meltdown' as Hunter Biden's plea deal unravels https://youtu.be/2RBBixXnxlk via @YouTube
10	@DeeDeeBoyds	Dee Dee Boyd @DeeDeeBoyds · 58sReplying to @ProjectLincoln
11	@pietrogranero	pietro granero @pietrogranero · 58sNato, Biden e Zelensky prigionieri di un circolo vizioso
12	@Mark24196744	Mark @Mark24196744 · 59sReplying to @ConservBrief
13	@USCitiz51701118	USCitizen @USCitiz51701118 · 59sReplying to @CollinRugg and @therecount
14	@Vee97594531	Vee @Vee97594531 · 1mCRAZY JOE! Biden Shuffles, Tries to Jog, Screams About US Needing More 'Diverse' Military https://thefirsttv.com/crazy-joe-biden-shuffles-tries-to-jog-screams-about-us-needing-more-diverse-milit...
15	@GlowieBrick	Glowie @GlowieBrick · 1mReplying to @ElijahSchaffer
16	@JTTCOTM	JTTCOTM @JTTCOTM · 1mThe timing is conspicuous - this coincides with the Hunter Biden plea hearing and the renewed charges in the Trump documents situation. The Biden DOJ are a mafia.
17	@JimmyFalunGong	Jimmy Falun Gong @JimmyFalunGong · 1mReplying to @sadsarandoy @EnbyHoxha and 2 others
18	@JpXellotath	JP_ Xel'lotath @JpXellotath · 1mReplying to @WalshFreedom
19	@antique73	Carol Blanchard @antique73 · 1mRep. James Comer Reveals 170 Suspicious Activity Reports Filed by Six Major Banks Including JP Morgan, Bank of America, and Wells Fargo, Pointing to Potential Money Laundering, Hu...
20	@GaryCrum13	Gary Crum @GaryCrum13 · 1mBREAKING NEWS: Senate Republicans Accuse Biden Of Allowing Human Traffic... https://youtu.be/t2zcMQdwtZ0 via @YouTube
21	@BillJakewa17028	Bill Jakeway @BillJakewa17028 · 1mBiden's shiny new boarder rules in actuality means we no longer have a Southern Boarder. Illegals can now receive all the benefits allowed to actually legal citizens. How much longer mu...
22	@BuildingBack_US	Building Back Together @BuildingBack_US · 1mIf you live in Wisconsin and want to know if you qualify for cheaper - or free! - Internet access through the Biden-Harris Administration's Affordable Connectivity Program, check...
23	@JasonButtrill	Jason Buttrill @JasonButtrill · 1mReplying to @JasonButtrill
24	@SheDidItRight22	I voted for Hunter Biden's Daddy! #BidenHarris2024 @SheDidItRight22 · 1mReplying to @grim_pet @highbrow_nobrow and @MSNBC
25	@PlayerFreeHg	GAMMER FF GAMING @PlayerFreeHg · 1mClubRare Big #Airdrop Rewards»» $1,000 $USDT #AGOV #MPWR #ClubRare #Worldcoin #Binance Xeet #The X" Barbie #FOMC "The $MELON Airdrop" #1000x "Ice Cube" #B...
26	@GiacomoUSA	Magic Eight Ball @GiacomoUSA · 1mReplying to @BillyBaldwin
27	@Randall46799505	Randall Smith @Randall46799505 · 1mBark Brandon is out of control, LOL. https://slate.com/news-and-politics/2023/07/bidens-dog-commander-keeps-biting-people.html?utm_medium=social&utm_campaign=traffic&utm_so...
28	@GJL00409	GJL0409 @GJL00409 · 1mFed lawsuit asks judge to force Hunter Biden registration as foreign agent https://nypost.com/2023/07/18/fed-lawsuit-asks-judge-to-force-hunter-biden-registration-as-foreign-agent/?utm_source=tw...
29	@MarySueWhitaker	Mary Sue Whitaker @MarySueWhitaker · 1mHunter Biden Admits Receiving Money From Chinese CCP-Linked Company
30	@SeanaContreras	S C @SeanaContreras · 1mReplying to @RhondaJLast @jaystormy99 and 2 others
31	@Termina648	Termina.mp3 @Termina648 · 1mParece el pollo Joe Biden de Eric André.
32	@CYOSteven	Steven Infante @CYOSteven · 1mRead "Zelensky Allegedly has Knowledge of Biden's Wrongdoings and is Using It as 'Leverage,' Famed Political Biographer Claims" on SmartNews: https://l.smartnews.com/p-Lr2Hj/2kGrm...
33	@arobles63	ighty ak @arobles63 · 1mReplying to @PaulKeepsItReal and @laurenboebert
34	@BryantOdwyer	Bryant Odwyer @BryantOdwyer · 1mReplying to @ElonMuskAOC and @NFTHabib
35	@guillu	Alejandro Guillú Mendoza @guillu · 1mPerhaps Joe Biden should ban the sale of vehicles still burning diesel, ethanol and gasoline until the situation has been resolved somehow.

PART II TRAINING

ALGORITHM

1. Import the required libraries and packages (numpy, pandas, torch, transformers, etc.).
2. Set a random seed for reproducibility (optional).
3. Load the training data and inspect the data briefly.
4. Pre-process the text data by applying a cleaning function to remove unwanted characters and patterns.
5. Tokenize the text data using the BERT tokenizer, and create dataloaders for training and validation sets.
6. Define the BERT-based model for sequence classification, specifying the number of labels (toxic, severe_toxic, etc.).
7. Transfer the model to the appropriate device (GPU if available).
8. Set the hyper parameters such as learning rate, loss function, and number of epochs.
9. Create an optimizer (AdamW) and a learning rate scheduler (linear schedule with warm-up).
10. Define functions for training and validation steps.
11. Iterate over the specified number of epochs: a. Train the model using the training dataloader and calculate training loss and accuracy. b. Validate the model using the validation dataloader and calculate validation loss, accuracy, and predicted probabilities. c. Save the model if the validation loss improves.
12. Plot the training and validation losses, as well as the training and validation accuracies over the epochs.
13. Calculate the ROC curve and the Area Under the Curve (AUC) for the validation set.

PSEUDOCODE

- # Step 1: Import required libraries
- import numpy as np
- import pandas as pd
- import torch
- import transformers
- import torch.nn as nn
- from torch.utils.data import DataLoader, Dataset
- from transformers import AdamW, get_linear_schedule_with_warmup
- # ... (other imports)
- # Step 2: Set a random seed (optional)
- SEED = 34
- # ... (random seed function)
- # Step 3: Load and inspect the training data
- train = pd.read_csv('path/to/train.csv', nrows=200)
- # ... (brief data inspection)
- # Step 4: Text preprocessing
- # ... (clean_text function)
- # Step 5: Tokenize the text and create dataloaders
- # ... (BertDataSet class)
- # Step 6: Define the BERT-based model
- model = transformers.BertForSequenceClassification.from_pretrained('bert-base-cased', num_labels=6)
- # ... (model transfer to device)
- # Step 7: Set hyperparameters
- epochs = 5
- LR = 2e-5
- # Step 8: Create optimizer and scheduler
- optimizer = AdamW(model.parameters(), LR, betas=(0.9, 0.999), weight_decay=1e-2, correct_bias=False)
- scheduler = get_linear_schedule_with_warmup(optimizer, num_steps, train_steps)
- # Step 9: Define loss function
- loss_fn = nn.BCEWithLogitsLoss()
- # Step 10: Define functions for training and validation
- # ... (training function)
- # ... (validation function)
- # Step 11: Main training loop
- best_score = 1000
- train_accs = []
- valid_accs = []

- train_losses = []
- valid_losses = []
- for epoch in range(epochs):
- # ... (training step)
- # ... (validation step)
- # ... (model saving if validation loss improves)
- # Step 12: Plot training and validation losses and accuracies
- # Step 13: Calculate ROC curve and AUC for the validation set

EXECUATABLE CODE

```
import numpy as np

import pandas as pd

import os

import random

import time


import re

import string

import nltk

from nltk.corpus import stopwords


import matplotlib.pyplot as plt

import seaborn as sns

sns.set(style="ticks", context="talk")

plt.style.use('dark_background')


from tqdm import tqdm


import torch

import torch.nn as nn
```

```
import torch.nn.functional as func
from torch.utils.data import DataLoader, Dataset

import transformers
from transformers import AdamW, get_linear_schedule_with_warmup

import tokenizers
from sklearn.metrics import mean_squared_error, roc_auc_score, roc_curve, auc

import warnings
warnings.simplefilter('ignore')
SEED = 34

def random_seed(SEED):
    random.seed(SEED)
    os.environ['PYTHONHASHSEED'] = str(SEED)
    np.random.seed(SEED)
    torch.manual_seed(SEED)
    torch.cuda.manual_seed(SEED)
    torch.cuda.manual_seed_all(SEED)
    torch.backends.cudnn.deterministic = True

random_seed(SEED)
train = pd.read_csv('../Toxic-Comment-Classification/input/train.csv', nrows = 200 )
train.head()
temp = train[train['toxic'] == 1]
temp.head()
```

```
print(len(train['comment_text'][10]), 'Total Characters')

train['comment_text'][10]

labels = train.drop(['id', 'comment_text'], axis = 1)

unique_values = lambda x: train[x].unique()

[unique_values(col) for col in labels.columns.tolist()]

test = pd.read_csv('../Toxic-Comment-Classification/input/train.csv', nrows = 10)

test.head()

test_labels = pd.read_csv('../Toxic-Comment-Classification/input/test_labels.csv',
nrows = 10)

test_labels.head()

submission = pd.read_csv('../Toxic-Comment-
Classification/input/sample_submission.csv', nrows = 10)

submission.head()

train.isnull().sum()

test.isnull().sum()

df_train = train.drop(['id', 'comment_text'], axis = 1)

label_counts = df_train.sum()

df_counts = pd.DataFrame(label_counts)

df_counts.rename(columns = {0:'counts'}, inplace = True)

df_counts = df_counts.sort_values('counts', ascending = False)

df_counts

train.shape, test.shape

def clean_text(text):


    text = re.sub('\[.*?\]', '', text)

    #pattern = [zero or more character]


    text = re.sub('https?://\S+|www\.\S+', '', text)
```

```
    #pattern = removes (http),://, 'and' www.

    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    #pattern = any punctionation

    text = re.sub('\n', '', text)
    #pattern = any new line

    text = re.sub('\w*\d\w*', '', text)
    #pattern = any from[a-zA-Z0-9_], any from[0-9], any from [a-zA-Z0-9_]

    return text
%%time
train['clean_text'] = train['comment_text'].apply(str).apply(lambda x: clean_text(x))
test['clean_text'] = test['comment_text'].apply(str).apply(lambda x: clean_text(x))
kfold = 5
train['kfold'] = train.index % kfold
train.index % kfold
p_train = train[train["kfold"] != 0].reset_index(drop = True)
p_valid = train[train["kfold"] == 0].reset_index(drop = True)
p_train.head()
tokenizer = transformers.BertTokenizer.from_pretrained('bert-base-cased')
%%time
senten_len = []
#tqdm is progress bar
for sentence in tqdm(p_train['clean_text']):
```

```
    token_words = tokenizer.encode_plus(sentence)['input_ids']
    senten_len.append(len(token_words))
max_len = 256
```

We define a class BertDataSet with Dataset as super class and overwirte the init, len and getitem function in it. It will get the comment list and relevant toxic labels (6 labels in this case) and creates token ids and attention mask to distinguish the comments from the zero padding.

torch.tensors are designed to be used in the context of gradient descent optimization, and therefore they hold not only a tensor with numeric values, but (and more importantly) the computational graph leading to these values. This computational graph is then used (using the chain rule of derivatives) to compute the derivative of the loss function w.r.t each of the independent variables used to compute the loss.

```
class BertDataSet(Dataset):
#Bidirectional Encoder Representations from Transformers

    def __init__(self, sentences, toxic_labels):
        self.sentences = sentences
        #target is a matrix with shape [#1 x #6(toxic, obscene, etc)]
        self.targets = toxic_labels.to_numpy()

    def __len__(self):
        return len(self.sentences)



    def __getitem__(self, idx):
        sentence = self.sentences[idx]
        bert_senten = tokenizer.encode_plus(sentence,
                                            add_special_tokens = True, # [CLS],[SEP]
```

```
                                            max_length = max_len,
                                            pad_to_max_length = True,
                                            truncation = True,
                                            return_attention_mask = True
                                            )
        ids = torch.tensor(bert_senten['input_ids'], dtype = torch.long)
        mask = torch.tensor(bert_senten['attention_mask'], dtype = torch.long)
        toxic_label = torch.tensor(self.targets[idx], dtype = torch.float)



        return {
            'ids' : ids,
            'mask' : mask,
            'toxic_label':toxic_label
        }
train_dataset = BertDataSet(p_train['clean_text'], p_train[['toxic',
'severe_toxic','obscene', 'threat', 'insult','identity_hate']])
valid_dataset = BertDataSet(p_valid['clean_text'], p_valid[['toxic',
'severe_toxic','obscene', 'threat', 'insult','identity_hate']])
# for a in train_dataset:
#     print(a)
#     break
train_batch = 32
valid_batch = 32
train_dataloader = DataLoader(train_dataset, batch_size = train_batch, pin_memory =
True, num_workers = 4, shuffle = True)
valid_dataloader = DataLoader(valid_dataset, batch_size = valid_batch, pin_memory
= True, num_workers = 4, shuffle = False)
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

device

%%time

model = transformers.BertForSequenceClassification.from_pretrained('bert-base-cased', num_labels = 6)

model.to(device)

model.train()

%%time

for a in train_dataloader:

    ids = a['ids'].to(device)

    mask = a['mask'].to(device)

    output = model(ids, mask)

    break

output

func.softmax(output['logits'], dim = 1)

output_probs = func.softmax(output['logits'], dim = 1)

torch.max(output_probs, dim = 1)

epochs = 5

LR = 2e-5 #Learning rate

optimizer = AdamW(model.parameters(), LR, betas = (0.9, 0.999), weight_decay = 1e-2, correct_bias = False)

train_steps = int((len(train) * epochs)/train_batch)

num_steps = int(train_steps * 0.1)

scheduler = get_linear_schedule_with_warmup(optimizer, num_steps, train_steps)

loss_fn = nn.BCEWithLogitsLoss()

loss_fn.to(device)

scaler = torch.cuda.amp.GradScaler()

def training(train_dataloader, model, optimizer, scheduler):
```

```
    model.train()
    torch.backends.cudnn.benchmark = True
    correct_predictions = 0

    for a in train_dataloader:
        losses = []
        optimizer.zero_grad()

        #allpreds = []
        #alltargets = []

        with torch.cuda.amp.autocast():

            ids = a['ids'].to(device, non_blocking = True)
            mask = a['mask'].to(device, non_blocking = True)

            output = model(ids, mask) #This gives model as output, however we want the
values at the output
            output = output['logits'].squeeze(-1).to(torch.float32)

            output_probs = torch.sigmoid(output)
            preds = torch.where(output_probs > 0.5, 1, 0)

            toxic_label = a['toxic_label'].to(device, non_blocking = True)
            loss = loss_fn(output, toxic_label)

            losses.append(loss.item())
```

```
            #allpreds.append(output.detach().cpu().numpy())

            #alltargets.append(toxic.detach().squeeze(-1).cpu().numpy())

            correct_predictions += torch.sum(preds == toxic_label)


        scaler.scale(loss).backward() #Multiplies ('scales') a tensor or list of tensors by
the scale factor.

                                      #Returns scaled outputs. If this instance of GradScaler is not
enabled, outputs are returned unmodified.

        scaler.step(optimizer) #Returns the return value of optimizer.step(*args,
**kwargs).

        scaler.update() #Updates the scale factor.If any optimizer steps were skipped the
scale is multiplied by backoff_factor to reduce it.

                        #If growth_interval unskipped iterations occurred consecutively, the
scale is multiplied by growth_factor to increase it

        scheduler.step() # Update learning rate schedule


    losses = np.mean(losses)

    corr_preds = correct_predictions.detach().cpu().numpy()

    accuracy = corr_preds/(len(p_train)*6)


    return losses, accuracy

def validating(valid_dataloader, model):


    model.eval()

    correct_predictions = 0

    all_output_probs = []


    for a in valid_dataloader:

        losses = []
```

```
        ids = a['ids'].to(device, non_blocking = True)
        mask = a['mask'].to(device, non_blocking = True)
        output = model(ids, mask)
        output = output['logits'].squeeze(-1).to(torch.float32)
        output_probs = torch.sigmoid(output)
        preds = torch.where(output_probs > 0.5, 1, 0)

        toxic_label = a['toxic_label'].to(device, non_blocking = True)
        loss = loss_fn(output, toxic_label)
        losses.append(loss.item())
        all_output_probs.extend(output_probs.detach().cpu().numpy())

        correct_predictions += torch.sum(preds == toxic_label)
        corr_preds = correct_predictions.detach().cpu().numpy()

    losses = np.mean(losses)
    corr_preds = correct_predictions.detach().cpu().numpy()
    accuracy = corr_preds/(len(p_valid)*6)

    return losses, accuracy, all_output_probs
%%time

best_score = 1000
train_accs = []
valid_accs = []
train_losses = []
valid_losses = []
```

```
for eboch in tqdm(range(epochs)):

    train_loss, train_acc = training(train_dataloader, model, optimizer, scheduler)
    valid_loss, valid_acc, valid_probs = validating(valid_dataloader, model)

    print('train losses: %.4f' % train_loss, 'train accuracy: %.3f' % train_acc)
    print('valid losses: %.4f' % valid_loss, 'valid accuracy: %.3f' % valid_acc)
    train_losses.append(train_loss)
    valid_losses.append(valid_loss)
    train_accs.append(train_acc)
    valid_accs.append(valid_acc)


    if valid_loss < best_score:
        best_score = valid_loss
        print('Found a good model!')
        state = {
            'state_dict': model.state_dict(),
            'optimizer_dict': optimizer.state_dict(),
            'best_score': best_score
        }
        torch.save(state, 'best_model.pth')
    else:
        pass
x = np.arange(epochs)
fig, ax = plt.subplots(1, 2, figsize = (15,4))
```

```
ax[0].plot(x, train_losses)

ax[0].plot(x, valid_losses)

ax[0].set_ylabel('Losses', weight = 'bold')

ax[0].set_xlabel('Epochs')

ax[0].grid(alpha = 0.3)

ax[0].legend(labels = ['train losses', 'valid losses'])


ax[1].plot(x, train_accs)

ax[1].plot(x, valid_accs)

ax[1].set_ylabel('Accuracy', weight = 'bold')

ax[1].set_xlabel('Epochs')

ax[1].legend(labels = ['train acc', 'valid acc'])


ax[1].grid(alpha = 0.3)

fig.suptitle('Fold = 0', weight = 'bold')

valid_loss, valid_acc, valid_probs = validating(valid_dataloader, model)

valid_probs = np.asarray(valid_probs).flatten()

y_valid = p_valid[['toxic', 'severe_toxic','obscene', 'threat',
'insult','identity_hate']].to_numpy().flatten()

fpr, tpr, _ = roc_curve(y_valid, valid_probs)

fig, ax = plt.subplots()

ax.plot(fpr, tpr)

ax.set_title('ROC Curv')

ax.set_xlabel('FPR')

ax.set_ylabel('TPR')

plt.show()

auc(fpr, tpr)
```

In [3]:

```
train = pd.read_csv('/kaggle/input/jigsaw-toxic-comment-classification-challenge/train.csv', nrows =
200 )
train.head()
```

Out[3]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

In [4]:

```
temp = train[train['toxic'] == 1]
temp.head()
```

Out[4]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
6	0002bcb3da6cb337	COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1	1	1	0	1	0
12	0005c987bdfc9d4b	Hey... what is it..\n@ | talk .\nWhat is it......	1	0	0	0	0	0
16	0007e25b2121310b	Bye! \n\nDon't look, come or think of comming ...	1	0	0	0	0	0
42	001810bf8c45bf5f	You are gay or antisemmitian? \n\nArchangel WH...	1	0	1	0	1	1
43	00190820581d90ce	FUCK YOUR FILTHY MOTHER IN THE ASS, DRY!	1	0	1	0	1	0

In [12]:

```
df_train = train.drop(['id', 'comment_text'], axis = 1)
label_counts = df_train.sum()
df_counts = pd.DataFrame(label_counts)
df_counts.rename(columns = {0:'counts'}, inplace = True)
df_counts = df_counts.sort_values('counts', ascending = False)
df_counts
```

Out[12]:

	counts
toxic	20
insult	12
obscene	11
severe_toxic	3
identity_hate	3
threat	2

```
In [14]:
def clean_text(text):

    text = re.sub('\[.*?\]', '', text)
    #pattern = [zero or more character]

    text = re.sub('https?://\S+|www\.\S+', '', text)
    #pattern = removes (http),://, 'and' www.

    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    #pattern = any punctionation

    text = re.sub('\n', '', text)
    #pattern = any new line

    text = re.sub('\w*\d\w*', '', text)
    #pattern = any from[a-zA-Z0-9_], any from[0-9], any from [a-zA-Z0-9_]

    return text
```

```
In [15]:
%%time
train['clean_text'] = train['comment_text'].apply(str).apply(lambda x: clean_text(x))
test['clean_text'] = test['comment_text'].apply(str).apply(lambda x: clean_text(x))

CPU times: user 37.6 ms, sys: 0 ns, total: 37.6 ms
Wall time: 37.6 ms
```

```
In [19]:
tokenizer = transformers.BertTokenizer.from_pretrained('bert-base-cased')

Downloading (…)solve/main/vocab.txt: 100%        213k/213k [00:00<00:00, 4.70MB/s]

Downloading (…)okenizer_config.json: 100%        29.0/29.0 [00:00<00:00, 1.88kB/s]

Downloading (…)lve/main/config.json: 100%        570/570 [00:00<00:00, 40.1kB/s]
```

```
In [22]:
class BertDataSet(Dataset):
#Bidirectional Encoder Representations from Transformers

    def __init__(self, sentences, toxic_labels):
        self.sentences = sentences
        #target is a matrix with shape [#1 x #6(toxic, obscene, etc)]
        self.targets = toxic_labels.to_numpy()

    def __len__(self):
        return len(self.sentences)


    def __getitem__(self, idx):
        sentence = self.sentences[idx]
        bert_senten = tokenizer.encode_plus(sentence,
                                            add_special_tokens = True, # [CLS],[SEP]
                                            max_length = max_len,
                                            pad_to_max_length = True,
                                            truncation = True,
                                            return_attention_mask = True
                                             )
        ids = torch.tensor(bert_senten['input_ids'], dtype = torch.long)
        mask = torch.tensor(bert_senten['attention_mask'], dtype = torch.long)
        toxic_label = torch.tensor(self.targets[idx], dtype = torch.float)


        return {
            'ids' : ids,
            'mask' : mask,
            'toxic_label':toxic_label
        }
```

```
In [23]:
train_dataset = BertDataSet(p_train['clean_text'], p_train[['toxic', 'severe_toxic','obscene', 'thre
at', 'insult','identity_hate']])
valid_dataset = BertDataSet(p_valid['clean_text'], p_valid[['toxic', 'severe_toxic','obscene', 'thre
at', 'insult','identity_hate']])
```

```
In [27]:
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device

Out[27]:
device(type='cuda')
```

```
Out[28]:
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(28996, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=6, bias=True)
)
```

```
  0%|          | 0/5 [00:00<?, ?it/s]

train losses: 0.5616 train accuracy: 0.512
valid losses: 0.4786 valid accuracy: 0.796
Found a good model!

 20%|██        | 1/5 [00:04<00:18,  4.70s/it]

train losses: 0.3512 train accuracy: 0.919
valid losses: 0.2308 valid accuracy: 0.954
Found a good model!

 40%|████      | 2/5 [00:14<00:22,  7.51s/it]

train losses: 0.1943 train accuracy: 0.958
valid losses: 0.1367 valid accuracy: 0.954
Found a good model!

 60%|██████    | 3/5 [00:25<00:18,  9.04s/it]

train losses: 0.2047 train accuracy: 0.958
valid losses: 0.1115 valid accuracy: 0.954
Found a good model!

 80%|████████  | 4/5 [00:35<00:09,  9.49s/it]

train losses: 0.1430 train accuracy: 0.960
valid losses: 0.0876 valid accuracy: 0.954
Found a good model!

100%|██████████| 5/5 [00:41<00:00,  8.22s/it]

CPU times: user 14.4 s, sys: 9.89 s, total: 24.3 s
Wall time: 41.1 s
```

PART III ANALYSIS

ALGORITHM

1. Import the required libraries and packages.
2. Read the train, test, and submission data from CSV files.
3. Set the random seed for reproducibility.
4. Define a function **clean_text** to clean the text data.
5. Apply the **clean_text** function to the 'comment_text' column in the train and test data to preprocess the text.
6. Split the train data into k-folds for cross-validation.
7. Initialize the BERT tokenizer with 'bert-base-cased'.
8. Define the **BertDataSet** class, subclassed from **Dataset**, to prepare data for training and validation.
9. Set the hyperparameters: epochs, batch sizes, and device (CPU or GPU).
10. Define the loss function, optimizer, and scaler for mixed-precision training.
11. Implement the **training** function for training the BERT model and updating the weights.
12. Implement the **validating** function for validating the model on the validation set.
13. For each fold in k-folds: a. Prepare the train and validation datasets using **BertDataSet**. b. Load the BERT model for classification. c. Train the model and save the best model based on the validation loss. d. Store the validation loss, accuracy, and probabilities for each fold.
14. Calculate the mean of the best validation losses for all folds to evaluate the overall model performance.
15. Define the **BERTinferenceDataSet** class to prepare data for inference on the new dataset.
16. Load the trained BERT model.
17. Predict the probabilities for the new dataset using the trained model.
18. Format the results and create the submission file.
19. Visualize the distribution of toxicity categories in the submission.

20. Visualize the ROC curve for the model's performance.

PSEUDOCODE

- Import required libraries and packages
- Read train, test, and submission data
- Set random seed
- Define clean_text function to preprocess text data
- for each comment in train['comment_text']:
- Clean the text using clean_text function
- for each comment in test['comment_text']:
- Clean the text using clean_text function
- Split train data into k-folds for cross-validation
- Initialize BERT tokenizer with 'bert-base-cased'
- Define BertDataSet class for preparing data
- Set hyperparameters: epochs, batch sizes, and device
- Define loss function, optimizer, and scaler for mixed-precision training
- Define training function for model training
- Define validating function for model validation
- for each fold in k-folds:
- Prepare train and validation datasets using BertDataSet
- Load BERT model for classification
- Train the model and save the best model based on validation loss
- Store validation loss, accuracy, and probabilities for each fold
- Calculate mean of best validation losses for all folds to evaluate overall model performance
- Define BERTinferenceDataSet class for preparing inference data
- Load trained BERT model
- Predict probabilities for new dataset using the trained model
- Format results and create submission file
- Visualize distribution of toxicity categories in submission
- Visualize ROC curve for model performance

EXECUATABLE CODE

```
%%time
import numpy as np
import pandas as pd
import os
import random
import time

import re
import string
import nltk
from nltk.corpus import stopwords

import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="ticks", context="talk")
plt.style.use('dark_background')

from tqdm import tqdm

import torch
import torch.nn as nn
import torch.nn.functional as func
from torch.utils.data import DataLoader, Dataset

import transformers
from transformers import AdamW, get_linear_schedule_with_warmup
```

```
import tokenizers
from sklearn.metrics import mean_squared_error, roc_auc_score, roc_curve, auc


import warnings
warnings.simplefilter('ignore')


train = pd.read_csv('../input/jigsaw-toxic-comment-classification-challenge/train.csv.zip', nrows = 2000)
test = pd.read_csv('../input/jigsaw-toxic-comment-classification-challenge/test.csv.zip', nrows = 100)
submission = pd.read_csv('../input/jigsaw-toxic-comment-classification-challenge/sample_submission.csv.zip')


SEED = 34
def random_seed(SEED):
    random.seed(SEED)
    os.environ['PYTHONHASHSEED'] = str(SEED)
    np.random.seed(SEED)
    torch.manual_seed(SEED)
    torch.cuda.manual_seed(SEED)
    torch.cuda.manual_seed_all(SEED)
    torch.backends.cudnn.deterministic = True
random_seed(SEED)


def clean_text(text):


    text = re.sub('\[.*?\]', '', text)
```

```
    text = re.sub('https?://\S+|www\.\S+', '', text)

    text = re.sub('<.*?>+', '', text)

    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)

    text = re.sub('\n', '', text)

    text = re.sub('\w*\d\w*', '', text)

    return text



train['clean_text'] = train['comment_text'].apply(str).apply(lambda x: clean_text(x))

test['clean_text'] = test['comment_text'].apply(str).apply(lambda x: clean_text(x))


kfold = 5

train['kfold'] = train.index % kfold


tokenizer = transformers.BertTokenizer.from_pretrained('bert-base-cased')

max_len = 200


class BertDataSet(Dataset):


    def __init__(self, sentences, toxic_labels):

        self.sentences = sentences

        #target is a matrix with shape [#1 x #6(toxic, obscene, etc)]

        self.targets = toxic_labels.to_numpy()


    def __len__(self):

        return len(self.sentences)
```

```
    def __getitem__(self, idx):
        sentence = self.sentences[idx]
        bert_senten = tokenizer.encode_plus(sentence,
                                            add_special_tokens = True, # [CLS],[SEP]
                                            max_length = max_len,
                                            pad_to_max_length = True,
                                            truncation = True,
                                            return_attention_mask = True
                                             )
        ids = torch.tensor(bert_senten['input_ids'], dtype = torch.long)
        mask = torch.tensor(bert_senten['attention_mask'], dtype = torch.long)
        toxic_label = torch.tensor(self.targets[idx], dtype = torch.float)



        return {
            'ids' : ids,
            'mask' : mask,
            'toxic_label':toxic_label
        }

epochs = 5
train_batch = 32
valid_batch = 32
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

loss_fn = nn.BCEWithLogitsLoss()
```

```
loss_fn.to(device)
scaler = torch.cuda.amp.GradScaler()


def training(train_dataloader, model, optimizer, scheduler):
    model.train()
    torch.backends.cudnn.benchmark = True
    correct_predictions = 0

    for a in train_dataloader:
        losses = []
        optimizer.zero_grad()

        #allpreds = []
        #alltargets = []

        with torch.cuda.amp.autocast():

            ids = a['ids'].to(device, non_blocking = True)
            mask = a['mask'].to(device, non_blocking = True)

            output = model(ids, mask) #This gives model as output, however we want the
values at the output
            output = output['logits'].squeeze(-1).to(torch.float32)

            output_probs = torch.sigmoid(output)
            preds = torch.where(output_probs > 0.5, 1, 0)
```

```
            toxic_label = a['toxic_label'].to(device, non_blocking = True)
            loss = loss_fn(output, toxic_label)

            losses.append(loss.item())
            #allpreds.append(output.detach().cpu().numpy())
            #alltargets.append(toxic.detach().squeeze(-1).cpu().numpy())
            correct_predictions += torch.sum(preds == toxic_label)

        scaler.scale(loss).backward() #Multiplies ('scales') a tensor or list of tensors by the scale factor.
                                      #Returns scaled outputs. If this instance of GradScaler is not enabled, outputs are returned unmodified.
        scaler.step(optimizer) #Returns the return value of optimizer.step(*args, **kwargs).
        scaler.update() #Updates the scale factor.If any optimizer steps were skipped the scale is multiplied by backoff_factor to reduce it.
                        #If growth_interval unskipped iterations occurred consecutively, the scale is multiplied by growth_factor to increase it
        scheduler.step() # Update learning rate schedule

    losses = np.mean(losses)
    corr_preds = correct_predictions.detach().cpu().numpy()
    accuracy = corr_preds/(len(p_train)*6)

    return losses, accuracy

def validating(valid_dataloader, model):

    model.eval()
```

```
    correct_predictions = 0
    all_output_probs = []

    for a in valid_dataloader:
        losses = []
        ids = a['ids'].to(device, non_blocking = True)
        mask = a['mask'].to(device, non_blocking = True)
        output = model(ids, mask)
        output = output['logits'].squeeze(-1).to(torch.float32)
        output_probs = torch.sigmoid(output)
        preds = torch.where(output_probs > 0.5, 1, 0)

        toxic_label = a['toxic_label'].to(device, non_blocking = True)
        loss = loss_fn(output, toxic_label)
        losses.append(loss.item())
        all_output_probs.extend(output_probs.detach().cpu().numpy())

        correct_predictions += torch.sum(preds == toxic_label)
        corr_preds = correct_predictions.detach().cpu().numpy()

    losses = np.mean(losses)
    corr_preds = correct_predictions.detach().cpu().numpy()
    accuracy = corr_preds/(len(p_valid)*6)

    return losses, accuracy, all_output_probs
%%time
```

```
best_scores = []
for fold in tqdm(range(0,5)):

    # initializing the data
    p_train = train[train['kfold'] != fold].reset_index(drop = True)
    p_valid = train[train['kfold'] == fold].reset_index(drop = True)

    train_dataset = BertDataSet(p_train['clean_text'], p_train[['toxic',
'severe_toxic','obscene', 'threat', 'insult','identity_hate']])
    valid_dataset = BertDataSet(p_valid['clean_text'], p_valid[['toxic',
'severe_toxic','obscene', 'threat', 'insult','identity_hate']])

    train_dataloader = DataLoader(train_dataset, batch_size = train_batch, shuffle =
True, num_workers = 4, pin_memory = True)
    valid_dataloader = DataLoader(valid_dataset, batch_size = valid_batch, shuffle =
False, num_workers = 4, pin_memory = True)

    model =
transformers.BertForSequenceClassification.from_pretrained("../input/bert-base-
cased", num_labels = 6)
    model.to(device)

    LR = 2e-5
    optimizer = AdamW(model.parameters(), LR,betas = (0.9, 0.999), weight_decay =
1e-2) # AdamW optimizer

    train_steps = int(len(p_train)/train_batch * epochs)
    num_steps = int(train_steps * 0.1)

    scheduler = get_linear_schedule_with_warmup(optimizer, num_steps, train_steps)
```

```
best_score = 1000
train_accs = []
valid_accs = []
train_losses = []
valid_losses = []
best_valid_probs = []

print("-------------- Fold = " + str(fold) + "-------------")

for epoch in tqdm(range(epochs)):
    print("-------------- Epoch = " + str(epoch) + "-------------")

    train_loss, train_acc = training(train_dataloader, model, optimizer, scheduler)
    valid_loss, valid_acc, valid_probs = validating(valid_dataloader, model)

    train_losses.append(train_loss)
    train_accs.append(train_acc)
    valid_losses.append(valid_loss)
    valid_accs.append(valid_acc)

    print('train losses: %.4f' %(train_loss), 'train accuracy: %.3f' %(train_acc))
    print('valid losses: %.4f' %(valid_loss), 'valid accuracy: %.3f' %(valid_acc))

    if (valid_loss < best_score):

        best_score = valid_loss
```

```
        print("Found an improved model! :)")

        state = {'state_dict': model.state_dict(),
                 'optimizer_dict': optimizer.state_dict(),
                 'best_score':best_score
                }

        torch.save(state, "model" + str(fold) + ".pth")
        best_valid_prob = valid_probs
        torch.cuda.memory_summary(device = None, abbreviated = False)
    else:
        pass


best_scores.append(best_score)
best_valid_probs.append(best_valid_prob)

##Plotting the result for each fold
x = np.arange(epochs)
fig, ax = plt.subplots(1, 2, figsize = (15,4))
ax[0].plot(x, train_losses)
ax[0].plot(x, valid_losses)
ax[0].set_ylabel('Losses', weight = 'bold')
ax[0].set_xlabel('Epochs')
ax[0].grid(alpha = 0.3)
ax[0].legend(labels = ['train losses', 'valid losses'])
```

```
    ax[1].plot(x, train_accs)
    ax[1].plot(x, valid_accs)
    ax[1].set_ylabel('Accuracy', weight = 'bold')
    ax[1].set_xlabel('Epochs')
    ax[1].legend(labels = ['train acc', 'valid acc'])


    ax[1].grid(alpha = 0.3)
    fig.suptitle('Fold = '+str(fold), weight = 'bold')
best_scores
print('Mean of',kfold, 'folds for best loss in', epochs, 'epochs cross-validation folds is %.4f.' %(np.mean(best_scores)))
def predicting(test_dataloader, model, pthes):
    allpreds = []


    for pth in pthes:
        state = torch.load(pth)
        model.load_state_dict(state['state_dict'])
        model.to(device)
        model.eval()
        preds = []
        with torch.no_grad():
            for a in test_dataloader:
                ids = a['ids'].to(device)
                mask = a['mask'].to(device)
                output = model(ids, mask)
                output = output['logits'].squeeze(-1)
                output_probs = torch.sigmoid(output)
```

```
            preds.append(output_probs.cpu().numpy())
        preds = np.concatenate(preds)
        allpreds.append(preds)

    return allpreds
pthes = [os.path.join("./",s) for s in os.listdir("./") if ".pth" in s]
allpreds = predicting(valid_dataloader, model, pthes)
valid_probs = np.zeros((len(p_valid),6))
for i in range(kfold):
    valid_probs += allpreds[i]
valid_probs = valid_probs / kfold
valid_probs = np.asarray(valid_probs).flatten()
#valid_probs = allpreds[0].flatten() #This line is used when trianing for one model and not k-fold model
y_valid = p_valid[['toxic', 'severe_toxic','obscene', 'threat', 'insult','identity_hate']].to_numpy().flatten()
fpr, tpr, _ = roc_curve(y_valid, valid_probs)
print('auc score for kfold =', kfold, 'models is: %.2f' %(auc(fpr, tpr)*100))
fig, ax = plt.subplots()
ax.plot(fpr, tpr)
ax.set_title('ROC Curv')
ax.set_xlabel('FPR')
ax.set_ylabel('TPR')
plt.show()
```

Inference

```
class BERTinferenceDataSet(Dataset):

    def __init__(self, sentences):
```

```
        self.sentences = sentences

    def __len__(self):
        return len(self.sentences)

    def __getitem__(self, idx):
        sentence = self.sentences[idx]
        bert_sent = tokenizer.encode_plus(sentence,
                                add_special_tokens = True, #[SEP][PAD]
                                max_length = max_len,
                                pad_to_max_length = True,
                                truncation = True)

        ids = torch.tensor(bert_sent['input_ids'], dtype = torch.long)
        mask = torch.tensor(bert_sent['attention_mask'], dtype = torch.long)

        return{
            'ids' : ids,
            'mask' : mask
            }
tweets_data = pd.read_csv('/kaggle/input/tweetdata/biden_tweets_clean.csv')
tweets_data.head()
tweets_data['clean_text'] = tweets_data['comment_text'].apply(str).apply(lambda x: clean_text(x))

tweets_data.head()
# #reducing data set from 1167 to 16
```

```
# tweets_data = tweets_data.iloc[:-2264]

len(tweets_data)

test_batch = 32

test_dataset = BERTinferenceDataSet(tweets_data['clean_text'])

test_dataloader = DataLoader(test_dataset, batch_size = test_batch, shuffle = False,
num_workers = 4, pin_memory = True)

pthes = [os.path.join("../input/final-models",s) for s in os.listdir('../input/final-models')
if ".pth" in s]

pthes

#/kaggle/input/final-models

model = transformers.BertForSequenceClassification.from_pretrained("bert-base-
cased", num_labels = 6)

allpreds = predicting(test_dataloader, model, pthes)

print('allpreds is an array with the shape of:',len(allpreds), 'x',len(allpreds[0]),
'x',len(allpreds[0][0]))

allpreds[0][0]

preds = np.zeros((len(test_dataset),6))

for i in range(kfold):

    preds += allpreds[i]

preds = preds / kfold

results = pd.DataFrame(preds)

submission = pd.concat([test,results], axis = 1).drop(['comment_text', 'clean_text'],
axis = 1)

submission.rename(columns = { 0:'toxic', 1:'severe_toxic', 2:'obscene', 3:'threat',
4:'insult', 5:'identity_hate'}, inplace = True)

submission.to_csv("submission.csv", index = False)

s = pd.read_csv('/kaggle/working/submission.csv')

s

import matplotlib.pyplot as plt
```

```
import pandas as pd



submission["toxicity_category"] = "Non-toxic"


# set the threshold

threshold = 0.2


# create a dictionary to keep the count of each category

category_counts =
{'toxic':0,'severe_toxic':0,'obscene':0,'threat':0,'insult':0,'identity_hate':0,'Non-toxic':0}


#iterate over all the categories

for col in ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']:

    submission.loc[submission[col] > threshold, "toxicity_category"] = col

    category_counts[col] = submission[submission["toxicity_category"] ==
col].shape[0]


# non-toxic comments

category_counts['Non-toxic'] = submission[submission["toxicity_category"] == "Non-
toxic"].shape[0]


# Data to plot

labels = list(category_counts.keys())

sizes = [v/len(submission)*100 for v in category_counts.values()]



plt.figure(figsize=(10,10))
```

```
# Plot
plt.pie(sizes, labels=labels, autopct='%1.1f%%', shadow=True, startangle=140)
plt.axis('equal')
plt.show()
import matplotlib.pyplot as plt
import pandas as pd

# Read the csv file
submission = pd.read_csv("submission.csv")

# Create a new column "toxicity_category" and categorize the comments
submission["toxicity_category"] = "Non-toxic"

# set the threshold
threshold = 0.3

# create a dictionary to keep the count of each category
category_counts =
{'toxic':0,'severe_toxic':0,'obscene':0,'threat':0,'insult':0,'identity_hate':0,'Non-toxic':0}

#iterate over all the categories
for col in ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']:
    submission.loc[submission[col] > threshold, "toxicity_category"] = col
    category_counts[col] = submission[submission["toxicity_category"] ==
col].shape[0]

# non-toxic comments
```

```
category_counts['Non-toxic'] = submission[submission["toxicity_category"] == "Non-toxic"].shape[0]


# Data for the chart

labels = list(category_counts.keys())

counts = list(category_counts.values())

plt.figure(figsize=(15,8))

# Create the bar chart

plt.bar(labels, counts)


# Add labels and title

plt.xlabel('Toxicity Category')

plt.ylabel('Number of Comments')

plt.title('Toxicity Categories Distribution')


# Show the chart

plt.show()

counts
```

In [2]:

```python
%%time

best_scores = []
for fold in tqdm(range(0,5)):

    # initializing the data
    p_train = train[train['kfold'] != fold].reset_index(drop = True)
    p_valid = train[train['kfold'] == fold].reset_index(drop = True)

    train_dataset = BertDataSet(p_train['clean_text'], p_train[['toxic', 'severe_toxic','obscene',
'threat', 'insult','identity_hate']])
    valid_dataset = BertDataSet(p_valid['clean_text'], p_valid[['toxic', 'severe_toxic','obscene',
'threat', 'insult','identity_hate']])

    train_dataloader = DataLoader(train_dataset, batch_size = train_batch, shuffle = True, num_worke
rs = 4, pin_memory = True)
    valid_dataloader = DataLoader(valid_dataset, batch_size = valid_batch, shuffle = False, num_work
ers = 4, pin_memory = True)

    model = transformers.BertForSequenceClassification.from_pretrained("/kaggle/input/bert-base-case
d", num_labels = 6)
    model.to(device)

    LR = 2e-5
    optimizer = AdamW(model.parameters(), LR,betas = (0.9, 0.999), weight_decay = 1e-2) # AdamW opti
mizer

    train_steps = int(len(p_train)/train_batch * epochs)
    num_steps = int(train_steps * 0.1)

    scheduler = get_linear_schedule_with_warmup(optimizer, num_steps, train_steps)

    best_score = 1000
    train_accs = []
    valid_accs = []
    train_losses = []
    valid_losses = []
    best_valid_probs = []
```

```python
    print("------------- Fold = " + str(fold) + "-------------")

    for epoch in tqdm(range(epochs)):
        print("------------- Epoch = " + str(epoch) + "-------------")

        train_loss, train_acc = training(train_dataloader, model, optimizer, scheduler)
        valid_loss, valid_acc, valid_probs = validating(valid_dataloader, model)

        train_losses.append(train_loss)
        train_accs.append(train_acc)
        valid_losses.append(valid_loss)
        valid_accs.append(valid_acc)

        print('train losses: %.4f' %(train_loss), 'train accuracy: %.3f' %(train_acc))
        print('valid losses: %.4f' %(valid_loss), 'valid accuracy: %.3f' %(valid_acc))

        if (valid_loss < best_score):

            best_score = valid_loss
            print("Found an improved model! :)")

            state = {'state_dict': model.state_dict(),
                     'optimizer_dict': optimizer.state_dict(),
                     'best_score':best_score
                    }

            torch.save(state, "model" + str(fold) + ".pth")
            best_valid_prob = valid_probs
            torch.cuda.memory_summary(device = None, abbreviated = False)
        else:
            pass


    best_scores.append(best_score)
    best_valid_probs.append(best_valid_prob)

    ##Plotting the result for each fold
    x = np.arange(epochs)
    fig, ax = plt.subplots(1, 2, figsize = (15,4))
    ax[0].plot(x, train_losses)
    ax[0].plot(x, valid_losses)
    ax[0].set_ylabel('Losses', weight = 'bold')
    ax[0].set_xlabel('Epochs')
    ax[0].grid(alpha = 0.3)
    ax[0].legend(labels = ['train losses', 'valid losses'])

    ax[1].plot(x, train_accs)
    ax[1].plot(x, valid_accs)
    ax[1].set_ylabel('Accuracy', weight = 'bold')
    ax[1].set_xlabel('Epochs')
    ax[1].legend(labels = ['train acc', 'valid acc'])

    ax[1].grid(alpha = 0.3)
    fig.suptitle('Fold = '+str(fold), weight = 'bold')
```

```
-------------- Fold = 4-------------

  0%|          | 0/5 [00:00<?, ?it/s]

-------------- Epoch = 0-------------
train losses: 0.3104 train accuracy: 0.759
valid losses: 0.2465 valid accuracy: 0.967
Found an improved model! :)

 20%|██        | 1/5 [00:24<01:36, 24.19s/it]

-------------- Epoch = 1-------------
train losses: 0.1587 train accuracy: 0.961
valid losses: 0.0799 valid accuracy: 0.971
Found an improved model! :)

 40%|████      | 2/5 [00:49<01:14, 24.84s/it]

-------------- Epoch = 2-------------
train losses: 0.1083 train accuracy: 0.968
valid losses: 0.0613 valid accuracy: 0.973
Found an improved model! :)

 60%|██████    | 3/5 [01:14<00:49, 24.98s/it]

-------------- Epoch = 3-------------
train losses: 0.0708 train accuracy: 0.976
valid losses: 0.0521 valid accuracy: 0.975
Found an improved model! :)

 80%|████████  | 4/5 [01:39<00:24, 24.82s/it]

-------------- Epoch = 4-------------
train losses: 0.0763 train accuracy: 0.979
valid losses: 0.0474 valid accuracy: 0.976
Found an improved model! :)

100%|██████████| 5/5 [02:03<00:00, 24.75s/it]
100%|██████████| 5/5 [10:27<00:00, 125.42s/it]

CPU times: user 8min 40s, sys: 44.3 s, total: 9min 24s
Wall time: 10min 27s
```

```
In [14]:
tweets_data = pd.read_csv('/kaggle/input/tweetdata/biden_tweets_clean.csv')
tweets_data.head()
```

Out[14]:

	Unnamed: 0	Handle	comment_text
0	0	@rifter741	Vote for Pedro rifter741 1sReplying to theviva...
1	1	@JackBurtonMercr	Jack Burton Mercer JackBurtonMercr 2sReplying ...
2	2	@GayNinja18	Gay Ninja GayNinja18 5sReplying to bennyjohnso...
3	3	@vtotheworld	THE VOICE THAT SPEAKETH vtotheworld 7sReplying...
4	4	@RaffaeleGianni4	Raffaele Giannini RaffaeleGianni4 8sReplying t...

```
In [15]:
tweets_data['clean_text'] = tweets_data['comment_text'].apply(str).apply(lambda x: clean_text(x))

tweets_data.head()
```

Out[15]:

	Unnamed: 0	Handle	comment_text	clean_text
0	0	@rifter741	Vote for Pedro rifter741 1sReplying to theviva...	Vote for Pedro to thevivafrei and POTUSThat ...
1	1	@JackBurtonMercr	Jack Burton Mercer JackBurtonMercr 2sReplying ...	Jack Burton Mercer JackBurtonMercr to GarettJ...
2	2	@GayNinja18	Gay Ninja GayNinja18 5sReplying to bennyjohnso...	Gay Ninja to bennyjohnsonTrump becomes speak...
3	3	@vtotheworld	THE VOICE THAT SPEAKETH vtotheworld 7sReplying...	THE VOICE THAT SPEAKETH vtotheworld to and ...
4	4	@RaffaeleGianni4	Raffaele Giannini RaffaeleGianni4 8sReplying t...	Raffaele Giannini to and QuirinaleLa deriva...

```
In [21]:
results = pd.DataFrame(preds)
submission = pd.concat([test,results], axis = 1).drop(['comment_text', 'clean_text'], axis = 1)
submission.rename(columns = { 0:'toxic', 1:'severe_toxic', 2:'obscene', 3:'threat', 4:'insult', 5:'identity_hate'}, inplace = True)
submission.to_csv("submission.csv", index = False)
```

```
In [22]:
s = pd.read_csv('/kaggle/working/submission.csv')
s
```

Out[22]:

	id	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	00001cee341fdb12	0.053325	0.023605	0.028230	0.021882	0.031558	0.023474
1	0000247867823ef7	0.060815	0.022539	0.030143	0.021191	0.032518	0.023213
2	00013b17ad220c46	0.066758	0.023039	0.033834	0.021811	0.035709	0.025448
3	00017563c3f7919a	0.409210	0.070394	0.218947	0.076033	0.228523	0.085567
4	00017695ad8997eb	0.214863	0.040439	0.112736	0.037952	0.106022	0.045674
...	...	...	...	...	...	...	...
1162	NaN	0.095324	0.024446	0.041691	0.023956	0.043475	0.027035
1163	NaN	0.202943	0.031845	0.085832	0.034258	0.091867	0.040923
1164	NaN	0.057404	0.022690	0.029418	0.020994	0.030697	0.023713
1165	NaN	0.044127	0.025736	0.026847	0.022977	0.028077	0.024827
1166	NaN	0.074523	0.023628	0.035724	0.022968	0.039004	0.025950

1167 rows × 7 columns

TESTING

The process involves testing the effectiveness of a language model in handling different toxicity levels using a single tweet as the prompt. The goal is to assess the model's performance in recognizing and responding to various degrees of offensive or harmful content. By providing the model with a tweet as input, evaluators can observe how well it identifies and handles different levels of toxicity, ranging from mild to severe. This evaluation is crucial in determining the model's ability to maintain respectful and appropriate language while engaging with users and generating content. Ultimately, the process aims to gauge the model's proficiency in addressing toxicity and guiding further improvements to enhance its overall effectiveness in communication and user interaction.

In addition to the prompt tweet, the evaluation process involves providing the language model with an unseen clean dataset consisting of tweets centered on a specific topic. This clean dataset serves as a test set to assess the model's effectiveness in generating appropriate responses and engaging in constructive conversations within the given context. By analyzing the model's output when interacting with the clean data, evaluators can measure its ability to maintain relevance, coherence, and accuracy while avoiding any toxic or offensive language. The use of clean data helps in determining whether the model can effectively adapt its responses to different topics while upholding a high standard of language quality. This comprehensive evaluation allows for a holistic assessment of the language model's capabilities, ensuring that it not only handles toxicity but also excels in generating meaningful and contextually appropriate content for a wide range of topics.

TESTING CODE

```
data=[('@test','What the hell')]
import csv
with open('yt_s3.csv', 'w', newline='', encoding='utf-8') as f:
    header = ['Handle', 'Text']
    writer = csv.writer(f)
    writer.writerow(header)
    writer.writerows(data)


yt_s3 = pd.read_csv('/kaggle/working/yt_s3.csv')
yt_s3['clean_text'] = yt_s3['Text'].apply(str).apply(lambda x: clean_text(x))


yt_s3.head()
test_batch = 32
test_dataset = BERTinferenceDataSet(yt_s3['clean_text'])
test_dataloader = DataLoader(test_dataset, batch_size = test_batch, shuffle = False,
num_workers = 4, pin_memory = True)
pthes = [os.path.join("../input/final-models",s) for s in os.listdir('../input/final-models')
if ".pth" in s]
pthes
#/kaggle/input/final-models
model = transformers.BertForSequenceClassification.from_pretrained("bert-base-
cased", num_labels = 6)
allpreds = predicting(test_dataloader, model, pthes)
preds = np.zeros((len(test_dataset),6))
for i in range(kfold):
    preds += allpreds[i]
```

preds = preds / kfold

results = pd.DataFrame(preds)

results

preds

```
In [26]:
data=[('@test','What the hell')]
```

```
In [27]:
import csv
with open('yt_s3.csv', 'w', newline='', encoding='utf-8') as f:
    header = ['Handle', 'Text']
    writer = csv.writer(f)
    writer.writerow(header)
    writer.writerows(data)

yt_s3 = pd.read_csv('/kaggle/working/yt_s3.csv')
yt_s3['clean_text'] = yt_s3['Text'].apply(str).apply(lambda x: clean_text(x))

yt_s3.head()
```

Out[27]:

	Handle	Text	clean_text
0	@test	What the hell	What the hell

```
In [28]:
test_batch = 32
test_dataset = BERTinferenceDataSet(yt_s3['clean_text'])
test_dataloader = DataLoader(test_dataset, batch_size = test_batch, shuffle = False, num_workers =
4, pin_memory = True)
pthes = [os.path.join("../input/final-models",s) for s in os.listdir('../input/final-models') if ".p
th" in s]
pthes
#/kaggle/input/final-models
model = transformers.BertForSequenceClassification.from_pretrained("bert-base-cased", num_labels =
6)
```

```
Some weights of the model checkpoint at bert-base-cased were not used when initializing BertForS
equenceClassification: ['cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transfor
m.dense.bias', 'cls.predictions.transform.dense.weight', 'cls.predictions.transform.LayerNorm.bi
as', 'cls.predictions.bias', 'cls.seq_relationship.bias', 'cls.seq_relationship.weight']
- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of
a model trained on another task or with another architecture (e.g. initializing a BertForSequenc
eClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint
of a model that you expect to be exactly identical (initializing a BertForSequenceClassification
model from a BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at
bert-base-cased and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions
and inference.
```

In [29]:
```
allpreds = predicting(test_dataloader, model, pthes)
preds = np.zeros((len(test_dataset),6))
for i in range(kfold):
    preds += allpreds[i]
preds = preds / kfold
```

In [30]:
```
results = pd.DataFrame(preds)
results
```

Out[30]:

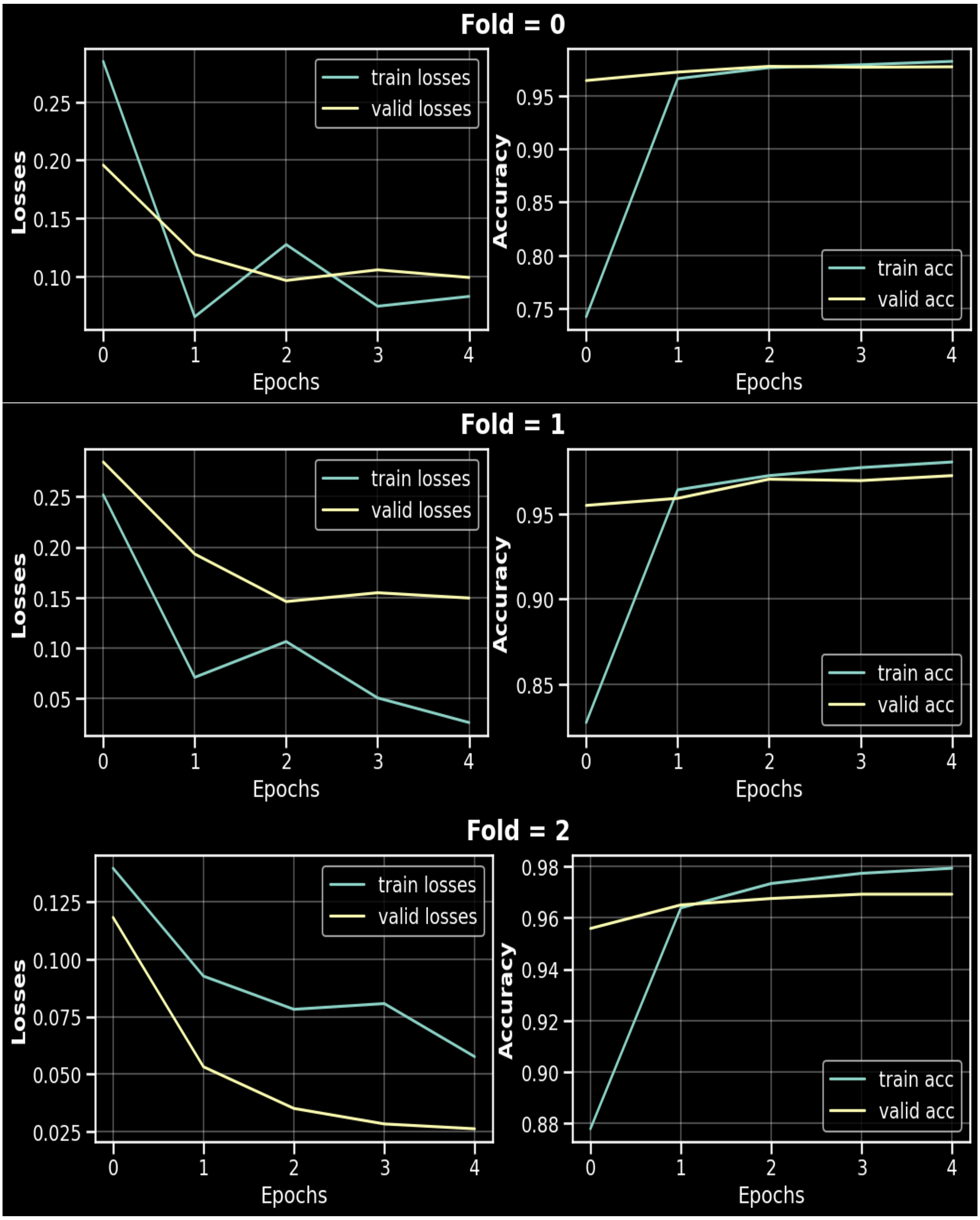
	0	1	2	3	4	5
0	0.560776	0.178306	0.433997	0.233033	0.443271	0.193188

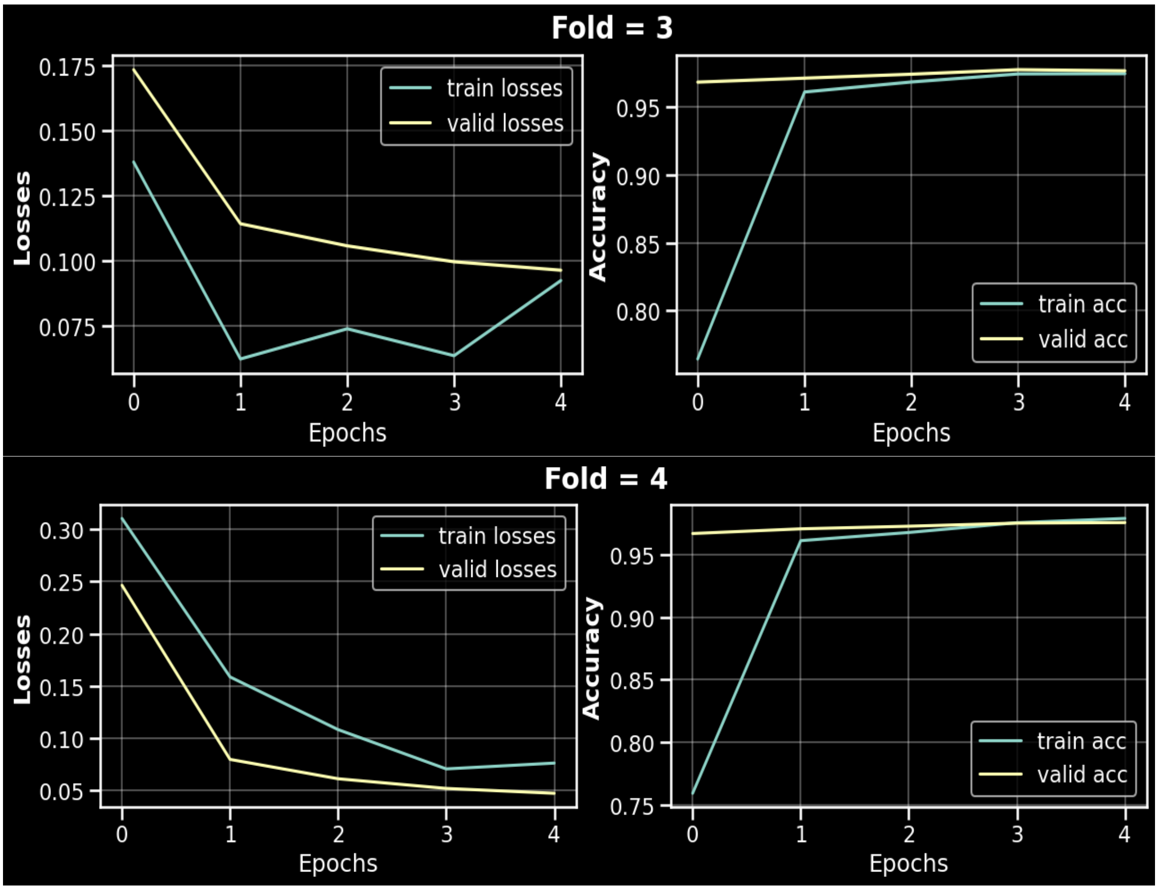
In [31]:
```
preds
```

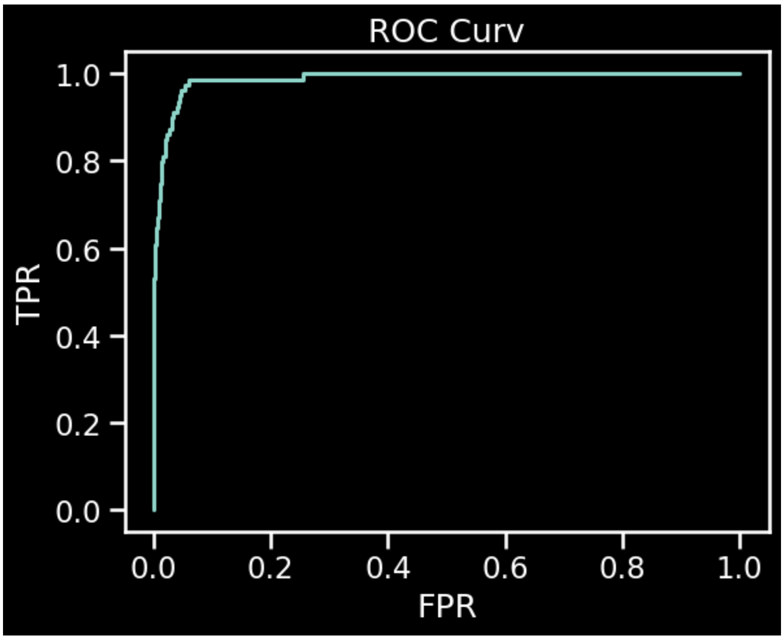
Out[31]:
```
array([[0.56077611, 0.17830615, 0.4339973 , 0.23303322, 0.44327111,
        0.19318834]])
```

RESULTS AND ANALYSIS

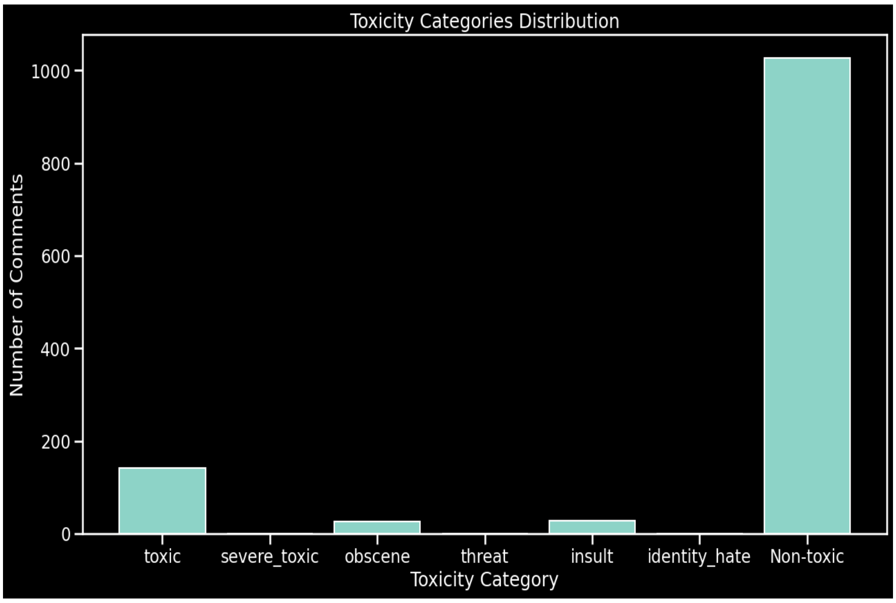
Loss Curves- From the given plots we can see as the number of folds increases throughout the training process the loss of the given model decreases and the accuracy of the given model increases over time, We achieve the best possible accuracy score of: 0.976 and the best possible loss score of: 0.0763.

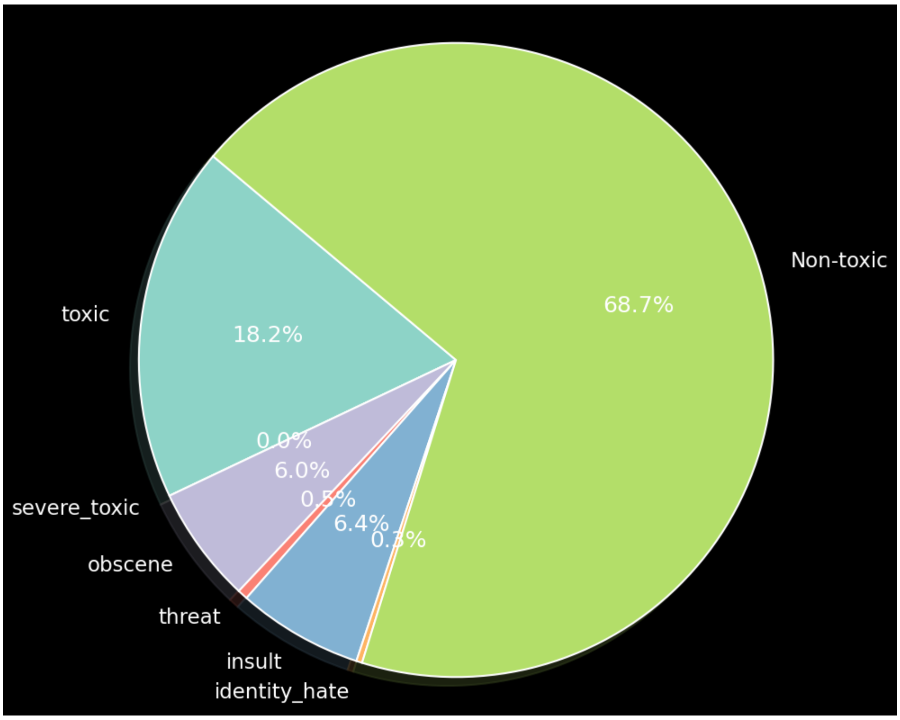






From the given ROC curve we can see that the performance of the model for the multi-classification model is on point.





The above two plots give us information about the various levels of toxicity for the given Tweet Dataset under the topic ‘Biden_Tweets’.

CONCLUSION

In this project, we focused on toxic comment classification of tweets, employing the powerful BERT model that achieved an impressive accuracy of 97.6%. The minimum loss value of 0.0763 showcased the model's robustness and efficiency in analyzing and classifying toxic comments from the vast collection of tweets.As we analyzed tweets with the keyword 'Biden' on Twitter, we found that a significant portion of 68.7% was non-toxic. However, a concerning 31.3% of tweets fell under various categories of toxicity, including toxic, obscene, threat, and insult. This highlights the prevalence of harmful content on social media platforms, warranting a dire need for effective moderation.

Moreover, our analysis brings to light the constant presence of racism, Islamophobia, and anti-Semitic sentiments rampant on the internet and social media websites like Twitter, Reddit, and Facebook. This perpetuates an unsafe and unwelcoming digital environment for users, hindering meaningful discussions and fostering hostility.

Furthermore, we recognize the pressing issue of racism, casteism, and violence against minority groups in India that often find a platform on social media. These harmful narratives can have severe real-life consequences, leading to mental health impacts and, tragically, even suicides.The necessity for stringent moderation of toxicity on social media platforms cannot be overstated. The ease of anonymous communication has led to a surge in toxic comments, posing significant threats to individuals' mental well-being and the harmony of online communities.

In light of these findings, social media platforms must take proactive measures in implementing robust content moderation systems. Employing advanced machine learning techniques like BERT, which has demonstrated exceptional accuracy in toxic comment classification, can significantly aid in identifying and removing harmful content.Public awareness and education campaigns are equally vital to foster a more empathetic and respectful online culture. Encouraging users to treat each other with kindness and respect can help reduce the prevalence of toxic behaviour.

Hannah Smith, a 14-year-old girl from the UK, tragically took her own life in 2013 after enduring relentless cyberbullying on the social media platform Ask.fm. The perpetrators bombarded her with abusive messages and threats, pushing her to the brink of despair. Similarly, Brandy Vela, an 18-year-old, faced a similar fate in 2016 due to online harassment and cyberbullying. She was targeted with cruel and derogatory messages on various social media platforms and messaging apps.

These heart-wrenching cases underscore the urgency to address the pervasive issue of toxicity on social media. The prevalence of harmful comments, cyberbullying, and harassment can have profound impacts on vulnerable individuals, leading to severe mental health repercussions and, tragically, even suicides.

In conclusion, our toxic comment classification project utilizing the BERT model revealed the extent of toxicity prevalent on social media platforms, with alarming implications for mental health and societal harmony. This necessitates a collective effort from technology companies, policymakers, and individuals alike to actively combat toxicity on social media, fostering a safer and more inclusive digital space for everyone.

FUTURE WORKS AND RECOMMENDATIONS

Enhanced Toxicity Classification Model: Continuously fine-tune and update the BERT model to improve its accuracy and ability to classify a broader range of toxic comments. This can be achieved by incorporating more labeled data from various sources to increase the model's understanding of different forms of toxicity.

Multilingual Toxic Comment Classification: Extend the project to classify toxic comments in multiple languages, as toxicity is not limited to one language or region. By incorporating multilingual data, the model can be adapted to address toxicity on a global scale.

Real-Time Moderation: Develop a real-time toxicity moderation system that can instantly detect and filter toxic comments as they are posted on social media platforms. This would prevent harmful content from spreading and minimize its impact on users.

Contextual Analysis: Enhance the classification model by considering the context in which comments are made. Understanding the tone, sentiment, and context of a conversation can help in identifying sarcasm or irony, ensuring more accurate classification.

User Profiling: Implement user profiling to identify habitual toxic commenters and address the issue at its source. By tracking patterns of toxic behavior, social media platforms can take proactive measures to prevent repeated harmful actions.

Collaborative Efforts: Collaborate with social media platforms to implement the classification model as an integrated feature for content moderation. Joint efforts can lead to a safer online environment for millions of users.

Mental Health Support and Collaboration with Mental Health Professionals: Introduce mental health resources and helplines within social media platforms to provide immediate support to individuals affected by toxicity. Collaborating with mental health professionals can help study the impact of online toxicity on mental well-being and devise effective strategies for intervention and support.

REFERENCES AND BIBLIOGRAPHY

[1] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805.

[2] Thain, Nithum; Dixon, Lucas; Wulczyn, Ellery (2017). Wikipedia Talk Labels: Toxicity. figshare. Dataset. https://doi.org/10.6084/m9.figshare.4563973.v2

[3] F, Farzaneh. (n.d.). BERT Model. Kaggle. Retrieved from https://www.kaggle.com/code/oceands/bert-model-for-dummies

[4] Alammar, J. (2018). The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning). Retrieved from https://jalammar.github.io/illustrated-bert

[5] Akhtar, Z. (n.d.). A Deep Learning Approach for Native Language Identification. https://iq.opengenus.org/native-language-identification-dl/

[6] Androcec, Darko. (2020). Machine learning methods for toxic comment classification: a systematic review. Acta Universitatis Sapientiae, Informatica. 12. 205-216. 10.2478/ausi-2020-0012.

[7] Aken, V., Risch, B., Kestrel, J., Löser, R., Alexander (2018). Challenges for toxic comment classification in-depth analysis.http://dx.doi.org/10.18653/v1/W18-5105.

[8] Georgakopoulos, S. V., Tasoulis, S. K., Vrahatis, A. G., & Plagianakos, V. P. (2018). Convolutional Neural Networks for Toxic Comment Classification. arXiv:1802.09957. Retrieved from https://arxiv.org/abs/1802.09957

[9] Wang, K., Yang, J., & Wu, H. (2021). A Survey of Toxic Comment Classification Methods. arXiv:2112.06412. Retrieved from https://arxiv.org/abs/2112.06412

[10] Li, H., Mao, W., & Liu, H. (2019). Toxic Comment Detection and Classification. Stanford University CS229 Project Report. Retrieved from https://cs229.stanford.edu/proj2019spr/report/71.pdf