Exercise2

March 8, 2024

[ ]:
```
import time
import requests
from bs4 import BeautifulSoup
import pandas as pd
pd.set_option('display.max_colwidth', 200)
pd.set_option('display.max_rows', None)

'''For the purpose of not making this pdf version of the notebook many-thousand␣
 ↪pages long
i have not executed some of the cells. Mainly because the scraping of the␣
 ↪webpages on bbc took
over 2 hours to complete. As seen by the code the output was saved into a csv␣
 ↪file with columns for
each key in the result dicts, namely: "Headline", "Author", "Published_date"␣
 ↪and "Text". ''';
```

[ ]:
```
#Part1
#1
#Importing the cleaned dataset from last assignment
news_data_cleaned = pd.read_csv('news_data_cleaned.csv')
```

[ ]:
```
#2
#A
#In determining which article types should be omitted we can start
#by looking at all the different types:
unique_types = news_data_cleaned['type'].unique()
# print(unique_types)

'''Since we are to make a fake news predicter given a set of labeled data it␣
 ↪might
not be wise to include the documents where the label is 'unknown' since it
maybe indicate that the process that labeled the data in the first place
were unable to decide anything for this. Furthermore when reading about the
dataset on github we see that articles with the label 'clickbait' generally␣
 ↪contain
credible news - just with very exaggerated headlines to promote attention, so
```

```
there is reason to group this together with the reliable data. In continuation␣
 ↪of this
i have also chosen to include documents with the label 'political' since it is
stated on github that these contain "generally verifiable information in␣
 ↪support of
certain points of view or political orientation." indicating the the information
residing in the articles are not fake at all - they just might have some␣
 ↪underlying
political view but in my opinion that does not constitute fake.'''

#B
#This all results in the following grouping of 'fake' and 'reliable':
fake_news = ['unreliable', 'fake', 'conspiracy', 'bias', 'hate', 'junksci']
reliable_news = ['clickbait', 'reliable', 'political']

def label_news(label):
    '''Function for labeling the data
    according to the two new categories:'''
    if label in fake_news:
        return 'fake'
    else:
        return 'reliable'

#Deleting rows where label is 'unknown':
news_data_cleaned_filtered = news_data_cleaned[news_data_cleaned['type'] !=␣
 ↪'unknown']

article_types_old = news_data_cleaned_filtered['type']

# Label the type column with the new appropriate label using the label_news␣
 ↪function
article_types_new = news_data_cleaned_filtered['type'].apply(lambda x:␣
 ↪label_news(x))


#C
# Print value counts for content_column and article_type
print('Articly types and counts, old: ', article_types_old.value_counts())
print('Articly types and counts, new: ', article_types_new.value_counts())

#By our new groupings we end up with 205 articles labeled as fake news and
#39 labeled as reliable.
fake_percentage = (205/(205+39))*100
reliable_percentage =(39/(205+39))*100
print('Percentage of fake articles in dataset: ', fake_percentage)
print('Percentage of fake articles in dataset: ', reliable_percentage)
```

```
'''We see that the dataset consists roughly of 84 percent fake news
and 16 reliable. Thereby we can say the the dataset is not very balanced.
This can be an issue when training models with the data, as the overwhelmingly␣
 ↪larger half
of fake news in the dataset will lead to biased models, labeling the majority␣
 ↪of data fake because its
trained almost exclusively on this and failing to label the minority data - in␣
 ↪this case
the reliable news. This can ultimately lead to bad generalization when␣
 ↪introduced to new data.
An unbalanced dataset can also lead to skewed evaluation metrics such as␣
 ↪accuracy.''';
```

```
Articly types and counts, old:  type
fake          155
conspiracy     31
political      23
unreliable      6
bias            6
junksci         6
reliable        3
clickbait       1
hate            1
Name: count, dtype: int64
Articly types and counts, new:  type
fake        205
reliable     39
Name: count, dtype: int64
Percentage of fake articles in dataset:  84.01639344262296
Percentage of fake articles in dataset:  15.983606557377051
```

[ ]:
```
#Part2
#2
response = requests.get('https://www.bbc.com/news/world/europe')
contents = response.text

#3
soup = BeautifulSoup(contents, 'html.parser')
```

[ ]:
```
def extract_article_links(html_content):
    '''function that for each article in some html file retrieves the
     corresponding link and returns these links in a list '''
    #Creating a bs object containing the input html content parsed with the␣
 ↪html.parser.
    soup = BeautifulSoup(html_content, 'html.parser')
    #Find all article tags
```

```
    articles = soup.find_all('div', {'type' : 'article'})
    #Initializing empty list to store links
    article_links = []
    #Traversing article tags, extracting the link for each article and append␣
 ↪it to the list
    #initialized above
    for article in articles:
        link = article.find('a')['href']
        article_links.append(link)
    #Return a list of article links from the input
    return article_links

article_links = extract_article_links(contents)
```

[ ]:
```
#4
def extract_all_article_links(url, total_pages):
    '''Function for extracting every article link in
    each page of "https://www.bbc.com/news/world/INSERTREGIONHERE", given some␣
 ↪region.
    It assumes that one has manually identified the number of total pages of␣
 ↪the website
    in advance and takes that as an argument as well as the url written above'''
    all_article_links = []
    #range from 1st site to total_pages + 1 cause the range function doesnt␣
 ↪include the last
    for page_num in range(1, total_pages + 1):
        #create the appropriate url for the given page
        page_url = f"{url}?page={page_num}"
        response = requests.get(page_url)
        soup = BeautifulSoup(response.text, 'html.parser')
        #All articles are wrapped in a div - so we find_all divs with the␣
 ↪type=article
        articles = soup.find_all('div', {'type' : 'article'})
        for article in articles:
            #Extract all links as with the function "extract_all_links" defined␣
 ↪above.
            link = article.find('a')['href']
            all_article_links.append(link)
    return all_article_links


#Extrating all links from the europe section
# all_article_links = extract_all_article_links('https://www.bbc.com/news/world/
 ↪europe', 42)
# print(all_article_links)
# #Returns 904 links
```

[ ]:
```
#5
'''Dictionary containing each region of interest. Keys are the names of the␣
 ↪regions
and values are tuples containing the appropriate url as well as total page␣
 ↪count pr region,
which i identified manually (by looking at each webpage)'''

regions_dict = {
    'Europe' : ('https://www.bbc.com/news/world/europe', 42),
    'Asia' : ('https://www.bbc.com/news/world/asia', 42),
    'australia' : ('https://www.bbc.com/news/world/australia', 42),
    'Africa' : ('https://www.bbc.com/news/world/africa', 25),
    'Latin America' : ('https://www.bbc.com/news/world/latin_america', 41),
    'Middle East' : ('https://www.bbc.com/news/world/middle_east', 41)
}

def extract_all_region_article_links(regions):
    '''Function for extracting all article links from different regions of the
    bbc news website. As input the function assumes a dictionary containing the␣
 ↪name of the
    region as key and more importantly. - The value of each element in the␣
 ↪dictionary
    should be a tuple containing the appropriate url as well as the total page␣
 ↪count'''
    all_region_article_links = {}
    for region, (url, total_pages) in regions.items():
        region_links = extract_all_article_links(url, total_pages)
        all_region_article_links[region] = region_links
    return all_region_article_links

all_region_article_links = extract_all_region_article_links(regions_dict)

for region, article_links in all_region_article_links.items():
    print(f"{region}: {len(article_links)} article links found.")

total_links = sum(len(links) for links in all_region_article_links.values())
print(f"Total article links found: {total_links}")
```

```
Europe: 905 article links found.
Asia: 907 article links found.
australia: 827 article links found.
Africa: 492 article links found.
Latin America: 839 article links found.
Middle East: 818 article links found.
Total article links found: 4788
```

[ ]:
```
#6

#Creating list containing tuples with all links and their corresponding region
region_link_data = []
for region, links in all_region_article_links.items():
    for link in links:
        region_link_data.append((region, link))

#Creating dataframe with region, and link columns based on data in list created␣
 ↪above
df_regionandlink = pd.DataFrame(region_link_data, columns=['Region', 'Link'])

'''Uncomment and run the below line to create csv containing all links and␣
 ↪their respective regions''';
# df_regionandlink.to_csv('region_article_links.csv')
```

[ ]:
```
#Part 3
#1 and 2
def extract_article_information(url):
    '''Function for extracting the information of an article given the url to␣
 ↪its location
    it assumes that the base url is bbc's website and the formal
    parameter is the extension of the base and is thus only for scraping␣
 ↪articles on
    the bbc. It returns a dictionary containing the headline, author,␣
 ↪published_date and paragraphs
    of the given webpage.'''
    try:
        response = requests.get('https://www.bbc.com' + url)
        contents = response.text
        soup = BeautifulSoup(contents, 'html.parser')

        author_element = soup.find('div',␣
 ↪class_="ssrcss-68pt20-Text-TextContributorName e8mq1e96")
        #Check if element exists before assigning to author variable to avoid␣
 ↪calling get_text() on none object and get an attribute error
        author = author_element.get_text() if author_element else None

        headline_element = soup.find(id='main-heading')
        #Check if element exists before assigning to author variable to avoid␣
 ↪calling get_text() on none object and get an attribute error
        headline = headline_element.get_text() if headline_element else None

        published_date_element = soup.find('time')
        # Handle KeyError exception when reaching: /news/live/
 ↪uk-england-gloucestershire-67611521
```

```
        # and TypeError when reaching /news/live/world-asia-67605206
        try:
            published_date = published_date_element['datetime'] if␣
↪published_date_element else None
        except KeyError:
            published_date = None

        text_ugly = soup.find_all('p', {'class' : 'ssrcss-1q0x1qg-Paragraph␣
↪e1jhz7w10'})
        text_pretty = []
        for text in text_ugly:
            text_pretty.append(text.get_text())

        #Throwing all the gathered information into dictionary with␣
↪corresponding keys
        article_information_dict = {
            'Headline' : headline,
            'Author' : author,
            'published_date' : published_date,
            'text' : text_pretty
        }
    #Handles exceptions that might occur while fething an article
    except requests.exceptions.RequestException as req_e:
        print(f'An error occured while fetching the url: {req_e}')
        return None

    return article_information_dict
```

[ ]:
```
#Reading in the 'region_article_links.csv' into dataframe
region_article_links = pd.read_csv('region_article_links.csv')
print(region_article_links.head(10))

#Extracting column with the links
article_links = region_article_links['Link']
print(article_links.head(10))
```

```
   Region                          Link
0  Europe     /sport/football/68436828
1  Europe  /news/world-europe-68431803
2  Europe  /news/world-europe-68423990
3  Europe            /news/uk-68429901
4  Europe  /news/world-europe-68423229
5  Europe  /news/world-europe-68415802
6  Europe      /news/business-68401814
7  Europe     /sport/football/68417181
8  Europe  /news/world-europe-68420566
9  Europe  /news/world-europe-68420565
```

```
0       /sport/football/68436828
1    /news/world-europe-68431803
2    /news/world-europe-68423990
3              /news/uk-68429901
4    /news/world-europe-68423229
5    /news/world-europe-68415802
6        /news/business-68401814
7       /sport/football/68417181
8    /news/world-europe-68420566
9    /news/world-europe-68420565
Name: Link, dtype: object
```

[ ]:
```
#3
'''This last cell has not been executed because the output is very large'''

#Looping through the links and using the extract_article_information() function
#on each of these appending resulting dictionary into result list
scraped_article_info = []
for link in article_links:
    print(link)
    #dismiss links in the sport section because it contains entire different␣
 ↪layout
    if str(link[:6]) == '/sport':
        continue
    else:
        scraped_article_info.append(extract_article_information(link))
        #added delay to avoid getting blocked
        time.sleep(1.0)

for article_info in scraped_article_info:
    print(article_info)


df_scraped_article_info = pd.DataFrame(scraped_article_info)

#4
# df_scraped_article_info.to_csv("scraped_article_info.csv")
```

[ ]:
```
'''To determine whether it would make sense to include this newly aquired
data in the dataset we can look at multiple things. First of all we saw that
the original dataset is quite unbalanced. This could be made somewhat up for by␣
 ↪adding all/some
of the scraped data since they come from bbc which one could argue was a␣
 ↪reputable source of
information. However the size of the new dataset is much greater than the␣
 ↪news_data sample so we
```

```
would have to introduce some more from the full dataset as well (assuming that␣
 ↪the full news data set also
consists of mainly fake news articles) to balance the new concatenated dataset.
That said, the scraped data has not been labeled at all so all of the above is␣
 ↪just based on
assumptions about the credibility of the bbc. Another thing that could be worth␣
 ↪to consider doing before
merging the two dataset is to measure the quality and reliability of the␣
 ↪scraped data by for example
looking at the percentage of articles with an author attribution etc. To␣
 ↪introduce this new data into the original dataset
there are therefore many factors to consider to avoid introducing more bias etc.
 ↪ into the dataset.'''
```