Exploration: Domain Extractor for Finicial-Related Terms

Author: Runting Shao

In [1]:

```
#Import python package
import numpy as np
import pandas as pd
from bs4 import BeautifulSoup
import requests
from tqdm import tqdm
import ahocorasick
```

```
C:\Users\SophiaShao\Anaconda3\lib\site-packages\requests\__init__.py:91: RequestsDependen
cyWarning: urllib3 (1.26.7) or chardet (3.0.4) doesn't match a supported version!
  RequestsDependencyWarning)
```

In [2]:

```
train_data = pd.read_csv('topicDF_domains_hosted_outside_US.csv')
#test_data = pd.read_csv('Data/Cleaned_Data/TCCSocialMediaData_test_clean.csv')
unique_domain_train = pd.unique(train_data.domain)
#unique_domain_test = pd.unique(test_data.domain)
#l1 = [x for x in unique_domain_train]
#l2 = [x for x in unique_domain_test]
#unique_domain = np.unique(l1 + l2).tolist()
print(len(unique_domain_train))
```

```
147
```

In [3]:

```
train_data.head()
```

Out[3]:

	Unnamed: 0	domain	voterfraud	antilgbt	antiblack	misogyny	qanon	antilatinx	aliens	antisemitic
0	0	ac24.cz	0	0	0	0	0	0	0	0
1	1	active-democracy.com	1	0	0	0	0	0	0	1
2	2	actualidad.rt.com	1	1	0	1	0	0	0	1
3	3	andrewcarringtonhitchcock.com	0	0	1	0	0	0	0	1
4	4	arab.southfront.org	1	1	0	0	0	0	0	1

5 rows × 26 columns

In [4]:

```
domains = unique_domain_train
```

In [5]:

```
search_terms = {
    1: ["donation","donate","patron"],# Donate, Be a Patron etc.
    2: ["store","shop"], # Shop, Shoping, Shop with us etc.
    3: ["subscribe","subscription","membership"],
    4: ["advertis"], # Advertise, Advertising, Advertisement
    5: ["sale","deal","discount","% off","low price","coupon"],
    6: ["free","no cost"],
    7: ["money","cash","dollar"],
    8: ["pay","buy","earn"],
    9: ["newsletter"]
}
```

In [6]:

```
#This function uses the Aho-Corasick Algorithm to count the existence of a category of te
rms in a text string
def ahocorasickCount(terms, text):
    count = 0
    # Make a searcher
    searcher = ahocorasick.Automaton()
    for i, term in enumerate(terms):
        searcher.add_word(term, i)
    searcher.make_automaton()
    # Add up all counts for a category of terms
    for _ in searcher.iter(text):
        count = count + 1
    return count
```

In [7]:

```
class finicialTermsExtractor():

    def __init__(self, userAgent):
        self.userAgent = userAgent

    '''Read through all the domains and get all availiable htmls
    Parse the htmls with beautiful soup
    Input: domains - list of domains
    Output: 1. accessacle_domain - a dictionry with domain(key) and its html after parsin
g(value)
            2. errors - a list of domains that are not able to open'''
    def htmlCrawler(self, domains):
        errors = []
        accessable_domain = {}
        headers = {'userAgent' : self.userAgent}
        print("Html Crawl Progress - Getting html for all domains:")
        for d in tqdm(domains):
            fulllink = "http://www." + d
            try:
                req = requests.get(fulllink,headers,timeout=5)
                soup = BeautifulSoup(req.text, "html.parser")
                accessable_domain.update({d:soup})
            except Exception as e:
                errors.append(d)
        return accessable_domain, errors

    '''Check if the terms in "dict_sublink" existed in the sublink of the domain
    Input: 1. accessable_domains -  a dictionry with domain(key) and its html after parsi
ng(value)
           2. dict_sublink - dictionary of terms to be checked
    Output: result - a dictionary containing categories of terms (key), whether terms exi
st in sublink(T/F)
            Append true for a category if any of the terms in the category existed as sub
link'''
    def checkSublink(self, accessable_domains, search_terms):
        result = search_terms.copy()
        #Initialize a result dict
        for k in result.keys():
            result.update({k: []})
        for domain in tqdm(accessable_domains):
            domain_name = domain
```

```
            if '.'in domain:
                domain_name = domain[:domain.index('.')]
            soup = accessable_domains.get(domain)
            all_link_txt = soup.get_text()
            #Get all link text and search for terms
            for link in soup.find_all('a'):
                href = link.get("href")
                # Define if a href exists and not belong to the domain and text exists
                if(href and (href[0] == '/' or href[0] =='#' or domain_name in href) and
link.string):
                        all_link_txt = all_link_txt + link.string.lower()
            for category in search_terms:
                terms = search_terms.get(category) # list of finicial related terms
                count = ahocorasickCount(terms, all_link_txt)# count terms existed in al
l_link_txt
                result.get(category).append(count)
        return result

    '''Check if the terms in "dict_adcontent" existed in the text from third-party domain
s
    Input: 1. accessable_domains -  a dictionry with domain(key) and its html after parsi
ng(value)
           2. dict_adcontent - dictionary of terms to be checked
    Output: result - a dictionary containing categories of terms (key), count of terms ex
isted in text
            Append count of all terms existed for a category'''
    def checkAdContent(self, accessable_domains, search_terms):
        #Initialize result
        result = search_terms.copy()
        for k in result.keys():
            result.update({k: []})

        for domain in tqdm(accessable_domains):
            #Get domain name - domain: oann.com, domain name: oann
            domain_name = domain
            if '.'in domain:
                domain_name = domain[:domain.index('.')]
            soup = accessable_domains.get(domain)
            ad_txt = ""
            for link in soup.find_all('a'):
                href = link.get("href")
                # Define if a href exists and not belong to the domain
                if(href and href[0] != '/' and href[0] !='#' and domain_name not in href
):
                    # if the text exists
                    if(link.string):
                        ad_txt = ad_txt + link.string.lower()
                    # if img-alt exists
                    for img in link.find_all('img', alt= True):
                        ad_txt = ad_txt + img['alt'].lower()
            for category in search_terms:
                terms = search_terms.get(category) # list of finicial related terms
                count = ahocorasickCount(terms, ad_txt)# count terms existed in ad_txt
                result.get(category).append(count)
        return result
```

In [8]:

```
#Apply user agent
userAgent = 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Ge
cko) Chrome/95.0.4638.69 Safari/537.36 Edg/95.0.1020.53'
crawler = finicialTermsExtractor(userAgent)
#Getting parsed htmls
accessable_domain, errors = crawler.htmlCrawler(domains)
```

Html Crawl Progress - Getting html for all domains:

```
100%|██████████████████████████████████████████████████████████████████████████████| 14
7/147 [04:27<00:00,  1.82s/it]
```

In [9]: