Text Analytics

Name: Muljayan Jalangan RGU Number: 2236772

Question

1. You are required to identify 10 popular Sri Lankan news sources who have an active presence on Twitter. List these 10 twitter handles together with their follower and following counts, and the number of tweets each made during the past 12 months.
2. Extract all articles indexed by the twitter handles of these news sources and state the dimensions of the resulting article collection (NOT tweet collection) of all news agencies in terms of the total tokens and the unique tokens. State also the total tokens and unique tokens of each of the news agencies separately. In preparation for building a classifier of such news articles, address any potential imbalance in the dataset using at least two (02) different methods.
3. Use a sparse and a dense vector representation for extracting features for training a classifier for this dataset. Interpret the dimensions of the sparse vector and justify the dimensions of the dense vector used.
4. Train classifiers with the two (02) representations above using three (03) non-deep learning algorithms, stating your reasons for selecting each algorithm. Compare and contrast the performance of each of the classifiers.
5. Train also three (03) deep learning classifiers with distinct architectures using two (02) embedding techniques and one (01) contextual embedding technique, justifying the architectures you employ. Compare the performance of each of the models and interpret the results.

```
# Install dependencies
!pip install contractions # installs contractions package
import nltk
nltk.download("stopwords")
nltk.download("punkt")
nltk.download('wordnet')
!pip install transformers
```

```
Collecting contractions
  Downloading contractions-0.1.73-py2.py3-none-any.whl (8.7 kB)
Collecting textsearch>=0.0.21 (from contractions)
  Downloading textsearch-0.0.24-py2.py3-none-any.whl (7.6 kB)
Collecting anyascii (from textsearch>=0.0.21->contractions)
  Downloading anyascii-0.3.2-py3-none-any.whl (289 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 289.9/289.9 kB 6.3 MB/s eta 0:00:00
Collecting pyahocorasick (from textsearch>=0.0.21->contractions)
  Downloading pyahocorasick-2.0.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 110.8/110.8 kB 12.6 MB/s eta 0:00:00
Installing collected packages: pyahocorasick, anyascii, textsearch, contractions
Successfully installed anyascii-0.3.2 contractions-0.1.73 pyahocorasick-2.0.0 textsearch-0.0.24
Saved successfully!age stopwords to /root/nltk_data...
ora/stopwords.zip.
age punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
Collecting transformers
  Downloading transformers-4.31.0-py3-none-any.whl (7.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.4/7.4 MB 48.2 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.12.2)
Collecting huggingface-hub<1.0,>=0.14.1 (from transformers)
  Downloading huggingface_hub-0.16.4-py3-none-any.whl (268 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 268.8/268.8 kB 30.2 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.22.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (23.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2022.10.
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.27.1)
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1 (from transformers)
  Downloading tokenizers-0.13.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (7.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 112.2 MB/s eta 0:00:00
Collecting safetensors>=0.3.1 (from transformers)
  Downloading safetensors-0.3.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 74.0 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.65.0)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.14.1->trar
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hu
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transforr
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests->trans
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.4
Installing collected packages: tokenizers, safetensors, huggingface-hub, transformers
Successfully installed huggingface-hub-0.16.4 safetensors-0.3.1 tokenizers-0.13.3 transformers-4.31.0
```

The above dependencies are required to be installed to run the below code snippets.

1. Identifying popular Sri Lankan news sources

1. Newsfirst.lk Sri Lanka (@NewsfirstSL, 10 Following, 526.6K Followers)

1. Newsfirst.lk Sri Lanka (@NewsfirstSL, 10 Following, 526.6K Followers)
2. Ada Derana (@adaderana, 15 Following, 533.7K Followers)
3. DailyMirror (@Dailymirror_SL, 30 Following, 589.1K Followers)
4. NewsWire 🇱🇰 (@NewsWireLK, 2 Following, 223.4K Followers)
5. Colombo Gazette (@colombogazette, 114 Following, 25.4K Followers)
6. Ceylon Today (@CeylonToday, 33 Following, 63K Followers)
7. EconomyNext Sri Lanka (@Economynext, 333 Following, 10.3K Followers)
8. theisland.lk (@theisland_lk, 0 Following, 262 Followers)
9. Daily News (@DailyNews_lk, 96 Following, 14.8K Followers)
10. Hiru News (@hirunews, 48 Following, 1.6K Followers)

> Due to the new restrictions on Twitter API, we were unable to get the number of tweets made during the past 12 months. Therefore, I have resorted to other means of data fetching such as webscraping and using APIs of the news agencies. This the relevant code can be found in part 2.

2. Data extraction, Tokenization and Initial Analysis

> As mentioned above, Twitter has imposed several new restrictions on the API, we are unable to extract a substantial amount of tweets for article extraction. therefore we will be using webscraping techniques and also the APIs used by news websites to extract articles.

Assumptions

- We will be trying to fetch news items posted after 01/06/2022

2.1. Data Extraction

2.1.1 NewsFirst.lk

> I was able to find out that the NewsFirst website uses an API to fetch articles. This API can be used to retrieve articles.

Category Codes

> After inspecting the API url structure we can identify the following categories and codes
>
> - Features : 54387
>
> Saved successfully!
>
> - World/Foreign : 3
> - Sports : 4
> - Business: 5

```
import time
import requests
from datetime import datetime
import json

sleep_time = 5
page_size = 1000

category_codes = {
     'featured' : 54387,
     'local' : 2,
     'foreign' : 3,
     'sports' : 4,
     'business' : 5,
}

merged_data = []

for category_name, category_code in category_codes.items():
    print(f"Running for: {category_name} : {category_code}")

    date_range_reached = False
    page = 0

    while not date_range_reached:
        if page != 0:
            # Delay for <sleep_time> seconds
            time.sleep(sleep_time)
```

```
        print(f'Running for page {page+1} and category {category_name} ({category_code})')

        # API call
        endpoint = f'https://apienglish.newsfirst.lk/post/categoryPostPagination/{category_code}/{page}/{page_size}/'
        response = requests.get(endpoint)

        # Convert response to JSON
        data = response.json()

        # Extract the data we need
        data = data['postResponseDto']

        # Loop through the data and convert it to the desired format
        for val in data:
            date_gmt = datetime.strptime(val["date_gmt"], "%Y-%m-%dT%H:%M:%S.%fZ")

            # Ignore objects with dates earlier than June 2022
            if date_gmt >= datetime(2022, 6, 1):
                converted_val = {
                    "id": val["id"],
                    "date": val["date_gmt"],
                    "title": val["title"]["rendered"],
                    "content": val["content"]["rendered"],
                    "excerpt": val["excerpt"]["rendered"],
                    "postName": val["post_name"],
                    "short_title": val["short_title"],
                    "url": f"https://english.newsfirst.lk/{val['post_url']}",
                    "category": category_name,
                    "source": "newsfirst"
                }
                merged_data.append(converted_val)
            else:
                date_range_reached = True
                break

        # Process the response as needed
        print(f'Response #{page + 1}: {response.status_code}')
        page += 1

# Save data from response to a file
with open(f'newsfirst.json', 'w') as f:
    f.write(json.dumps(merged_data))

print('Data collection completed for all categories for newsfirst.')
```

```
Saved successfully!        r all categories for newsfirst.
```

2.1.2 Ada Derana

> We will be resorting to webscraping to get the news items from Ada Derana.

The structure of news items of Ada Derana is `https://adaderana.lk/news.php?nid=82795` and I have identified through manual means that the news items for June 1st 2022 start from id 82795.

```
import requests
import json
from bs4 import BeautifulSoup
import time
from datetime import datetime, timedelta

base_url = "https://adaderana.lk/news.php?nid={id}"
start_id = 82795
batch_size = 100
page = 1
articles = []


def parse_datetime(datetime_str):
    # Parse the date and time string into the desired format: "2022-06-01T07:42:04.000Z"
    datetime_obj = datetime.strptime(datetime_str, "%B %d, %Y   %I:%M %p")
    datetime_obj -= timedelta(hours=5, minutes=30)  # Convert to Sri Lankan time (GMT +5:30)
    formatted_datetime = datetime_obj.strftime("%Y-%m-%dT%H:%M:%S.000Z")
    return formatted_datetime


def scrape_article(article_id):
    url = base_url.format(id=article_id)
    # print(f"Scraping article #{article_id} from {url}")
    headers = {
```

```
        "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatic
        "Accept-Encoding": "gzip, deflate, br",
        "Accept-Language": "en-GB,en-US;q=0.9,en;q=0.8",
        "Cache-Control": "max-age=0",
        "Cookie": "",
        "Sec-Ch-Ua": '"Not.A/Brand";v="8", "Chromium";v="114", "Google Chrome";v="114"',
        "Sec-Ch-Ua-Mobile": "?0",
        "Sec-Ch-Ua-Platform": '"macOS"',
        "Sec-Fetch-Dest": "document",
        "Sec-Fetch-Mode": "navigate",
        "Sec-Fetch-Site": "none",
        "Sec-Fetch-User": "?1",
        "Upgrade-Insecure-Requests": "1",
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
    }
    try:
        response = requests.get(url, headers=headers)
        if response.status_code == 200:
            soup = BeautifulSoup(response.content, "html.parser")
            title = soup.select_one("h1").text.strip()
            datetime_str = soup.select_one(".news-datestamp").text.strip()
            content_element = soup.select_one(".news-content")
            content_html = str(content_element)
            if(content_html == "<div class=\"news-content\">\n</div>"):
                return False
            parsed_datetime = parse_datetime(datetime_str)
            article = {
                "id": article_id,
                "title": title,
                "datetime": parsed_datetime,
                "content": content_html,
                "url": url,
                "source": "adaderana"
            }
            articles.append(article)
            return True
    except Exception as e:
        print(f"Error scraping article #{article_id}: {str(e)}")
    return False

while True:
    if not scrape_article(start_id):
        print(f"Could not scrape article #{start_id}. Stopping.")
        print(f"Total articles scraped: {len(articles)}")
        break
    start_id += 1
```

Saved successfully!

```
# Save the articles as a JSON array
with open("datasets/adaderana.json", "w") as file:
    json.dump(articles, file)


print(f"Scraped {len(articles)} articles from Ada Derana.")
```

```
Scraped 8856 articles from Ada Derana.
```

2.1.3 Daily Mirror

> We will be resorting to webscraping to get the news items from Daily Mirror. The structure of news list is as follows
>
> - Features : https://www.dailymirror.lk/features/131
> - News : https://www.dailymirror.lk/news/209
> - Financial : https://www.dailymirror.lk/financial-news/265
> - Other : https://www.dailymirror.lk/Other/117
> - Sports : https://www.dailymirror.lk/sports
> - Expose : https://www.dailymirror.lk/expose/333

> Additionally the pagination works in multiples of 30. eg: https://www.dailymirror.lk/news/209/30 goes to page 2 of the News section

```
from datetime import datetime
import requests
from bs4 import BeautifulSoup
import json
import time
```

```
base_url = "https://www.dailymirror.lk"
categories = {
    "featured": "/features/131",
    "news": "/news/209",
    "financial": "/financial-news/265",
    "other": "/Other/117",
    "sports": "/sports",
    "expose": "/expose/333",
    "hardtalk": "/hard-talk/334",
    "business": "/business-news/273",
}

complete_article_list = []

# loop through the categories
for category_name, category_url in categories.items():
    print(f"Running for: {category_name} : {base_url}{category_url}")
    limit_reached = False
    loop_count = 0
    page_size = 30 # number of articles per page, this is fixed
    article_list = []

    while not limit_reached:
        url = f"{base_url}{category_url}/{loop_count * page_size}"
        print(f"Scraping page #{loop_count + 1} from {url}")
        time.sleep(1)
        response = requests.get(url)
        soup = BeautifulSoup(response.content, "html.parser")

        if(category_name != "sports"):
            main_div =  soup.find('div', id='breakingnewsads')
            articles = main_div.find_all('div', class_='lineg')
        else:
            main_div =  soup.find('div', class_='inleft')
            # Find all the div elements with class "lineg" within the main div
            articles = main_div.find_all('div', class_='row')




        # Check if the page has no articles
        if len(articles) == 0:
            limit_reached = True
            break
```



```
                          le.find('span', class_='gtime').text.strip()
            date_time = datetime.strptime(date_time_str, '%d %b %Y')
            date_time_iso = date_time.strftime("%Y-%m-%dT%H:%M:%S.000Z")
            if date_time <= datetime(2022, 6, 1):
                limit_reached = True
                break  # Break out of the loop

            title = article.find('h3', class_='cat-hd-tx').text.strip()
            excerpt = article.find_all('p')[1].text.strip()
            url = article.select_one("a")["href"]

            time.sleep(1)
            # ignore article if url has "https://www.dailymirror.lk/infographics"
            if "https://www.dailymirror.lk/infographics" in url:
                continue

            try:
                print(f"Scraping article #{url}")
                # get the full article content from the article url
                article_response = requests.get(url)
                article_soup = BeautifulSoup(article_response.content, "html.parser")
                article_content = article_soup.find('header', class_='inner-content').text.strip()
            except Exception as e:
                print(f"Error scraping article #{url}: {str(e)}")
                continue

            article_dict = {
                "title": title,
                "excerpt": excerpt,
                "date_time": date_time_iso,
                "url": url,
                "content": article_content,
                "category": category_name,
                "source": "daily_mirror"
```

```
            }
            article_list.append(article_dict)
            complete_article_list.append(article_dict)
        loop_count += 1
    print(f"Total articles scraped for {category_name}: {len(article_list)}")
    with open(f"datasets/daily_mirror/{category_name}.json", "w") as file:
        json.dump(article_list, file)


print(f"Total articles scraped: {len(article_list)}")

# Save the articles as a JSON array
with open("datasets/daily_mirror.json", "w") as file:
    json.dump(complete_article_list, file)

print("Data collection completed for all categories.")

# read datasets/daily_mirror.json and print number of articles collected
with open(f'datasets/daily_mirror.json', 'r') as f:
    daily_mirror_data = json.load(f);

print(f'Number of news items collected from daily mirror: {len(daily_mirror_data)}')
```

```
    Number of news items collected from daily mirror: 5903
```

2.1.4 NewsWire

> We will be resorting to webscraping to get the news items from News Wire. The structure of news list is as follows
>
> - Foreign : https://www.newswire.lk/category/international-news/page/1
> - News : https://www.newswire.lk/category/news/page/1
> - Business: https://www.newswire.lk/category/business/page/1
> - Sports : https://www.newswire.lk/category/sports/page/1
> - Education : https://www.newswire.lk/category/eduwire/page/1

```
from datetime import datetime
import requests
from bs4 import BeautifulSoup
import json


base_url = "https://www.newswire.lk/category/"
```

Saved successfully!

```
s",
    "news": "news",
    "business":  "business",
    "sports": "sports",
    "education": "eduwire",
}

# loop through the categories
for category_name, category_url in categories.items():
    limit_reached = False
    page = 1
    article_list = []

    while not limit_reached:
        url = f"{base_url}{category_url}/page/{page}"
        print(f"Scraping page #{page} for category {category_name} from {url}")
        response = requests.get(url)
        soup = BeautifulSoup(response.content, "html.parser")

        articles = soup.find_all('article')
        if len(articles) == 0:
            limit_reached = True
            break

        # article length is 1 and it has a div called "no-results" with content "Hello World" break
        if len(articles) == 1 and articles[0].find('h1', class_='entry-title').text.strip() == "Nothing found":
            limit_reached = True
            break


        for article in articles:
            date_time_str = article.find('time', class_='entry-published updated').text.strip()
            date_time = datetime.strptime(date_time_str, '%B %d, %Y')
            date_time_iso = date_time.strftime("%Y-%m-%dT%H:%M:%S.000Z")
```

```
            if date_time <= datetime(2022, 6, 1):
                limit_reached = True
                break

            title = article.find('h2', class_='entry-title').text.strip()
            excerpt = article.find('div', class_='entry-summary').find('p')
            if excerpt is not None:
                excerpt = excerpt.text.strip()
            else:
                excerpt = ""
            url = article.select_one("a")["href"]

            try:
                # fetch the full article content from the article url
                print(f"Scraping article #{url}")
                article_response = requests.get(url)
                article_soup = BeautifulSoup(article_response.content, "html.parser")
                article_content = article_soup.find('div', class_='entry-the-content').text.strip()
            except Exception as e:
                print(f"Error scraping article #{url}: {str(e)}")
                continue

            article_dict = {
                "title": title,
                "excerpt": excerpt,
                "date_time": date_time_iso,
                "url": url,
                "content": article_content,
                "category": category_name,
                "source": "newswire"
            }
            article_list.append(article_dict)
        page += 1
    print(f"Total articles scraped for {category_name}: {len(article_list)}")
    with open(f"datasets/newswire/{category_name}.json", "w") as file:
        json.dump(article_list, file)


#  get all json files in datasets/newswire/*.json and merge them into a single json file
import glob

all_files = glob.glob("datasets/newswire/*.json")
complete_article_list = []

for file in all_files:
    with open(file, "r") as f:
```



```
end(data)

print(f"Total articles scraped: {len(complete_article_list)}")

# Save the articles as a JSON array
with open("datasets/newswire.json", "w") as file:
    json.dump(complete_article_list, file)

print("Data collection completed for all categories.")
```

```
Total articles scraped: 8855
Data collection completed for all categories.
```

2.1.5 Colombo Gazette

We will be resorting to webscraping to get the news items from Colombo Gazette. The structure of news list is as follows

- news : https://colombogazette.com/category/news/page/1
- featured : https://colombogazette.com/category/feature/page/1
- special report : https://colombogazette.com/category/special-report/page/1
- opinion : https://colombogazette.com/category/oped/page/1
- lifestyle : https://colombogazette.com/category/life/
- advertorial : https://colombogazette.com/category/advertorial/page/1/

```
from datetime import datetime
import requests
from bs4 import BeautifulSoup
import json

base_url = "https://colombogazette.com/category/"
categories = {
    "news": "news",
```

```
        "featured": "feature",
        "special_report":  "special-report",
        "opinion": "oped",
        "lifestyle": "life",
        "advetorial": "advertorial",
    }

    # loop through the categories
    for category_name, category_url in categories.items():
        limit_reached = False
        page = 1
        article_list = []

        while not limit_reached:
            page_string = "" if page == 1 else f"page/{page}"
            url = f"{base_url}{category_url}/{page_string}"
            print(f"Scraping page #{page} for category {category_name} from {url}")
            response = requests.get(url)
            soup = BeautifulSoup(response.content, "html.parser")

            articles_container = soup.find('div', class_='td-ss-main-content')

            # create a match case for different categories
            match category_name:
                case "news":
                    articles = articles_container.find_all('div', class_='td_module_1 td_module_wrap td-animation-stack')
                case _:
                    print("No match")
                    articles = articles_container.find_all('div', class_='td_module_10 td_module_wrap td-animation-stack')

            if len(articles) == 0:
                limit_reached = True
                break


            for article in articles:
                date_time_str = article.find('time', class_='entry-date updated td-module-date').text.strip()
                date_time = datetime.strptime(date_time_str, '%B %d, %Y')
                date_time_iso = date_time.strftime("%Y-%m-%dT%H:%M:%S.000Z")

                if date_time <= datetime(2022, 6, 1):
                    limit_reached = True
                    break

                heading = article.find('h3', class_='entry-title').select_one("a")
                title = heading.text.strip()

   Saved successfully!                  d('div', class_='td-excerpt')

                if excerpt is not None:
                    excerpt = excerpt.text.strip()
                else:
                    excerpt = ""

                try:
                    # fetch the full article content from the article url
                    print(f"Scraping article #{url}")
                    article_response = requests.get(url)
                    article_soup = BeautifulSoup(article_response.content, "html.parser")
                    article_content = article_soup.find('div', class_='td-post-content').text.strip()
                except Exception as e:
                    print(f"Error scraping article #{url}: {str(e)}")
                    continue

                article_dict = {
                    "title": title,
                    "excerpt": excerpt,
                    "date_time": date_time_iso,
                    "url": url,
                    "content": article_content,
                    "category": category_name,
                    "source": "colombo_gazzete"
                }
                article_list.append(article_dict)
            page += 1
        print(f"Total articles scraped for {category_name}: {len(article_list)}")
        with open(f"datasets/colombo_gazzete/{category_name}.json", "w") as file:
            json.dump(article_list, file)


    #  get all json files in datasets/colombo_gazzete/*.json and merge them into a single json file
    import glob
```

```
all_files = glob.glob("datasets/colombo_gazzete/*.json")
complete_article_list = []

for file in all_files:
    with open(file, "r") as f:
        data = json.load(f)
        complete_article_list.extend(data)

print(f"Total articles scraped: {len(complete_article_list)}")

# Save the articles as a JSON array
with open("datasets/colombo_gazzete.json", "w") as file:
    json.dump(complete_article_list, file)

print("Data collection completed for all categories.")
```

2.1.6 Ceylon Today

> We will be resorting to webscraping to get the news items from Ceylon today. The structure of news list is as follows
>
> - local: https://ceylontoday.lk/category/local/page/1
> - world: https://ceylontoday.lk/category/world/page/1
> - sports: https://ceylontoday.lk/category/sports/page/1
> - business: https://ceylontoday.lk/category/business/page/1
> - entertainment: https://ceylontoday.lk/category/entertainment/page/1
> - tech: https://ceylontoday.lk/category/tech/page/1

```
import glob
from datetime import datetime
import requests
from bs4 import BeautifulSoup
import json

base_url = "https://ceylontoday.lk/category/"
categories = {
    "local": "local",
    "world": "world",
    "sports": "sports",
    "business": "business",
    "entertainment": "entertainment",
    "tech": "tech"
```

Saved successfully!

```
# loop through the categories
for category_name, category_url in categories.items():
    limit_reached = False
    page = 1
    article_list = []

    while not limit_reached:
        page_string = "" if page == 1 else f"page/{page}"
        url = f"{base_url}{category_url}/{page_string}"
        print(f"Scraping page #{page} for category {category_name} from {url}")
        response = requests.get(url)
        soup = BeautifulSoup(response.content, "html.parser")

        articles = soup.find_all(
            'div', class_='tdb_module_loop td_module_wrap td-animation-stack')

        if len(articles) == 0:
            limit_reached = True
            break

        for article in articles:
            heading = article.find(
                'h3', class_='entry-title td-module-title').select_one("a")
            title = heading.text.strip()
            url = heading["href"]

            try:
                # fetch the full article content from the article url
                print(f"Scraping article #{url}")
                article_response = requests.get(url)
                article_soup = BeautifulSoup(
                    article_response.content, "html.parser")
                date_time_str = article.find(
                    'time', class_='entry-date updated td-module-date').text.strip()
```

```
                date_time = datetime.strptime(
                    date_time_str, "%B %d, %Y %I:%M%p")
                date_time_iso = date_time.strftime("%Y-%m-%dT%H:%M:%S.000Z")

                if date_time <= datetime(2022, 6, 1):
                    limit_reached = True
                    break
                article_content = article_soup.find(
                    'div', class_='td_block_wrap tdb_single_content tdi_108 td-pb-border-top td_block_template_1 td-post-conte

                if article_content is not None:
                    article_content = article_content.text.strip()
                else:
                    article_content = article_soup.find(
                        'div', class_='td-post-content tagdiv-type').text.strip()

            except Exception as e:
                print(f"Error scraping article #{url}: {str(e)}")
                continue

            article_dict = {
                "title": title,
                "date_time": date_time_iso,
                "url": url,
                "content": article_content,
                "category": category_name,
                "source": "ceylon_today"
            }
            article_list.append(article_dict)
            with open(f"datasets/ceylon_today/{category_name}_w.json", "w") as file:
                json.dump(article_list, file)
        page += 1
    print(f"Total articles scraped for {category_name}: {len(article_list)}")
    with open(f"datasets/ceylon_today/{category_name}.json", "w") as file:
        json.dump(article_list, file)


#  get all json files in datasets/ceylon_today/*.json and merge them into a single json file

all_files = glob.glob("datasets/ceylon_today/*.json")
complete_article_list = []

for file in all_files:
    with open(file, "r") as f:
        data = json.load(f)
        complete_article_list.extend(data)
Saved successfully!                len(complete_article_list)}")

# Save the articles as a JSON array
with open("datasets/ceylon_today.json", "w") as file:
    json.dump(complete_article_list, file)

print("Data collection completed for all categories.")
```

```
Total articles scraped: 2479
Data collection completed for all categories.
```

2.1.7 Economy Next

> We will be resorting to webscraping to get the news items from Economy Next. All news items can be obtained from the folllowing url structure
>
> - https://economynext.com/more-news/page/1

```
import glob
from datetime import datetime
import requests
from bs4 import BeautifulSoup
import json

base_url = "https://economynext.com/more-news/"

limit_reached = False
page = 1
article_list = []

while not limit_reached:
    page_string = "" if page == 1 else f"page/{page}"
```

```
        url = f"{base_url}{page_string}"
        print(f"Scraping page #{page} from {url}")
        headers = {
            "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatic
            "Accept-Encoding": "gzip, deflate, br",
            "Accept-Language": "en-GB,en-US;q=0.9,en;q=0.8",
            "Cache-Control": "max-age=0",
            "Cookie": "",
            "Sec-Ch-Ua": '"Not.A/Brand";v="8", "Chromium";v="114", "Google Chrome";v="114"',
            "Sec-Ch-Ua-Mobile": "?0",
            "Sec-Ch-Ua-Platform": '"macOS"',
            "Sec-Fetch-Dest": "document",
            "Sec-Fetch-Mode": "navigate",
            "Sec-Fetch-Site": "none",
            "Sec-Fetch-User": "?1",
            "Upgrade-Insecure-Requests": "1",
            "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0
        }
        response = requests.get(url, headers=headers)
        soup = BeautifulSoup(response.content, "html.parser")
        articles = soup.find_all('div', class_='story-grid-single-story')

        if len(articles) == 0:
            limit_reached = True
            break

        for article in articles:
            category_name = article.find(
                'div', class_='main-category all-caps').select_one("a").text.strip().lower().replace(" ", "_")
            heading = article.find(
                'h3', class_='recent-top-header font-size-story-grid').select_one("a")
            title = heading.text.strip()
            content_url = heading["href"]

            date_time_str = article.find(
                "span", class_="article-publish-date").text.strip()
            print("date_time_str", date_time_str)
            date_time = datetime.strptime(
                date_time_str, "%B %d, %Y")
            date_time_iso = date_time.strftime("%Y-%m-%dT%H:%M:%S.000Z")

            if date_time <= datetime(2022, 6, 1):
                limit_reached = True
                break

            print(f"Scraping article #{content_url}")
                                      ts.get(content_url, headers=headers)
                                      oup(
                                      ent, "html.parser")
            # story-page-text-content
            article_content = article_soup.find(
                'div', class_='story-page-text-content').text.strip()

            article_dict = {
                "title": title,
                "date_time": date_time_iso,
                "url": content_url,
                "content": article_content,
                "category": category_name,
                "source": "economy_next"
            }
            article_list.append(article_dict)
            with open(f"datasets/economy_next/economy_next_w.json", "w") as file:
                json.dump(article_list, file)
        page += 1

    with open(f"datasets/economy_next.json", "w") as file:
        json.dump(article_list, file)

    complete_article_list = []

    with open("datasets/economy_next.json", "r") as f:
        data = json.load(f)
        complete_article_list.extend(data)


    print(f"Total articles scraped: {len(complete_article_list)} for economy_next")
```

```
Total articles scraped: 4188 for economy_next
```

Saved successfully!

2.1.8 The Island

> We will be resorting to webscraping to get the news items from The Island. The structure of news list is as follows
>
> - news: https://island.lk/category/news/page/1
> - featured: https://island.lk/category/features/page/1
> - sports: https://island.lk/category/sports/page/1
> - business: https://island.lk/category/business/page/1
> - opinion: https://island.lk/category/opinion/page/1
> - fashion: https://island.lk/category/fashion/page/1
> - politics: https://island.lk/category/politics/page/1

```
import glob
from datetime import datetime
import requests
from bs4 import BeautifulSoup
import json

base_url = "https://island.lk/category/"
categories = {
    "news": "news",
    "featured": "features",
    "sports": "sports",
    "business": "business",
    "opinion": "opinion",
    "fashion": "fashion",
    "politics": "politics",
}

# loop through the categories
for category_name, category_url in categories.items():
    limit_reached = False
    page = 1
    article_list = []

    while not limit_reached:
        page_string = "" if page == 1 else f"page/{page}"
        url = f"{base_url}{category_url}/{page_string}"
        print(f"Scraping page #{page} for category {category_name} from {url}")
        response = requests.get(url)
        soup = BeautifulSoup(response.content, "html.parser")

        articles = soup.find_all(
            'li', class_='mvp-blog-story-wrap')
```



```
            break

        for article in articles:
            title = article.find('h2').text.strip()
            url = article.select_one("a")["href"]

            try:
                # fetch the full article content from the article url
                print(f"Scraping article #{url}")
                article_response = requests.get(url)
                article_soup = BeautifulSoup(
                    article_response.content, "html.parser")
                article_content = article_soup.find(
                    'div', id='mvp-content-main').text.strip()

                date_container = article_soup.find(
                    'div', class_='mvp-author-info-wrap left relative')
                date_time_str = date_container.find('time').text.strip()
                date_time = datetime.strptime(date_time_str, "%Y/%m/%d")
                date_time_iso = date_time.strftime("%Y-%m-%dT%H:%M:%S.000Z")


                if date_time <= datetime(2022, 6, 1):
                    limit_reached = True
                    break

            except Exception as e:
                print(f"Error scraping article #{url}: {str(e)}")
                continue

            article_dict = {
                "title": title,
                "date_time": date_time_iso,
                "url": url,
```

```
                "content": article_content,
                "category": category_name,
                "source": "the_island"
            }
            article_list.append(article_dict)
            with open(f"datasets/the_island/{category_name}_w.json", "w") as file:
                json.dump(article_list, file)
        page += 1
    print(f"Total articles scraped for {category_name}: {len(article_list)}")
    with open(f"datasets/the_island/{category_name}.json", "w") as file:
        json.dump(article_list, file)


#  get all json files in datasets/the_island/*.json and merge them into a single json file

all_files = glob.glob("datasets/the_island/*.json")
complete_article_list = []

for file in all_files:
    with open(file, "r") as f:
        data = json.load(f)
        complete_article_list.extend(data)

print(f"Total articles scraped: {len(complete_article_list)}")

# Save the articles as a JSON array
with open("datasets/the_island.json", "w") as file:
    json.dump(complete_article_list, file)

print("Data collection completed for all categories.")
```

2.1.9 Daily News

> We will be resorting to webscraping to get the news items from Daily News. The structure of news list is as follows
>
> - local: https://island.lk/category/local/page/1
> - politics: https://island.lk/category/politics/page/1
> - entertainment: https://island.lk/category/entertainment/page/1
> - sports: https://island.lk/category/sports/page/1
> - business: https://island.lk/category/business/page/1
> - featured: https://island.lk/category/features/page/1


```
from bs4 import BeautifulSoup
import json

base_url = "https://www.dailynews.lk/category/"
categories = {
    "local": "local",
    "politics": "politics",
    "entertainment": "entertainment",
    "sports": "sports",
    "business": "business",
    "featured": "featured",
}

source = "daily_news"
dataset_path = f"datasets/{source}"
next_path_url="page/"


# loop through the categories
for category_name, category_url in categories.items():
    limit_reached = False
    page = 1
    article_list = []

    while not limit_reached:
        page_string = "" if page == 1 else f"{next_path_url}{page}"
        url = f"{base_url}{category_url}/{page_string}"
        print(f"Scraping page #{page} for category {category_name} from {url}")
        response = requests.get(url)
        soup = BeautifulSoup(response.content, "html.parser")

        main_element = soup.select_one("#main > div > ul")

        if main_element is None:
```

```
          limit_reached = True
          break


        # Find all the articles within the main element
        articles = main_element.find_all('article')


        if len(articles) == 0:
            limit_reached = True
            break

        for article in articles:
            title_element = article.find('h2', class_='penci-entry-title')
            title = title_element.text.strip()
            url = title_element.a['href']
            date_element = article.find('time', class_='entry-date')
            date_time_str = date_element['datetime']
            date_time = datetime.strptime(date_time_str, "%Y-%m-%dT%H:%M:%S%z").replace(tzinfo=None)
            date_time_iso = date_time.strftime("%Y-%m-%dT%H:%M:%S.000Z")


            try:
                # fetch the full article content from the article url
                print(f"Scraping article #{url}")
                article_response = requests.get(url)
                article_soup = BeautifulSoup(
                    article_response.content, "html.parser")
                article_content = article_soup.select_one("article div.post-entry").text.strip()

                if date_time <= datetime(2022, 6, 1):
                    print("Reached limited")
                    limit_reached = True
                    break

            except Exception as e:
                print(f"Error scraping article #{url}: {str(e)}")
                continue

            article_dict = {
                "title": title,
                "date_time": date_time_iso,
                "url": url,
                "content": article_content,
                "category": category_name,
                "source": source
Saved successfully!                   rticle_dict)
                            _path}/{category_name}_w.json", "w") as file:
                json.dump(article_list, file)
        page += 1
    print(f"Total articles scraped for {category_name}: {len(article_list)}")
    with open(f"{dataset_path}/{category_name}.json", "w") as file:
        json.dump(article_list, file)


#  get all json files in datasets/daily_news/*.json and merge them into a single json file

all_files = glob.glob(f"{dataset_path}/*.json")
complete_article_list = []

for file in all_files:
    with open(file, "r") as f:
        data = json.load(f)
        complete_article_list.extend(data)

print(f"Total articles scraped: {len(complete_article_list)}")

# Save the articles as a JSON array
with open(f"{dataset_path}.json", "w") as file:
    json.dump(complete_article_list, file)

print("Data collection completed for all categories.")
```

```
Data collection completed for all categories.
```

2.1.10 Hiru News

We will be resorting to webscraping to get the news items from Hiru news. The structure of news list is as follows

- local: https://www.hirunews.lk/english/local-news.php?pageID=1

> - world: https://www.hirunews.lk/english/international-news.php?pageID=1
> - entertainment: https://www.hirunews.lk/english/entertainment?pageID=1
> - business: https://www.hirunews.lk/english/business?pageID=1
> - sports: https://www.hirunews.lk/english/sports?pageID=1

```
import glob
from datetime import datetime
import requests
from bs4 import BeautifulSoup
import json

base_url = "https://www.hirunews.lk/english/"
categories = {
    "local": "local-news.php",
    "world": "international-news.php",
    "entertainment": "entertainment",
    "business": "business",
    "sports": "sports",

}

source = "hiru_news"
dataset_path = f"datasets/{source}"
next_path_url="?pageID="


# loop through the categories
for category_name, category_url in categories.items():
    limit_reached = False
    page = 1
    article_list = []

    while not limit_reached:
        page_string = ".php" if page == 1 else f"{next_path_url}{page}"
        url = f"{base_url}{category_url}{page_string}"
        print(f"Scraping page #{page} for category {category_name} from {url}")
        response = requests.get(url)
        soup = BeautifulSoup(response.content, "html.parser")

        main_element = soup.select_one(".trending-section")

        # Find all the articles within the main element
        articles = main_element.find_all('div', class_='row')

        print(f"Found {len(articles)} articles")
```



```
            limit_reached = True
            break

        for article in articles:
            title_element = article.find('div', class_='all-section-tittle').find('a')
            title = title_element.text.strip()
            url = title_element['href']
            date_element = article.find('div', class_='middle-tittle-time')
            date_time_str = date_element.text.strip()
            date_time = datetime.strptime(date_time_str, '%A, %d %B %Y - %H:%M').replace(tzinfo=None)
            date_time_iso = date_time.strftime("%Y-%m-%dT%H:%M:%S.000Z")

            try:
                # fetch the full article content from the article url
                print(f"Scraping article #{url}")
                article_response = requests.get(url)
                article_soup = BeautifulSoup(
                    article_response.content, "html.parser")
                article_content = article_soup.select_one("#article-phara2").text.strip()

                if date_time <= datetime(2022, 6,30):
                    print("Reached limited")
                    limit_reached = True
                    break

            except Exception as e:
                print(f"Error scraping article #{url}: {str(e)}")
                continue

            article_dict = {
                "title": title,
                "date_time": date_time_iso,
                "url": url,
                "content": article_content,
```

```
                "category": category_name,
                "source": source
            }
            article_list.append(article_dict)
            with open(f"{dataset_path}/{category_name}_w.json", "w") as file:
                json.dump(article_list, file)
        page += 1
    print(f"Total articles scraped for {category_name}: {len(article_list)}")
    with open(f"{dataset_path}/{category_name}.json", "w") as file:
        json.dump(article_list, file)


#  get all json files in datasets/the_island/*.json and merge them into a single json file

all_files = glob.glob(f"{dataset_path}/*.json")
complete_article_list = []

for file in all_files:
    with open(file, "r") as f:
        data = json.load(f)
        complete_article_list.extend(data)

print(f"Total articles scraped: {len(complete_article_list)}")

# Save the articles as a JSON array
with open(f"{dataset_path}.json", "w") as file:
    json.dump(complete_article_list, file)

print("Data collection completed for all categories.")
```

```
import glob
import json

all_files = glob.glob("datasets/hiru_news/*.json")
complete_article_list = []

for file in all_files:
    with open(file, "r") as f:
        data = json.load(f)
        complete_article_list.extend(data)

print(f"Total articles scraped: {len(complete_article_list)}")

# Save the articles as a JSON array
with open("datasets/hiru_news.json", "w") as file:
    json.dump(complete_article_list, file)

for all categories.")
```

Saved successfully!

2.2 Preprocessing and Analysis

Loading up the data set

```
import glob
import json
import pandas as pd

# Initialize variables
fetched_data = {}
total_article_length = 0

# List to store data for DataFrame
article_count_list = []

# Iterate through JSON files
file_paths = glob.glob("datasets/*.json")
for file_path in file_paths:
    with open(file_path, 'r') as file:
        # Extract agency name from the file path
        agency_name = file_path.split("/")[-1].split(".")[0]
        news_data = json.load(file)

        # Store data for DataFrame
        article_count_list.append({
            "agency": agency_name,
            "article_count": len(news_data)
        })

        fetched_data[agency_name] = news_data
```

```
        fetched_data[agency_name] = news_data
        total_article_length += len(news_data)

# Calculate average articles per agency
num_agencies = len(fetched_data)
average_articles_per_agency = total_article_length // num_agencies
article_count_list.append({
    "agency": "Total",
    "article_count": total_article_length
})

print(f"Average articles per agency: {average_articles_per_agency}")

# Create DataFrame directly from the list of dictionaries
article_count_df = pd.DataFrame(article_count_list)
article_count_df
```

```
Average articles per agency: 5518
```

	agency	article_count
0	newsfirst	7320
1	daily_news	1748
2	the_island	6672
3	ceylon_today	2479
4	colombo_gazzete	3437
5	economy_next	4188
6	daily_mirror	5903
7	adaderana	9184
8	newswire	8856
9	hiru_news	5394
10	Total	55181

The above code snippet fetches all json files in the data sets folder. This is done this way as the previous scraping scripts fetch the data and store them in a json file in the formal `<agencyname>.json`.

We also do some preliminary calculations and add it to a dataframe to understand the quantity of data sets.

Addressing class imbalance

Saved successfully!

ere is a class imbalance. We will be using the following techniques to handle the class imbalance

- Oversampling
- Undersampling

We will have the average articles per agency as our target and randomly drop articles from agencies that have more articles than the average. We will also randomly duplicate articles from agencies that have less articles than the average.

```
import random

# articles per agency

# articles_count_to_be_considered = average_articles_per_agency
articles_count_to_be_considered = 1200

print(f"We will be considering {articles_count_to_be_considered} articles per agency")

# Calculate the number of articles to be added/removed for each agency
diff_count = {agency: articles_count_to_be_considered - len(articles) for agency, articles in fetched_data.items()}

print("Difference count:", diff_count)

# Undersample and Oversample
for agency, articles in fetched_data.items():
    if diff_count[agency] > 0:  # Oversample
        while len(articles) < articles_count_to_be_considered:
            random_article = random.choice(articles)
            articles.append(random_article)
    elif diff_count[agency] < 0:  # Undersample
        random.shuffle(articles)
        fetched_data[agency] = articles[:articles_count_to_be_considered]

# Verify the result
resulting_total_articles = sum(len(articles) for articles in fetched_data.values())
```

```
print("Number of agencies:", num_agencies)
print("Total number of articles:", resulting_total_articles)
print(f"Verified: {resulting_total_articles == num_agencies * articles_count_to_be_considered}")
```

```
We will be considering 1200 articles per agency
Difference count: {'newsfirst': -6120, 'daily_news': -548, 'the_island': -5472, 'ceylon_today': -1279, 'colombo_gazzete':
Number of agencies: 10
Total number of articles: 12000
Verified: True
```

This code aims to balance the number of articles across multiple agencies by either oversampling or undersampling their articles. It starts by calculating the difference in the number of articles each agency currently has compared to the desired `articles_count_to_be_considered`. This value was initially equalent to `average_articles_per_agency` but was reduced due to the massive data volume that caused performance issues. The `diff_count` dictionary stores these differences. Then, it proceeds to perform the balancing process:

For agencies with a positive difference (meaning they have fewer articles than the average), it randomly selects articles from their existing collection and duplicates them until the average is reached (oversampling).

On the other hand, for agencies with a negative difference (more articles than the average), it shuffles their articles and keeps only a subset of articles equal to the average (undersampling). Finally, the code verifies the result by calculating the total number of articles across all agencies and checks if it matches the expected value (i.e., `num_agencies * average_articles_per_agency`).

Due to restrictions from resources on google colab, the number of articles had to be narrowed down to 1200 per agency. This causes significant impact to the accuracy of the models.

```
import re
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
import contractions


def remove_html_markup(text):
    # Remove HTML markup
    text = re.sub('<.*?>', '', text)
    return text


def preprocess_and_tokenize(text):
    # Remove HTML markup
    text = remove_html_markup(text)

    # Expand contractions
    text = contractions.fix(text)

    # Remove non-alphanumeric and numeric characters
    text = re.sub(r'[^a-zA-Z]', ' ', text)

    # Tokenize the text
    tokens = word_tokenize(text)

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]

    # Lemmatize the tokens
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    return tokens


all_articles = []
all_tokens = []
agency_stats = []

# loop through the fetched_data dictionary and preprocess the content column
for agency_name, agency_data in fetched_data.items():
    print(f"Preprocessing data for {agency_name}")
    # change structure of fetched_data dictionary to have articles in another dictionary
    fetched_data[agency_name] = {
        "articles": agency_data,
        "tokenized_articles": [],
        "unique_tokens": [],
        "total_tokens": 0,
```

```
    }
    for article in agency_data:
        tokenized_content = preprocess_and_tokenize(article["content"])
        article["tokenized_content"] = tokenized_content
        article["unique_tokens"] = list(set(tokenized_content))
        all_articles.append(article)

        fetched_data[agency_name]["tokenized_articles"].append(
            tokenized_content)
        fetched_data[agency_name]["total_tokens"] += len(tokenized_content)

        unique_tokens_for_agency = tokenized_content + \
            fetched_data[agency_name]["unique_tokens"]
        fetched_data[agency_name]["unique_tokens"] = list(
            set(unique_tokens_for_agency))

        all_tokens.extend(tokenized_content)
    agency_stats.append({
        'agency': agency_name,
        'articles': len(agency_data),
        'tokens': fetched_data[agency_name]["total_tokens"],
        'unique_tokens': len(
            fetched_data[agency_name]["unique_tokens"])})

total_unique_tokens = list(set(all_tokens))

agency_stats.append({
    'agency': "all",
    'articles': len(all_articles),
    'tokens': len(all_tokens),
    'unique_tokens': len(total_unique_tokens)
})

# create a dataframe from the agency_stats

agency_stats_df = pd.DataFrame(agency_stats)

agency_stats_df
```

```
Preprocessing data for newsfirst
Preprocessing data for daily_news
Preprocessing data for the_island
Preprocessing data for ceylon_today
Preprocessing data for colombo_gazzete
Preprocessing data for economy_next
Preprocessing data for daily_mirror
Preprocessing data for adaderana
ire
news
```

Saved successfully!

	agency	articles	tokens	unique_tokens
0	newsfirst	1200	154776	13630
1	daily_news	1200	243733	16767
2	the_island	1200	406937	30931
3	ceylon_today	1200	107136	12339
4	colombo_gazzete	1200	229146	16228
5	economy_next	1200	255916	13886
6	daily_mirror	1200	311771	21368
7	adaderana	1200	172537	14692
8	newswire	1200	165250	15864
9	hiru_news	1200	135847	13851
10	all	12000	2183049	55807

The above code is a text preprocessing step.

We define a set of functions and importing the necessary libraries, including the NLTK toolkit for natural language processing.

The `remove_html_markup` function is used to strip HTML tags from the text.

The main function `preprocess_and_tokenize` processes the input text by converting it to lowercase, expanding contractions, removing non-alphanumeric and numeric characters, tokenizing the text, removing stopwords, and lemmatizing the tokens. Note that we are using an existing `contractions` library to ensure that we don't have to manually add a contractions file

We proceed to process data from the `fetched_data` dictionary, which contains articles from different agencies. It iterates through the dictionary, applies the preprocessing function to the content column of each article, and creates additional statistics for each agency, such as the total number of tokens, unique tokens, and tokenized articles. The processed articles are collected into the `all_articles` list, and the tokens are aggregated into the `all_tokens` list.

Finally we create a summary of the processed data in the `agency_stats` list, which contains information about each agency's number of articles, total tokens, and unique tokens. Additionally, a summary row for all agencies combined is added to the `agency_stats` list.

We use a pandas DataFrame `agency_stats_df` created from `agency_stats` to display the aggregated information neatly in tabular form.

```
# Add the agency data into a dataframe
import pandas as pd

# df of all articles with only title, source, url tokenized_content, unique_tokens
df = pd.DataFrame(all_articles, columns=[
                  "title", "source", "url", "content", "tokenized_content", "unique_tokens"])

# print(len(df))
# df.head()
# df.tail(10)
# # print columns
df.columns
```

```
Index(['title', 'source', 'url', 'content', 'tokenized_content',
       'unique_tokens'],
      dtype='object')
```

In the above code we create a dataframe from the all articles json array

```
# List of variables you want to keep
variables_to_keep = ['df']

# Get a dictionary of all variables in the current session
all_variables = %who_ls

# Variables to be cleared (excluding the ones in variables_to_keep)
variables_to_clear = [var for var in all_variables if var not in variables_to_keep]

# Clear the selected variables
%reset_selective -f {", ".join(variables_to_clear)}
```

The above code exists to clean up all variables except for `df` dataframe. This is done to ensure memory usage is reduced

Representation and Analysis

3.1 Sparse Vector Representation

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()

# Fit and transform the tokenized_content to create TF-IDF representation
sparse_vector = tfidf_vectorizer.fit_transform(df['tokenized_content'].apply(lambda x: ' '.join(x)))

# Get the feature names (unique tokens)
feature_names_tfidf = tfidf_vectorizer.get_feature_names_out()
print("feature_names_tfidf")
print(feature_names_tfidf)

# Convert the sparse representation to a DataFrame
sparse_vector_df = pd.DataFrame(sparse_vector.toarray(), columns=feature_names_tfidf)


# print(len(sparse_vector_df.columns))

# Get the dimensions of the sparse vector
num_sparse_articles, num_features = sparse_vector.shape

print(f"Number of articles transformed: {num_sparse_articles}")
print(f"Number of unique features (tokens): {num_features}")

# Sparse TF-IDF DataFrame:
sparse_vector_df.head()
```

```
feature_names_tfidf
['aa' 'aaa' 'aac' ... 'zwj' 'zxyxggushn' 'zylva']
Number of articles transformed: 12000
Number of unique features (tokens): 55470
```

	aa	aaa	aac	aacute	aadareayate	aadarei	aadarsh	aadhar	aadila	aadit
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(

5 rows × 55470 columns

For Sparse Vector representation we have used TF-IDF. Initially the BOW approach was considered, however the TFIDF approach would be better as it helps emphasize tokens that are more discriminative across the entire dataset

We use the TfidfVectorizer to transform a collection of tokenized content into a TF-IDF (Term Frequency-Inverse Document Frequency) representation.

We then convert this into a dataframe (`sparse_vector_df`) for ease of use.

Interpretation:

Number of articles: The variable num_sparse_articles represents the number of articles (rows) in the DataFrame. In this case, there are 12000 articles in the 'tokenized_content' column.

Number of unique features (tokens): The variable num_features represents the number of unique tokens (words) in the corpus. It indicates the number of columns in the sparse vector representation. In this case, there are 55403 unique tokens.

The "Sparse TF-IDF DataFrame" printed at the end shows the sparse vector representation in tabular form, with the TF-IDF scores for each token in each document. The values in the DataFrame indicate the importance of each token in the respective document, and 0 values represent tokens that do not occur in the specific document.

```
# split the data into train and test sets
from sklearn.model_selection import train_test_split

# Split the data into train and test sets
x_train_sparse, x_test_sparse, y_train_sparse, y_test_sparse = train_test_split(sparse_vector_df, df['source'], test_size=0.2,

print("Sparse vector df split as test and train")
```



```
st and train
```

In preparation of classification, we will be doing a 80/20 split for train and test respectively

3.2 Dense Vector Representation

> justify the dimensions of the dense vector

```
from gensim.models import Word2Vec
import numpy as np

tokenized_content_list = df['tokenized_content'].tolist()

model = Word2Vec(sentences=tokenized_content_list, vector_size=100, window=5, min_count=1, workers=4)

# Function to convert tokenized content into dense vectors
def get_doc_embedding(doc_tokens):
    embeddings = [model.wv[token] for token in doc_tokens if token in model.wv]
    if embeddings:
        return np.mean(embeddings, axis=0)
    return np.zeros(model.vector_size)  # Return zeros for empty documents

# Apply the function to the 'tokenized_content' column and store the dense vectors in a new column 'dense_vectors'
df['dense_vectors'] = df['tokenized_content'].apply(get_doc_embedding)

print("Completed dense vector representation")
```

```
Completed dense vector representation
```

For the dense representation we use the Word2Vec model on the `tokenized_content` column in the DataFrame `df`. The embedding is done using the `get_doc_embedding()`, to convert tokenized content into dense vectors by averaging the word embeddings. The resulting dense vectors are stored in a new column named `dense_vectors` in the DataFrame.

Justification of dimensions

Vector size:

I chose the vector size of 100 as it is a considerably small vector size and it therefore reduces computational complexity and memory. Due to the large volume of datasets I have currently my personal computer and google collab require large memory requirements. the value 100 strikes the right balance between retaining fine-grained information and semantic relationships between words.

Window:

As mentioned in the justification for vector size, due to a large volume of the data set I had some performance issues, the window size 5 was very efficient for computational efficiency. Additionally the Contextual and Sementic relationships did not take much of a hit as shown in the accuracy of the models

```
from sklearn.model_selection import train_test_split

# Assuming 'source' column contains the class labels
x_dense = np.vstack(df['dense_vectors'].to_numpy())

# Split the data into 80% training and 20% testing
x_train_dense, x_test_dense, y_train_dense, y_test_dense = train_test_split(x_dense, df['source'], test_size=0.2, random_state

print("Dense vector df split as test and train")
```

```
Dense vector df split as test and train
```

In preparation of classification, we will be doing a 80/20 split for train and test respectively

4. Training Classifiers with Non-Deep Learning Algorithms

For the section I have chosen Logistic Regression, Random Forest Classificiation and SVC.

Logistic Regression Classification:

Logistic Regression is a simple linear model that is well-suited for multi-class classification tasks like we have in this task. It is computationally efficient and can handle large datasets with ease. Additionally, it provides probabilistic outputs, making it useful for understanding class probabilities.

This ensemble learning method combines decision trees, providing improved accuracy and reduced overfitting. It handles multi-class problems, different feature types, and noisy data, making it a reliable choice for real-world datasets. It also identifies influential features and performs well on diverse classification tasks.

SVM Classification (Support Vector Machine):

SVM is a powerful and versatile classifier suitable for high-dimensional spaces. It works well with both linear and non-linear classification tasks, avoiding overfitting even with small datasets. SVM's kernel functions handle large feature sets effectively, making it popular for various classification tasks with complex decision boundaries.

4.1 Classification using Sparse Vector Representation

4.1.1 Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Initialize classifiers
logreg_clf = LogisticRegression()

# Training the classifiers
print("Training for LR")
logreg_clf.fit(x_train_sparse, y_train_sparse)

# Making predictions
print("Predicting for LR")
y_pred_logreg = logreg_clf.predict(x_test_sparse)

# Evaluating the classifiers
```

```
accuracy_logreg = accuracy_score(y_test_sparse, y_pred_logreg)

print("Logistic Regression Accuracy:", accuracy_logreg)

print("Logistic Regression Classification Report:")
print(classification_report(y_test_sparse, y_pred_logreg))
```

```
Training for LR
/Users/muljayan/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs f
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Predicting for LR
Logistic Regression Accuracy: 0.6483333333333333
Logistic Regression Classification Report:
                 precision    recall  f1-score   support

      adaderana       0.50      0.54      0.52       239
    ceylon_today       0.54      0.55      0.55       235
 colombo_gazzete       0.83      0.84      0.84       245
    daily_mirror       0.53      0.63      0.57       225
      daily_news       0.66      0.66      0.66       245
    economy_next       0.90      0.90      0.90       225
       hiru_news       0.41      0.34      0.37       246
       newsfirst       0.94      0.91      0.93       254
        newswire       0.71      0.55      0.62       234
      the_island       0.49      0.56      0.53       252

        accuracy                           0.65      2400
       macro avg       0.65      0.65      0.65      2400
    weighted avg       0.65      0.65      0.65      2400
```

4.1.2 Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Initialize classifiers
rf_clf = RandomForestClassifier()
```



```
print("Training for RFC")
rf_clf.fit(x_train_sparse, y_train_sparse)

# Making predictions
print("Predicting for RFC")
y_pred_rf = rf_clf.predict(x_test_sparse)

# Evaluating the classifiers
accuracy_rf = accuracy_score(y_test_sparse, y_pred_rf)

print("Random Forest Accuracy:", accuracy_rf)

print("Random Forest Classification Report:")
print(classification_report(y_test_sparse, y_pred_rf))
```

```
Training for RFC
Predicting for RFC
Random Forest Accuracy: 0.665
Random Forest Classification Report:
                 precision    recall  f1-score   support

      adaderana       0.57      0.48      0.52       239
    ceylon_today       0.53      0.54      0.54       235
 colombo_gazzete       0.85      0.90      0.87       245
    daily_mirror       0.46      0.65      0.54       225
      daily_news       0.90      0.68      0.77       245
    economy_next       0.90      1.00      0.94       225
       hiru_news       0.38      0.35      0.37       246
       newsfirst       0.93      0.94      0.94       254
        newswire       0.68      0.60      0.64       234
      the_island       0.52      0.53      0.52       252

        accuracy                           0.67      2400
       macro avg       0.67      0.67      0.66      2400
```

```
weighted avg       0.67      0.67      0.66      2400
```

4.1.3

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Initialize classifiers
svm_clf = SVC()

# Training the classifiers
print("Training for SVC")
svm_clf.fit(x_train_sparse, y_train_sparse)

# Making predictions
print("Predicting for SVC")
y_pred_svm = svm_clf.predict(x_test_sparse)

# Evaluating the classifiers
accuracy_svm = accuracy_score(y_test_sparse, y_pred_svm)

print("SVM Accuracy:", accuracy_svm)

print("SVM Classification Report:")
print(classification_report(y_test_sparse, y_pred_svm))
```

```
Training for SVC
```

```
print("Logistic Regression Accuracy:", accuracy_logreg)
print("Random Forest Accuracy:", accuracy_rf)
print("SVM Accuracy:", accuracy_svm)

print("Logistic Regression Classification Report:")
print(classification_report(y_test_sparse, y_pred_logreg))

print("Random Forest Classification Report:")
print(classification_report(y_test_sparse, y_pred_rf))

print("SVM Classification Report:")
print(classification_report(y_test_sparse, y_pred_svm))
```

Saved successfully!

Vector Representation

4.2.1 Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Initialize the classifiers
clf_logreg = LogisticRegression(max_iter=1000, random_state=42)
# Train the classifiers
clf_logreg.fit(x_train_dense, y_train_dense)
# Make predictions on the test set
y_pred_logreg = clf_logreg.predict(x_test_dense)
# Evaluate the classifiers
accuracy_logreg = accuracy_score(y_test_dense, y_pred_logreg)

# print("Accuracy (Logistic Regression):", accuracy_logreg)
# print("\nClassification Report (Logistic Regression):\n", classification_report(y_test_dense, y_pred_logreg))
```

4.2.2 RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Initialize the classifiers
clf_rf = RandomForestClassifier(random_state=42)

# Train the classifiers
clf_rf.fit(x_train_dense, y_train_dense)
```

```
# Make predictions on the test set
y_pred_rf = clf_rf.predict(x_test_dense)

# Evaluate the classifiers
accuracy_rf = accuracy_score(y_test_dense, y_pred_rf)

# print("Accuracy (Random Forest):", accuracy_rf)
# print("Classification Report (Random Forest):\n", classification_report(y_test_dense, y_pred_rf))
```

4.2.3 SVC

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Initialize the classifiers
clf_svm = SVC(random_state=42)

# Train the classifiers
clf_svm.fit(x_train_dense, y_train_dense)

# Make predictions on the test set
y_pred_svm = clf_svm.predict(x_test_dense)

# Evaluate the classifiers
accuracy_svm = accuracy_score(y_test_dense, y_pred_svm)

# print("Accuracy (SVM):", accuracy_svm)

# print("Classification Report (SVM):\n", classification_report(y_test_dense, y_pred_svm))
```

4.2.4 Comparison

```
print("Accuracy (Logistic Regression):", accuracy_logreg)
print("Accuracy (Random Forest):", accuracy_rf)
print("Accuracy (SVM):", accuracy_svm)


print("\nClassification Report (Logistic Regression):\n", classification_report(y_test_dense, y_pred_logreg))
print("Classification Report (Random Forest):\n", classification_report(y_test_dense, y_pred_rf))
print("Classification Report (SVM):\n", classification_report(y_test_dense, y_pred_svm))
```

```
Accuracy (Logistic Regression): 0.42916666666666664
Accuracy (Random Forest): 0.44333333333333336
```

Saved successfully!

```
Classification Report (Logistic Regression):
                 precision    recall  f1-score   support

      adaderana       0.24      0.28      0.26       225
   ceylon_today       0.39      0.41      0.40       239
colombo_gazzete       0.53      0.62      0.57       234
   daily_mirror       0.41      0.45      0.43       245
     daily_news       0.33      0.28      0.30       246
   economy_next       0.63      0.73      0.68       252
      hiru_news       0.29      0.15      0.20       254
      newsfirst       0.65      0.66      0.66       245
       newswire       0.29      0.20      0.24       225
     the_island       0.37      0.50      0.42       235

       accuracy                           0.43      2400
      macro avg       0.41      0.43      0.42      2400
   weighted avg       0.42      0.43      0.42      2400

Classification Report (Random Forest):
                 precision    recall  f1-score   support

      adaderana       0.24      0.30      0.27       225
   ceylon_today       0.44      0.40      0.42       239
colombo_gazzete       0.33      0.44      0.38       234
   daily_mirror       0.44      0.55      0.49       245
     daily_news       0.72      0.68      0.70       246
   economy_next       0.62      0.72      0.67       252
      hiru_news       0.33      0.21      0.26       254
      newsfirst       0.46      0.37      0.41       245
       newswire       0.34      0.20      0.25       225
     the_island       0.45      0.53      0.49       235

       accuracy                           0.44      2400
      macro avg       0.44      0.44      0.43      2400
   weighted avg       0.44      0.44      0.44      2400

Classification Report (SVM):
```

```
                precision    recall  f1-score   support

     adaderana       0.26      0.44      0.33       225
  ceylon_today       0.47      0.39      0.43       239
colombo_gazzete      0.50      0.59      0.55       234
   daily_mirror      0.40      0.57      0.47       245
    daily_news       0.37      0.28      0.32       246
  economy_next       0.64      0.78      0.70       252
     hiru_news       0.44      0.18      0.26       254
     newsfirst       0.74      0.51      0.60       245
      newswire       0.35      0.18      0.24       225
    the_island       0.38      0.51      0.43       235

      accuracy                           0.44      2400
     macro avg       0.45      0.44      0.43      2400
  weighted avg       0.46      0.44      0.43      2400
```

4.2.4 Comparison

Based on the above outcome we can see that the accuracy of the models for Dense representation are as follows.

- Logistic Regression: 0.42916666666666664
- Random Forest: 0.44333333333333336
- SVM: 0.44375

The low accuracy can be attributed to the amount of articles that had to be dropped due to performance issues

5. Training Classifiers with Deep Learning Algorithms

5.1 Convolutional Neural Network (CNN) (Embedding)

CNNs are excellent for capturing local patterns in data, making them well-suited for tasks like sentiment analysis, spam detection, and short text classification. Word embeddings enhance CNNs' capabilities by providing a meaningful representation of words, enabling the model to understand semantic relationships and achieve better generalization.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, Dense, Reshape, Dense, Conv1D, GlobalMaxPooling1D, Dropout
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score

# Convert source labels to numerical values using LabelEncoder
Saved successfully!
                                  it_transform(y_train_dense)
y_test_encoded = label_encoder.transform(y_test_dense)

num_features = 10

# Create the CNN model
cnn_model = Sequential()
cnn_model.add(Reshape((x_train_dense.shape[1], 1), input_shape=(x_train_dense.shape[1],)))
cnn_model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(num_features, 1)))
cnn_model.add(GlobalMaxPooling1D())
cnn_model.add(Dense(128, activation='relu'))
cnn_model.add(Dropout(0.5))
cnn_model.add(Dense(len(label_encoder.classes_), activation='softmax'))

# Compile the model
cnn_model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
cnn_model.fit(x_train_dense, y_train_encoded, epochs=10, batch_size=64, validation_split=0.1)

# Evaluate the model on the test set
test_loss, test_accuracy = cnn_model.evaluate(x_test_dense, y_test_encoded)
print("CNN Model Accuracy:", test_accuracy)
```

```
Epoch 1/10
135/135 [==============================] - 12s 5ms/step - loss: 2.2966 - accuracy: 0.1138 - val_loss: 2.2823 - val_accura
Epoch 2/10
135/135 [==============================] - 0s 3ms/step - loss: 2.2665 - accuracy: 0.1499 - val_loss: 2.2469 - val_accurac
Epoch 3/10
135/135 [==============================] - 0s 3ms/step - loss: 2.2204 - accuracy: 0.1802 - val_loss: 2.1983 - val_accurac
Epoch 4/10
135/135 [==============================] - 0s 3ms/step - loss: 2.1784 - accuracy: 0.2009 - val_loss: 2.1752 - val_accurac
Epoch 5/10
135/135 [==============================] - 0s 3ms/step - loss: 2.1553 - accuracy: 0.2134 - val_loss: 2.1687 - val_accurac
Epoch 6/10
```

```
135/135 [==============================] - 0s 3ms/step - loss: 2.1419 - accuracy: 0.2135 - val_loss: 2.1522 - val_accurac
Epoch 7/10
135/135 [==============================] - 0s 3ms/step - loss: 2.1317 - accuracy: 0.2154 - val_loss: 2.1435 - val_accurac
Epoch 8/10
135/135 [==============================] - 0s 4ms/step - loss: 2.1231 - accuracy: 0.2250 - val_loss: 2.1418 - val_accurac
Epoch 9/10
135/135 [==============================] - 1s 4ms/step - loss: 2.1184 - accuracy: 0.2236 - val_loss: 2.1489 - val_accurac
Epoch 10/10
135/135 [==============================] - 1s 4ms/step - loss: 2.1124 - accuracy: 0.2247 - val_loss: 2.1312 - val_accurac
75/75 [==============================] - 0s 3ms/step - loss: 2.0948 - accuracy: 0.2333
CNN Model Accuracy: 0.23333333432674408
```

In the above code, we have created and trained a Convolutional Neural Network (CNN) for a multi-class classification task using TensorFlow and Keras. The main steps are as follows:

- Converted source labels to numerical values using LabelEncoder.
- Designed a CNN model architecture consisting of Conv1D, GlobalMaxPooling1D, Dense, and Dropout layers.
- Compiled the model with the 'sparse_categorical_crossentropy' loss function, 'adam' optimizer, and 'accuracy' metric.
- Trained the model on the training data with 10 epochs, a batch size of 64, and used 10% of the data as a validation set.
- Evaluated the model on the test data and obtained the test accuracy.

The accuracy is low here as we have dropped many rows due to performance issues

5.2 LSTM with Word Embeddings

LSTMs are designed to handle sequential data, and they can capture long-range dependencies in the text. When combined with word embeddings, LSTMs can efficiently process textual data and maintain an understanding of the context throughout the sequence, leading to improved performance in various NLP tasks.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Reshape
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score

# Convert source labels to numerical values using LabelEncoder
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train_dense)
y_test_encoded = label_encoder.transform(y_test_dense)

# Create the LSTM model
lstm_model = Sequential()
ense.shape[1], 1), input_shape=(x_train_dense.shape[1],)))
on='relu'))
coder.classes_), activation='softmax'))

# Compile the model
lstm_model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
lstm_model.fit(x_train_dense, y_train_encoded, epochs=10, batch_size=64, validation_split=0.1)

# Evaluate the model on the test set
test_loss, test_accuracy = lstm_model.evaluate(x_test_dense, y_test_encoded)
print("LSTM Model Accuracy:", test_accuracy)
```



```
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU ke
Epoch 1/10
135/135 [==============================] - 14s 50ms/step - loss: 2.3057 - accuracy: 0.1281 - val_loss: 2.2863 - val_accu
Epoch 2/10
135/135 [==============================] - 8s 61ms/step - loss: 58057.3633 - accuracy: 0.1222 - val_loss: 2.3004 - val_ac
Epoch 3/10
135/135 [==============================] - 7s 53ms/step - loss: 2.4902 - accuracy: 0.1279 - val_loss: 2.2963 - val_accura
Epoch 4/10
135/135 [==============================] - 8s 58ms/step - loss: 2.2943 - accuracy: 0.1450 - val_loss: 2.2877 - val_accura
Epoch 5/10
135/135 [==============================] - 8s 57ms/step - loss: 2.2850 - accuracy: 0.1429 - val_loss: 2.2823 - val_accura
Epoch 6/10
135/135 [==============================] - 7s 54ms/step - loss: 54096809984.0000 - accuracy: 0.1348 - val_loss: 2.2836 -
Epoch 7/10
135/135 [==============================] - 9s 64ms/step - loss: 2.2819 - accuracy: 0.1348 - val_loss: 2.2810 - val_accura
Epoch 8/10
135/135 [==============================] - 7s 50ms/step - loss: 2.2798 - accuracy: 0.1366 - val_loss: 2.2785 - val_accura
Epoch 9/10
135/135 [==============================] - 8s 62ms/step - loss: 2.2779 - accuracy: 0.1370 - val_loss: 2.2767 - val_accura
Epoch 10/10
135/135 [==============================] - 7s 50ms/step - loss: 2.2761 - accuracy: 0.1497 - val_loss: 2.2749 - val_accura
75/75 [==============================] - 2s 21ms/step - loss: 2.2752 - accuracy: 0.1500
LSTM Model Accuracy: 0.15000000596046448
```

In the above code we do the following

- Imported the necessary libraries and modules from TensorFlow and scikit-learn.
- Converted the source labels of the training and test data into numerical values using the LabelEncoder from scikit-learn.
- Created an LSTM (Long Short-Term Memory) model using the Sequential API from TensorFlow's Keras module. The model consists of an LSTM layer with 128 units and a ReLU activation function, followed by a Dense layer with a softmax activation function to produce the output probabilities for each class.
- Compiled the LSTM model by specifying the loss function as 'sparse_categorical_crossentropy' (suitable for multi-class classification), the optimizer as 'adam', and the evaluation metric as 'accuracy'.
- Trained the LSTM model on the training data using the fit method with 10 epochs, a batch size of 64, and a validation split of 10% for monitoring training progress.
- Evaluated the performance of the trained LSTM model on the test data and obtained the test accuracy.

The accuracy is very low due to the majority of the dataset being dropped as it causes performance issues on my environment.

5.3 Bidirectional Encoder Representations from Transformers (BERT):

BERT's contextual embedding technique revolutionized the field of NLP by pre-training on a large corpus and fine-tuning on downstream tasks. BERT's bidirectional architecture allows it to understand the context of each word in a sentence, capturing complex relationships between words. This results in highly accurate and contextualized representations, making it a powerful choice for sophisticated NLP applications.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from transformers import BertTokenizer, BertForSequenceClassification, AdamW
import torch
from torch.utils.data import DataLoader, TensorDataset, random_split
from tqdm import tqdm
from sklearn.preprocessing import LabelEncoder

# Step 1: Split the dataset into training and testing sets (80/20 split)
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)

# Step 2: Load pre-trained BERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=10)



# Step 3: Tokenize the content and convert to PyTorch tensors
def preprocess(text):

    t)

    # Convert to lowercase
    text = text.lower()

    # Expand contractions
    text = contractions.fix(text)

    # Remove non-alphanumeric and numeric characters
    text = re.sub(r'[^a-zA-Z]', ' ', text)

    return text

def tokenize_data(df):
    input_ids = []
    attention_masks = []
    labels = []

    # To be adjusted according to your dataset and BERT model limitations
    max_length = 256

    for content, source in tqdm(zip(df['content'], df['source'])):
        # Skip empty content
        if not content:
            continue

        content = preprocess(content)

        encoded_data = tokenizer.encode_plus(
            content,
            add_special_tokens=True,
            max_length=max_length,
            # Use truncation to fit the content within 'max_length'
            truncation=True,
            padding='max_length',
```

Saved successfully!

```
            return_attention_mask=True,
            return_tensors='pt'
        )

        input_ids.append(encoded_data['input_ids'])
        attention_masks.append(encoded_data['attention_mask'])
        labels.append(label_encoder.transform([source])[0])  # Encode the label for this sample

    input_ids = torch.cat(input_ids, dim=0)
    attention_masks = torch.cat(attention_masks, dim=0)
    labels = torch.tensor(labels)

    return input_ids, attention_masks, labels

train_input_ids, train_attention_masks, train_labels = tokenize_data(train_df)
test_input_ids, test_attention_masks, test_labels = tokenize_data(test_df)

print(f"Train input_ids shape: {train_input_ids.shape}")
print(f"Train attention_masks shape: {train_attention_masks.shape}")
print(f"Train labels shape: {train_labels.shape}")

print(f"Test input_ids shape: {test_input_ids.shape}")
print(f"Test attention_masks shape: {test_attention_masks.shape}")
print(f"Test labels shape: {test_labels.shape}")

# Step 5: Create PyTorch DataLoader for efficient batching
train_dataset = TensorDataset(train_input_ids, train_attention_masks, torch.tensor(train_labels))
test_dataset = TensorDataset(test_input_ids, test_attention_masks, torch.tensor(test_labels))

batch_size = 8  # Adjust according to your system's memory
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_dataloader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Step 6: Fine-tune the BERT model on your dataset
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

optimizer = AdamW(model.parameters(), lr=2e-5, eps=1e-8)  # Learning rate and epsilon hyperparameters may need to be tuned
num_epochs = 4  # Adjust as needed

model.train()
for epoch in range(num_epochs):
    for batch in tqdm(train_dataloader):
        input_ids, attention_masks, labels = batch
        input_ids, attention_masks, labels = input_ids.to(device), attention_masks.to(device), labels.to(device)

Saved successfully!                       , attention_mask=attention_masks, labels=labels)

        loss.backward()
        optimizer.step()

# Step 7: Evaluate the model on the test set
model.eval()
with torch.no_grad():
    correct_predictions = 0
    total_predictions = 0

    for batch in tqdm(test_dataloader):
        input_ids, attention_masks, labels = batch
        input_ids, attention_masks, labels = input_ids.to(device), attention_masks.to(device), labels.to(device)

        outputs = model(input_ids, attention_mask=attention_masks)
        logits = outputs.logits
        predictions = torch.argmax(logits, dim=1)
        correct_predictions += (predictions == labels).sum().item()
        total_predictions += len(labels)

accuracy = correct_predictions / total_predictions
print(accuracy)
print(f'Test Accuracy: {accuracy * 100:.2f}%')
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
9600it [01:56, 82.69it/s]
2400it [00:27, 88.57it/s]
<ipython-input-23-e5281e0c8eca>:82: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.c
  train_dataset = TensorDataset(train_input_ids, train_attention_masks, torch.tensor(train_labels))
<ipython-input-23-e5281e0c8eca>:83: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.c
  test_dataset = TensorDataset(test_input_ids, test_attention_masks, torch.tensor(test_labels))
Train input_ids shape: torch.Size([9576, 256])
Train attention_masks shape: torch.Size([9576, 256])
```

```
Train labels shape: torch.Size([9576])
Test input_ids shape: torch.Size([2395, 256])
Test attention_masks shape: torch.Size([2395, 256])
Test labels shape: torch.Size([2395])
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:411: FutureWarning: This implementation of AdamW is
  warnings.warn(
100%|██████████| 1197/1197 [07:22<00:00,  2.70it/s]
100%|██████████| 1197/1197 [07:24<00:00,  2.70it/s]
100%|██████████| 1197/1197 [07:24<00:00,  2.69it/s]
100%|██████████| 1197/1197 [07:24<00:00,  2.70it/s]
100%|██████████| 300/300 [00:39<00:00,  7.62it/s]0.7294363256784969
Test Accuracy: 72.94%
```

In this code, we have used a pre-trained BERT model, which is a powerful language model, to train it for a specific task - classifying text into one of 10 categories.

- Data Preparation: We split our dataset into two parts, one for training and one for testing. We are not using the existing dense and sparse values as BERT has its own type of tokenization and embeddings
- Tokenization: We used a BERT tokenizer to convert our text data into numerical representations that BERT can understand.
- Model Setup: We loaded a pre-trained BERT model designed for sequence classification and adjusted it to handle our 10 categories.
- Data Conversion: We converted our tokenized data into PyTorch tensors, a format that BERT can process efficiently.
- Model Training: We fine-tuned the BERT model on our training data to make it better at classifying sequences.
- Model Evaluation: We tested the trained model on the test data to see how accurately it can classify the text sequences.

The test accurary is shown as 72.94% This can be improved with more data.