Comp 396: Re-evaluating Radical Content Pathways In the YouTube Recommendation System

Abstract: In this report I present a YouTube web scraping script whose purpose is to follow the YouTube recommendation system. Then, I present an experiment which starts at a seed video and follows all recommended videos until the scraper has visited five layers of recommended videos. The goal of the experiment is to test the hypothesis that the recommendation system favors right-wing videos and promotes radicalization. Using a dataset of political channels tagged according to political affiliation, I find that the recommendation system seems to recommend left-wing channels more frequently then right-wing channels and thus refute the claim that it has right-wing bias.

Table of Contents

Table of Figures

1. Introduction

After the election of Donald Trump, the rise in popularity of nationalist movements originating in online communities, and the spread of conspiracy theories on the internet, YouTube has been at the center of a lot of scrutiny. The platform has been accused of hosting radical content and its automated video recommendation system of slowly pulling users towards more extreme content by recommending and promoting videos that become gradually more 'hard-core'. A New York Times article claimed in 2018 that the recommendation system had a bias towards inflammatory content and users who watched fairly mainstream news sources were quickly presented with far-right videos. The article argued that this tendency was not a bug but a feature; the algorithm was simply exploiting our natural attraction towards watching more inflammatory content and our instinct to dig deeper into what excites us[1].

The goal of this project was to analyze the type of content that the platform recommended when political content was first watched to identify whether watching right-wing content would pull users down a rabbit-hole of extreme content. Specifically, I wanted to examine whether there was a right-wing advantage: YouTube's recommendation algorithm prefers right-wing content over other perspectives. To do so, I built a scraping tool whose purpose is to follow the videos that the recommendation system recommends. Once the scraper was run enough times and lists of trees starting from right- and left-wing videos saved, I analyzed the content creators that the recommendation system recommended, found their political affiliation in a dataset built and updated by Ledwich and Zaitsev containing the political affiliation of a huge number of political content creators on YouTube (which can be found in the GitHub repo linked in here), and compared the type of creators recommended by the two types of videos.

2. Background and Related Work

In 2019, Anna Zaitsev and Mark Ledwich published a paper in which they argued that empirical investigations of the YouTube algorithm did not support the New York Time's claim. Their paper studied four common accusations against the platform: that it created *radical bubbles*, there was a *right-wing advantage*, it had a *radicalization influence* and that it created *right-wing radicalization pathways*. The authors first build a dataset of over 816 channels with over 10,000 subscribers and where at least 30% of their content is political. Each of these channels was given tags from a list of 18 different tags whose descriptions can be found in Figure 4 of Appendix A. I used an updated version of this dataset to categorize the videos suggested to me. Starting with seed videos from tagged content creators, the authors collected information about the videos that were recommended with a scraping script. They concluded that instead of promoting radical content, the platform actively discourages viewers for visiting radicalizing or inflammatory videos. Their data suggests that channels that present left or centrist political content are advantaged by the recommendation algorithm, while channels that present content on the right are disadvantaged. They found that when someone will watch a video whose content creator belongs to the Partisan Left category, the algorithm will present about 3.4M impressions more to the Center/Left category than the other way around when watching content from content creators labelled as Partisan Right[2].

One of the principal flaws of their method is that they did not login to any account before building the list of recommended videos. They claim that they did not believe the recommendations of an anonymous user are that different from those who have logged into their accounts. However, no evidence is provided to

[1] Tufekci, Zeynep. "YouTube, the Great Radicalizer." The New York Times, The New York Times, 10 Mar. 2018, www.nytimes.com/2018/03/10/opinion/sunday/youtube-politics-radical.html.

[2] Ledwich, M., & Zaitsev, A. (2020). Algorithmic extremism: Examining YouTube's rabbit hole of radicalization. First Monday. https://doi.org/10.5210/fm.v25i3.10419

support that claim. Furthermore, their study seems to focus on individual video recommendations instead of recommendations made to a user who has built a watch history or clicked on previously recommended videos. This last point is important because the recommendation system will also suggest relevant videos based off an entire user profile and not just a single video. In my approach I attempt to faithfully replicate a user's experience by logging into an account with every pass so that the YouTube recommendation system suggests videos based on my user profile and the videos watched. Finally, the NYT article's criticism[3] was levied against the recommendation system and its propensity to push individual's "down the rabbit hole". Instead of limiting myself to the recommendations of one video, my experiment attempts to mimic that descent by visiting multiple layers of recommended videos.

3. Experimental design

a. High-level design

For this project I ran a single experiment using a YouTube web scraper that I built. First, I logged into a YouTube account. Then, after deleting the history, I choose a total of 129 popular videos from channels I deemed left-wing and 116 videos from channels I deemed 'right-wing' as seed videos. For each of these seed videos I collected its main features, and its top 3 recommended videos which were stored in a tree as children to the last visited video. Then I visited the stored recommended videos, collected their features and, in turn, took their recommended videos until my tree had a depth of 5. Finally, using a dataset of political YouTube content creators tagged according to political affiliation and content type, I find the frequency of channels tagged with typical right-wing and left-wing tags.

b. Design details

Login and history deletion:

To login, I had to use emails that weren't Google emails. Over the course of my experiment, I found that trying to log into a Google account with selenium would oftentimes (though not always) get me flagged for suspicious activity and subsequently blocked from logging into my account. I thus created my YouTube accounts with the Yandex free emailing service which never caused Google to raise an error for suspicious activity. Furthermore, I also found that simply sending the entire username and password using the *wedriver.sendkey(key)* method would lead Google to suspect I was using a bot and block my access. To overcome this issue, I sent one key at a time for each character in the username and password with wait time following a normal distribution with mean 0.1 seconds and standard deviation 0.3 seconds.

When creating the YouTube accounts with my Yandex emails, Google prompted me to choose between different data privacy options. I did not authorize Google to save Web & App activity since I wanted my YouTube history to be solely responsible for my recommended videos, ticked the option to keep that data until I deleted it manually, asked to see personalized ads, and declined to receive privacy reminders. These accounts were used so that YouTube content would be curated to their watch history. At the start of each new pass, I deleted the account's YouTube history. Since only the YouTube history data is used to curate recommended content, this ensured that all passes remained independent of one another.

Seed videos:

The scraping algorithm starts with a seed video. This video is the first video that is watched and from which information is extracted. The seed video URL and title used for this project can be found by

[3] Tufekci, Zeynep. "YouTube, the Great Radicalizer." The New York Times, The New York Times, 10 Mar. 2018, www.nytimes.com/2018/03/10/opinion/sunday/youtube-politics-radical.html.

cycling through the root videos of all trees in the 'Project_data_dataset.zip' folder in this repo. The videos were curated by me according to the following different criteria:

- Whether the content creators were tagged with 'PartisanLeft', 'PartisanRight', 'AntiSJW', 'SocialJustice' or personalities in the video typically associated with right-wing YouTube.
- Their popularity with regards to other videos made by the content creator.
- How politically charged the video is and how relevant the content discussed is nowadays; the idea being that not all videos from political content creators are political and I wanted to increase my chances of being recommended political content.
- How representative of the community at large I deemed these videos to be. For example, Jordan Peterson is an influential voice and vocal critic of "leftist hysteria". His Cathy Newman interview is a very popular video in the Anti-SJW community.
- How recently they were published to find a balance between popular videos and videos relevant now.

Video features

For each video I collected the title, content creator, description, date posted, number of views, likes and dislikes, as well as the video length. Furthermore, I decided to watch three minutes of each video to mimic a user that is interested in the content since the recommendation system is sensitive to the amount of time spent on each video. While watching the entirety of every video was unfeasible because it would take too much time, I still wanted to let the video play for three minutes to find a balance between time spent scraping and mimicking user interest.

Recommended videos:

I decided to store the first 3 recommended videos because I wanted to build a tree with multiple layers of recommended videos. The goal was to "go down" the recommendation system so that I could analyze what kind of path I was led on. Ideally, I would've been able to build a deep tree composed of nodes with many children and numerous layers of leaves. Due to time constraints I had to limit myself. The number of videos watched follows a geometric series. Denoting the first term as $a_1 = 1$, common ration r = number of children, number of terms n = tree depth + 1 and a scraping time lower bound as $t = \frac{S_n}{14} * 3$ for a single pass where S_n is the total number of videos watched, 14 the number of videos watched at once, and 3 the maximum amount of time spent per video, we get the following results:

- For r=3, n = 7 → S_6 = 364 and t = 78 minutes
- For r = 3, n = 7, → S_6 = 1093 and t = 234 minutes
- For r = 5, n = 6, → S_6 = 3906 and t = 837 minutes

Using three recommended videos and building a tree of depth 5 was the most time-efficient way of getting a large number of runs, while also exploring the recommendation system to a good enough depth.

4. Implementation of the YouTube scraper:

Libraries used:

Selenium[4]: selenium is a tool used for web browser automation. Using the Google's chromedriver tool, Selenium can automatically navigate a website, interact with website items, as well as locating items so

[4] Muthukadan, Baiju. "Selenium with Python." Selenium with Python - Selenium Python Bindings 2 Documentation, 2011, selenium-python.readthedocs.io/index.html

they can be scraped for further use. The main functionalities I used for my scraper after creating the web driver were:

- **Window manipulation:** I performed a lot of windows manipulations to speed up the process by opening numerous tabs at once at watch videos in them. The main methods I used were *driver.window_handles.pop()* which returns the last opened tab as well as *driver.switch_to.window(tab)* which puts into focus the specified tab and makes it interactable.
- **Locating elements**: for this task I used *webdriver.find_element_by_xpath(str_path)* and *webdriver.find_element_by_css_selector(str_path)* to locate elements using their web page xpaths and CSS paths respectively.
- **Implicit waits**: Sometimes I needed to wait for the webpage to load before trying to look for an item. If I had used any of the two methods above my code would raise an *ElementNotFoundException*. To curb this issue, Selenium gives users the option to use *WebDriverWait(self.driver, seconds to wait).until(EC.condition((By.PATH_TYPE,str_path)))* which will wait a user-defined number of seconds for a condition to be met about an element at a certain path. The conditions I used were *element_to_be_clickable* and *presence_of_element*.

Asyncio[5]: asyncio is a library to write concurrent code using the async/await syntax. I used this library so that I was able to interact with other videos while my scraper "watched" a video.

- **Coroutines: "**a coroutine is a function that can suspend its execution before reaching return, and it can indirectly pass control to another coroutine for some time."[6]. A coroutine is declared with the *async* syntax in front of the method and can be run using the *asyncio.run(coroutine)* method or awaited with await. I use these routines so that other videos can be visited while my scraper "watches" a YouTube video by letting it play.
- **Awaits:** this keyword passes the function control back to the event loop and suspends the execution of the surrounding coroutine, the coroutine in which the awaited function is called. This keyword must be called to get a coroutine's results.
- **Sleep:** I use *asyncio.sleep(seconds)* which is an asynchronous implementation of the *time.sleep(seconds)* method in python. This means that I can call *await asyncio.sleep(seconds)* after having started a video to pause the function that is visiting the video while the video is played for a user-defined number of seconds and go visit another video whose visit is in the event loop.
- **Gather:** Finally, I also use *await asyncio.gather(list_of_video_urls)*. A future is an awaitable object representing the eventual result of an asynchronous operation such as calling a coroutine. The *gather()* method which will create a single future object containing a list of coroutines and wait for all of them to be executed. I use this method to visit a list of videos containing queued videos to visit.

Anytree[7]: This is just a library that implements a Tree data structure. Nodes are created with the following attributes:

- id: the video URL which is unique because I only add videos which have not been visited yet,

[5] "Asyncio - Asynchronous i/o." Asyncio - Asynchronous I/O - Python 3.9.7 Documentation, Python Software Foundation, docs.python.org/3/library/asyncio.html

[6] https://realpython.com/async-io-python/

[7] "Any Python Tree Data." Any Python Tree Data - Anytree 2.8.0 Documentation, anytree.readthedocs.io/en/latest/index.html

- parent: the nodes predecessor and only element which indicates the node's ordering
- video: a dictionary containing every video's scraped information,
- title: the title of the video scraped. This feature is used mostly for visualization purposes since a node's ID is not very informative.

Apart from organizing the nodes into a tree structure and faithfully representing the order in which the videos were watched, the library is also able to convert trees into text JSON files using the *JsonExporter()* method. Furthermore, the library can also convert a JSON file into a tree using the *JSONImporter()* method. This step is essential to store videos over the long term.

Information storage:

I stored the scraped videos in a tree where each node's children are one of its recommended videos and the depth is equal to the number of "layers" of recommended videos that our scraper explored. I only decided to keep recommended videos that are not already present in the tree. The goal of the experiment is to analyze the "new" content recommended by the recommendation system and not get stuck looping over the same sequence of content. Once the scraper was done scraping, each tree is transformed into a JSON text file so that it can be stored for later analysis. I decided to use a tree because the task can naturally be represented using a tree and the tree structure itself can store important information. Using the path to a node we can retrace how a video was eventually recommended. Furthermore, we can also easily understand each video's recommended videos by looking at its children. Finally, the depth of the tree is also important information to have as it indicates at which recommendation layer a video was suggested.

Process:

In this section I will give a detailed explanation of the scraping algorithm scraping method.

Initialization

The scraping algorithm takes as input a text file containing the list of seed videos to be visited by the scraping algorithm. A new scraper object will be created for each seed video inputted in the scraper. Once the scraper object is created and the *run_scraper()* method called, the scraper will login to the YouTube account and delete the account's video history.

Login and history deletion

Logging in requires first going past a prompt to accept Google's cookie tracking. Logging is done by first entering the username and password. Google has a number of security measures put in place which makes this process very tedious during the first couple of times logging in. First, Google will sometimes have a Captcha security test in between the username and password input. To overcome this issue, I simply take a screenshot of the captcha message for it to be manually read by the user and inputted to the scraper. Then, if the computer running the scraper is not yet a trusted computer for the specific YouTube account, YouTube will send a code to the email or phone number associated to the YouTube account. Again, this requires a user to log into their email or phone to manually give the scraper the code. After the login is successful, the account's history will be deleted. This will only happen before any videos are visited.

Logging in is done every time the scraper is about to visit a new set of videos. This is done to speed up the process because the webdriver sees a significant decline in performance if a chrome window is used to visit multiple sets of videos. This seems to be due to a problem with the Selenium library and nothing to do with Wi-Fi connection speed or my script's efficiency.

Building the tree & video tracking

After an initial login and history deletion, a tree root is created with a seed video, the seed video is enqueued in an empty double-ended queue. Every time a set of recommended videos are saved, their URLs are stored at the end of the double-ended queue. I use a double-ended queue so I can dequeue videos at the front of the queue and build the tree layer by layer in the same order as a breadth-first search would visit the nodes of a tree while also keeping track of the videos that need to be seen. I also store all videos which have been visited in an array called *videos_watched* whose length is used in the termination condition.

Scraping the videos

The video URLs to be visited are dequeued from the front of the queue and placed in an array of videos. The method *video_processing()* is then called for all video URLs which are placed in an individual asyncio coroutines and placed in an asyncio future object. The *video_processing()* method is responsible for skipping any ads before the video, calling all the set of methods which extract the information, pausing the coroutine and awaiting for a video to be watched using *asyncio.sleep()*, as well as calling *get_video_recommendations()*. *video_processing()* will place a tuple containing an array of a video's recommended videos and a dictionary containing the scraped information in an array at the index that respects the video's position in the list of the videos used to create the future object. *get_video_recommendations()* extracts a video's recommended videos URL and places these nodes as children in the tree. I place the recommended videos in the tree before they are visited with *video_processing()* to avoid having duplicates.

Once all videos have been visited, their URLs are matched to a node in the tree with the breadth-first-search method *LevelOrderIter()*, the dictionary of extracted information stored in the node and the recommended videos added to the end of the double-ended-queue.

Termination and saving the tree

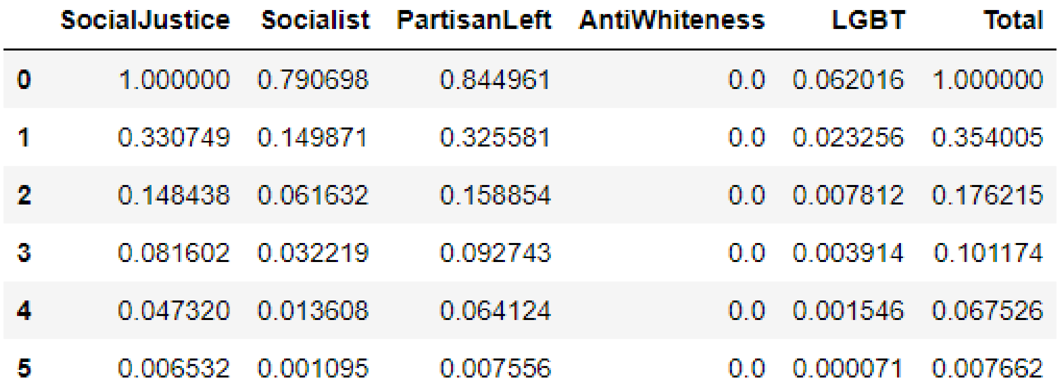
The termination condition is very simple. I use *geometric_series_calc()* to set *num_limit* which equals the total number of videos that should be visited for a given number of recommended videos and depth. Once the length of the *videos_watched* array is equal to *num_limit*, the array of tasks is created empty and the tree written to a text JSON file.

5. Hypothesis evaluation & data analysis:

In this data analysis section, I will investigate the following claim:

Right-Wing Advantage - YouTube's recommendation algorithm prefers right-wing content over other perspectives.

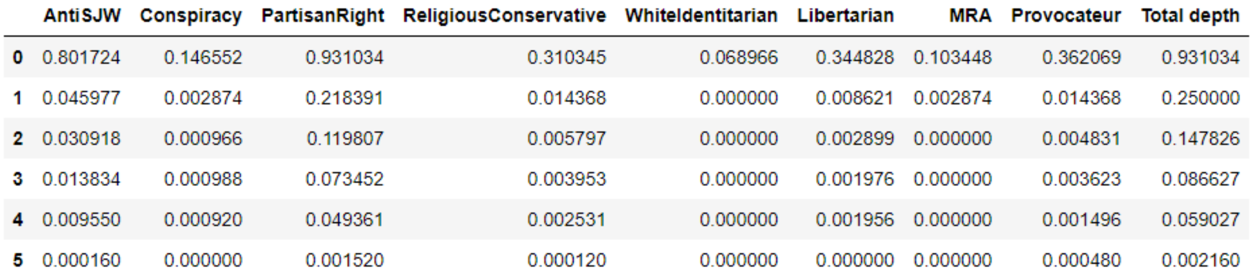
This claim does not seem to be supported by the data gathered over the course of this experiment which can be found in Figure 1 showing the results of trees generated with left-wing seeds



	SocialJustice	Socialist	PartisanLeft	AntiWhiteness	LGBT	Total
0	1.000000	0.790698	0.844961	0.0	0.062016	1.000000
1	0.330749	0.149871	0.325581	0.0	0.023256	0.354005
2	0.148438	0.061632	0.158854	0.0	0.007812	0.176215
3	0.081602	0.032219	0.092743	0.0	0.003914	0.101174
4	0.047320	0.013608	0.064124	0.0	0.001546	0.067526
5	0.006532	0.001095	0.007556	0.0	0.000071	0.007662

Figure 1: Trees generated from left-wing seeds

as well as in Figure 2 detailing the results of trees generated with right-wing seeds.



	AntiSJW	Conspiracy	PartisanRight	ReligiousConservative	WhiteIdentitarian	Libertarian	MRA	Provocateur	Total depth
0	0.801724	0.146552	0.931034	0.310345	0.068966	0.344828	0.103448	0.362069	0.931034
1	0.045977	0.002874	0.218391	0.014368	0.000000	0.008621	0.002874	0.014368	0.250000
2	0.030918	0.000966	0.119807	0.005797	0.000000	0.002899	0.000000	0.004831	0.147826
3	0.013834	0.000988	0.073452	0.003953	0.000000	0.001976	0.000000	0.003623	0.086627
4	0.009550	0.000920	0.049361	0.002531	0.000000	0.001956	0.000000	0.001496	0.059027
5	0.000160	0.000000	0.001520	0.000120	0.000000	0.000000	0.000000	0.000480	0.002160

Figure 2: Trees generated from right-wing seeds

I counted the frequency with which content creators labeled as ‘PartisanRight’ were recommended with a right-wing seed video as well as the frequency with which content creators labeled as ‘PartisanLeft’ were recommended with left-wing seed videos. From Figure 1 we can see that out of the 116 trees created from right-wing videos, 93% of the videos at depth 0 were labelled as ‘PartisanRight’. At depth 1 this number drops to 21%, then to 12%, to 7%, to 5% and then finally to 0.1% of the recommended videos. There is thus a drop of over 90 percentage points between depth 0 and depth 5. From Figure 1 we can also see that out of the 129 trees created from left-wing seeds, 84% of videos at depth 0 were labeled as ‘PartisanLeft’. At depth 1 this drops to 32%, then to 16%, to 9%, to 6% and finally to 0.7%. Even though a smaller percentage of starting video content creators were labeled as ‘PartisanLeft’ then starting video content creators labelled as ‘PartisanRight’, at every subsequent depth there was a higher frequency of recommended videos from ‘PartisanLeft’ content creators then ‘PartisanRight’ content creators. At depth 1 there was an 11 percentage-point difference, at depth 2 a 4 percentage-point difference, at depth 3 a 2 percentage-point difference, at depth 4 a 1.5 percentage-point difference, and finally at depth 5 a 0.6 percentage-point difference. Since the frequency of ‘PartisanLeft’ content creators is higher than the frequency of ‘PartisanRight’ content creators, this is evidence against the hypothesis that there is right-wing advantage in the recommendation system.

Furthermore, when comparing the frequency of content creators labeled as ‘Social Justice’ when starting with a left-wing seed and the frequency of content creators labeled as ‘AntiSJW’ when starting with a right-wing seed, the difference is even more striking. All video creators at depth 0 in the left-wing trees were labeled as ‘SocialJustice’, 33% at depth 1, 15% at depth 2, 8% at depth 3, 5% at depth 4 and finally 0.65% at depth 5. On the other hand, while 80% of content creators were labeled as ‘AntiSJW at depth 0, this number dropped to a mere 4.5% at depth 1, 3% at depth 2, 1% at depth 3, 0.9% at depth 4 and 0.01% at depth 5. Between depth 0 and 1, the frequency of ‘AntiSJW’ content creators drops by 93% while the

‘SocialJustice’ tag frequency drops by only 67%. Furthermore, the frequency of the ‘SocialJustice’ at depth 4 is slightly higher than the frequency of ‘AntiSJW’ at depth 1. This huge difference seems to indicate that the recommendation system is likely to continue suggesting ‘SocialJustice’ content but not ‘AntiSJW’ content. Comparing the ‘Socialist’ tag among the set of left-wing trees to the ‘AntiSJW’ tag among right-wing trees, the difference is also glaring. At depth 0 79% of left-wing tree video content creators were tagged as ‘Socialist’. This is very similar to the frequency of ‘AntiSJW’ tags which sits at 80%. However, at depth 1 the frequency of ‘Socialist’ channels drops to 14.9% against the 0.045% of ‘AntiSJW’ channels.

These findings are also observed if we only keep trees whose root content is labelled ‘AntiSJW’. In figure 2 below, we see the frequency of recommended content creators labelled as ‘AntiSJW’ at every depth for trees whose seed video was given that label. Again, we see an extremely significant drop of 95% between depth 0 and depth 1 with similar frequencies at subsequent depths. Both of these categories are emblematic of ‘left-wing YouTube’ or ‘right-wing YouTube’ and thus the frequency with which they appear is an indication of the YouTube recommendation system’s anti-anti-SJW content and anti-right-wing bias. Indeed, the AntiSJW category is heavily associated to channels part of the right-wing sphere of YouTube as many of its participants such as Ben Shapiro, Steven Crowder or David Ruben are vocal critics of “leftist hysteria”, and one of the defining ideas that many right-wing channels share are “a general opposition to feminism, social justice, or left-wing politics[8]”

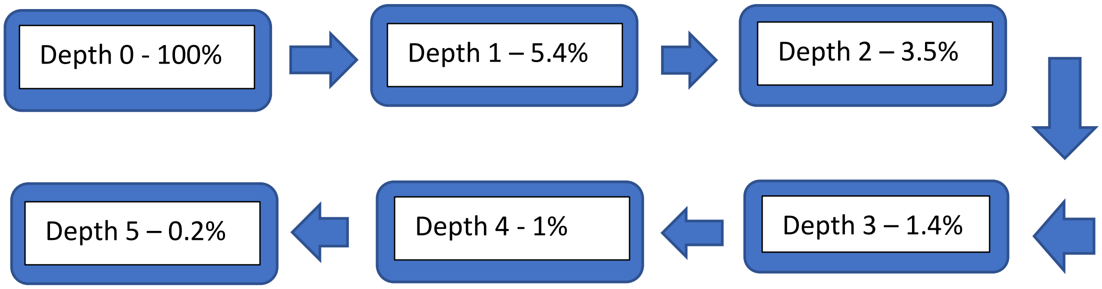


Figure 3: Frequency of AntiSJW tags in trees with roots tagged AntiSJW

Thus, our findings suggest that the YouTube recommendation system does *not* have a right-wing bias. On the contrary, my evidence is in line with Ledwitch & Zaitsev[9] in suggesting the opposite: the YouTube recommendation will suggest left-wing creators more aggressively when starting with left-wing content then it will promote right-wing creators when starting with right-wing content.

6. Limits & Conclusion

My method has a couple of flaws which greatly limits the impact of my conclusion. The goal of this experiment was to identify whether the YouTube algorithm had right-wing bias to measure whether whether users would be suggested more right-wing content which could lead to radicalization. My method assumes that this phenomenon can be measured by mimicking a user watching a YouTube video for three minutes as well as blindly following the YouTube recommendation system. In practice, real-life

[8] Rebecca Lewis, “Alternative Influence: Broadcasting the Reactionary Right on YouTube”. Data & Society, Sept. 18, 2018

[9] Ledwich, M., & Zaitsev, A. (2020). Algorithmic extremism: Examining YouTube’s rabbit hole of radicalization. First Monday. https://doi.org/10.5210/fm.v25i3.10419

users will curate their recommended videos, sometimes clicking on the 1st one, other times the 20th, and other times looking up a video directly. This behavior is not captured by my experiment which limited itself to following three recommended videos.

Furthermore, the NYT article's[10] critic was that the YouTube algorithm's tendency to cater to our attraction to inflammatory content is a feature which leads to radicalization. This psychological phenomenon is not captured at all by building a script which naively follows the recommendation system. While tags such as 'Provocateur' or 'AntiSJW' could be indicators that a channel was posting inflammatory content, some content creators not labelled with these tags could have individual videos with extreme content.

Another limit is that, while 'Provocateur' and 'AntiSJW' are tags which could be a good indicator of inflammatory content, it primarily describes the content creator as a whole and not the ideas promoted in individual videos. While my experiment can provide quantitative evidence to what the recommendation suggests, it cannot on its own prove or disprove that YouTube's recommendation system has been responsible for promoting videos which could lead to online radicalization.

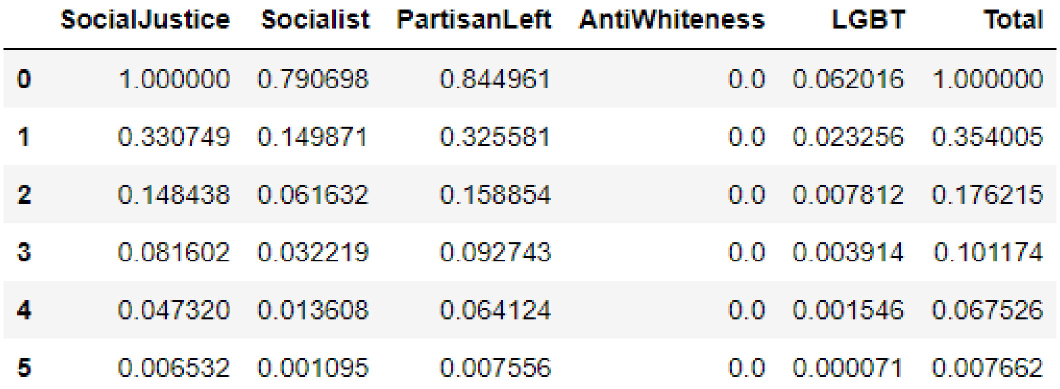
Finally, due to time constraints I was only able to run a scraper which took into consideration the top 3 recommended videos.

In this report I presented my YouTube scraper script, showed how it bypassed Google's login security measures, collected information, incorporated an asynchronous library to make watching videos more efficient and organized the collected information. I also presented its evidence disproving a common accusation that the YouTube recommendation system suggests right-wing content more frequently than other content. These findings are similar to the findings of Ledwitch and Zaitsev[11] in their own investigation of the YouTube recommendation system. While my script is already able to collect a number of features from YouTube videos, I hope to further expand on its data collection functions by enabling it to extract YouTube transcripts and comments.

All the code and folders of trees can be found in this GitHub repo.

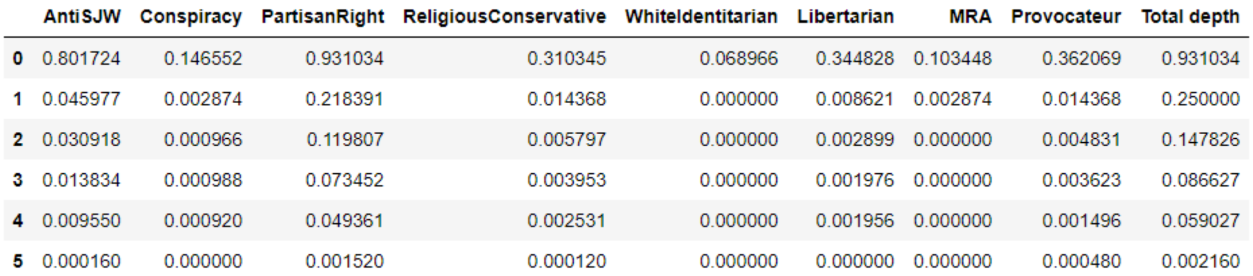
[10] Tufekci, Zeynep. "YouTube, the Great Radicalizer." The New York Times, The New York Times, 10 Mar. 2018, www.nytimes.com/2018/03/10/opinion/sunday/youtube-politics-radical.html.
[11] ibid

Appendix A : Tables & Diagram



	SocialJustice	Socialist	PartisanLeft	AntiWhiteness	LGBT	Total
0	1.000000	0.790698	0.844961	0.0	0.062016	1.000000
1	0.330749	0.149871	0.325581	0.0	0.023256	0.354005
2	0.148438	0.061632	0.158854	0.0	0.007812	0.176215
3	0.081602	0.032219	0.092743	0.0	0.003914	0.101174
4	0.047320	0.013608	0.064124	0.0	0.001546	0.067526
5	0.006532	0.001095	0.007556	0.0	0.000071	0.007662

Figure 1: Trees generated from left-wing seeds



	AntiSJW	Conspiracy	PartisanRight	ReligiousConservative	WhiteIdentitarian	Libertarian	MRA	Provocateur	Total depth
0	0.801724	0.146552	0.931034	0.310345	0.068966	0.344828	0.103448	0.362069	0.931034
1	0.045977	0.002874	0.218391	0.014368	0.000000	0.008621	0.002874	0.014368	0.250000
2	0.030918	0.000966	0.119807	0.005797	0.000000	0.002899	0.000000	0.004831	0.147826
3	0.013834	0.000988	0.073452	0.003953	0.000000	0.001976	0.000000	0.003623	0.086627
4	0.009550	0.000920	0.049361	0.002531	0.000000	0.001956	0.000000	0.001496	0.059027
5	0.000160	0.000000	0.001520	0.000120	0.000000	0.000000	0.000000	0.000480	0.002160

Figure 2: Trees generated from right-wing seeds

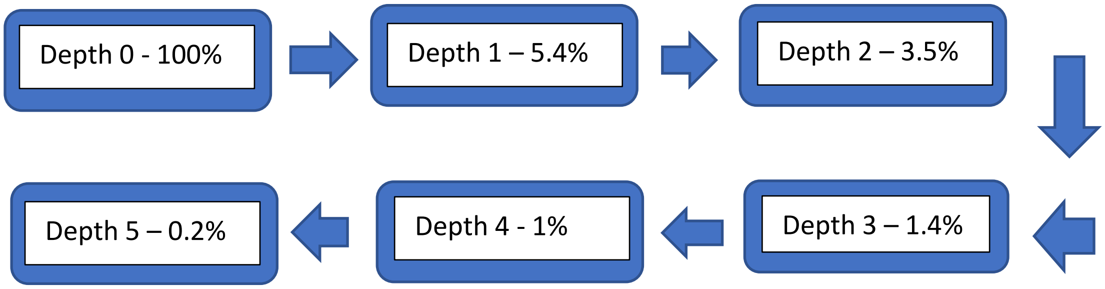
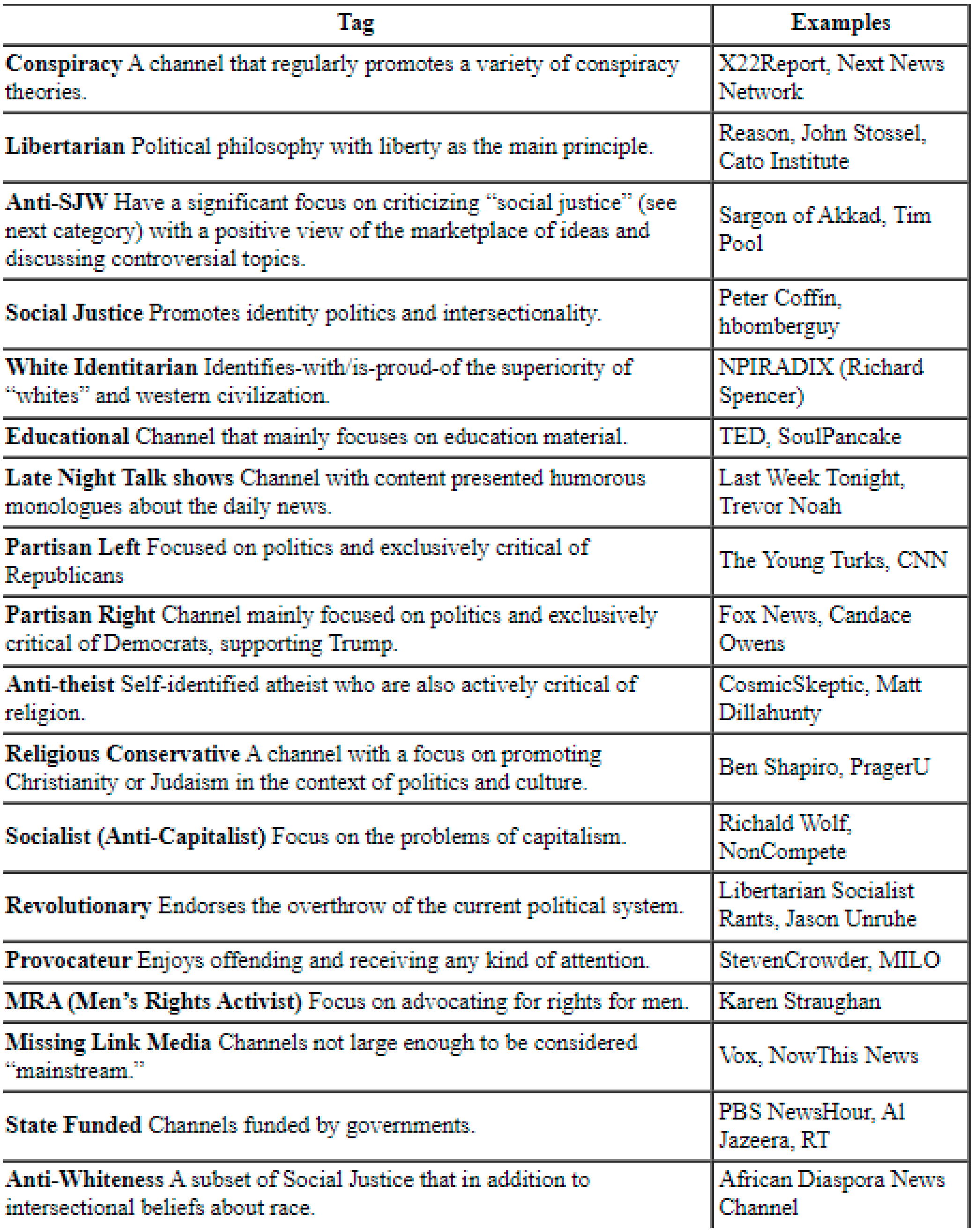


Figure 3: Frequency of AntiSJW tags in trees with roots tagged AntiSJW



Tag	Examples
Conspiracy A channel that regularly promotes a variety of conspiracy theories.	X22Report, Next News Network
Libertarian Political philosophy with liberty as the main principle.	Reason, John Stossel, Cato Institute
Anti-SJW Have a significant focus on criticizing "social justice" (see next category) with a positive view of the marketplace of ideas and discussing controversial topics.	Sargon of Akkad, Tim Pool
Social Justice Promotes identity politics and intersectionality.	Peter Coffin, hbomberguy
White Identitarian Identifies-with/is-proud-of the superiority of "whites" and western civilization.	NPIRADIX (Richard Spencer)
Educational Channel that mainly focuses on education material.	TED, SoulPancake
Late Night Talk shows Channel with content presented humorous monologues about the daily news.	Last Week Tonight, Trevor Noah
Partisan Left Focused on politics and exclusively critical of Republicans	The Young Turks, CNN
Partisan Right Channel mainly focused on politics and exclusively critical of Democrats, supporting Trump.	Fox News, Candace Owens
Anti-theist Self-identified atheist who are also actively critical of religion.	CosmicSkeptic, Matt Dillahunty
Religious Conservative A channel with a focus on promoting Christianity or Judaism in the context of politics and culture.	Ben Shapiro, PragerU
Socialist (Anti-Capitalist) Focus on the problems of capitalism.	Richald Wolf, NonCompete
Revolutionary Endorses the overthrow of the current political system.	Libertarian Socialist Rants, Jason Unruhe
Provocateur Enjoys offending and receiving any kind of attention.	StevenCrowder, MILO
MRA (Men's Rights Activist) Focus on advocating for rights for men.	Karen Straughan
Missing Link Media Channels not large enough to be considered "mainstream."	Vox, NowThis News
State Funded Channels funded by governments.	PBS NewsHour, Al Jazeera, RT
Anti-Whiteness A subset of Social Justice that in addition to intersectional beliefs about race.	African Diaspora News Channel

Figure 4: Table of channel tags and their description from Ledwitch and Zaitsev's paper

Work Cited:

Library documentation:

Muthukadan, Baiju. “Selenium with Python.” Selenium with Python - Selenium Python Bindings 2 Documentation, 2011, selenium-python.readthedocs.io/index.html

“Asyncio - Asynchronous i/o.” Asyncio - Asynchronous I/O - Python 3.9.7 Documentation, Python Software Foundation, docs.python.org/3/library/asyncio.html

“Any Python Tree Data.” Any Python Tree Data - Anytree 2.8.0 Documentation, anytree.readthedocs.io/en/latest/index.html

Paper, Reports and Articles:

Rebecca Lewis, “Alternative Influence: Broadcasting the Reactionary Right on YouTube”. Data & Society, Sept. 18, 2018

Tufekci, Zeynep. “YouTube, the Great Radicalizer.” The New York Times, The New York Times, 10 Mar. 2018, www.nytimes.com/2018/03/10/opinion/sunday/youtube-politics-radical.html.

Ledwich, M., & Zaitsev, A. (2020). Algorithmic extremism: Examining YouTube’s rabbit hole of radicalization. First Monday. https://doi.org/10.5210/fm.v25i3.10419