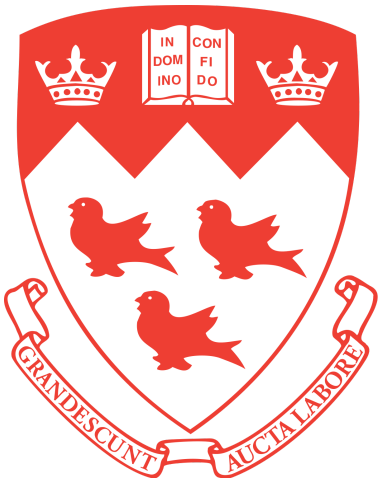
Experiment on Youtube Recommendation System: Conspiracy Theories

Maneth Kulatunge



School of Computer Science, Prometheus Lab, AI and Society Group

Final Report - COMP 400

McGill University

Montreal, Quebec, Canada

Contents

Abstract

This study is an extension to Mika Desblanc's work as it explores a different subject space with respect to the Youtube recommendation algorithm. It contains an experiment that determines if the Youtube algorithm siloes users to watch more conspiracy theory videos. This topic is a pertinent discussion topic as we have seen conspiracy theories affect people in a multitude of ways, especially during the recent COVID-19 pandemic.

In this experiment, I create and utilise a binary text classification algorithm which was pre-trained with a Twitter dataset that was cleaned and annotated before the training process. I have presented two web scraping algorithms that assist with the collection of metadata related to the recommended videos. These algorithms take either a depth or breadth-first approach and create an expansion tree with metadata results of recommendations that utilise a starter video list for the start of its iterative/recursive search.

The results suggest that there is a slight decrease in the percentage of videos recommended as the users keep following the recommendation loop of Youtube which might be an indication of the recent efforts made by the company to reduce the number of conspiracy-based video recommendations.

The results dictate that it's not possible as of now to conclude that there is a large bias for content related to conspiracy theories when examining the current trends but it'll be best to further examine this claim as there is still a relatively high proportion of conspiracy theory videos still being circulated around Youtube.

Information For Future Team

This project is an extension of Mika Desblanc's research. However, I have created a completely new repository as my approach was slightly different to Mika's and my directory organisation is a little different. My Github repository has all the code that I used to conduct analysis on the Youtube recommendation system and its nature when it comes to conspiracy-related videos. I have also attached all the results I retrieved in form of CSV files, python notebooks (google collab) and the script versions of these notebooks. This report provides an overview of my experiment and the improvements that can be made in case someone continues this project. Currently, I have completed a breadth first and depth first search scraper alongside a machine learning model for text classification that is used hand in hand to retrieve the recommendation results.

This is a list of items (click to visit link) that give you an overview of the project:

Text Classification Algorithm

Dataset(s) (feel free to experiment, I have used dataset_balanced.csv for my model training)

Data Cleaning Algorithms

Breadth First Scraper

Depth First Scraper

Analysis of Results

Goal

Observe whether the YouTube recommendation system siloes users towards conspiracy theory-related content.

Youtube and other big tech corporations have come under a lot of criticism over the last few years with regard to the spread of misinformation and conspiracy theories within their platforms. Twitter's new verified feature is at the heart of this discussion in current discourse. Youtube was considered a primary information source for a lot of individuals during the COVID-19 pandemic and this opened the path for conspiracy theories especially about the virus flourish. The platform was forced to change certain procedures to make sure the dissemination of information will be reduced. The goal of this project as mentioned earlier is to investigate the Youtube Recommendation system further to identify if users are still being siloed to watch conspiracy theory-related content.

Deliverable Schedule

Task	Deadline	Status + Comments
BERT Classification	7th November	Completed 8/11- Took longer than initially assumed due to the length taken for scraping tweets, initial ambiguity in the classification approach, and because it took a while to understand how to fine-tune the model with a dataset
Getting Survey Results	20th October	Completed on 27/10 - delayed by a week because there was discussion on whether we should disseminate my original survey. It made sense to use a previously made survey because it answered the required queries
UML Diagrams for code	30th October	Completed 1/11- Finished initial models on time post adjustment of the deadline but made a few more adjustments after changing approach related to classification
Mid Term Report	11th November	Completed on 11/11. I pushed the submission date after speaking to professor Vybihal. Submission on 12th of November because I was sick this week and I wanted to recover a bit before recording the demo
Finishing off code for data scrapers	25th November	Completed Breadth First Scraper on 28/11 Completed Depth First Scraper on 3/12 Reverted to Implementing Breadth and Depth First Scraper as opposed to replicating user actions
Completion of Analysis	30th November	Completed on 5/12. Took longer than expected to get the required data from Scrapers
Final Report	5th December	Completed with Demo

Experiment

Summary

There are broadly two components to my experiment. Web Scraping and Text Classification.

The Web Scraping component has two algorithms being used. A Depth-First Scraper and a Breadth-First Scraper explore the recommendations from a starter list of videos that explore the recommendations of each video in a breadth-first or depth-first approach and collect information related to the videos - title, description and recommendations.

The Text Classification algorithm allows us to pass in the text that we want to classify and have the model return a label (class value) which tells us if it is a conspiracy or not.

This scraper will find the text related to the newly suggested videos and use the aforementioned model to classify if the recommendation is conspiracy-leaning or otherwise.

High-Level Overview of Experiment

Text Classification

Dataset:

Original source of the dataset is from the A Labeled Dataset of Conspiracy Theory and 5G-Corona Misinformation Tweets data paper. The paper contained a data set which had the tweet IDs for 10,000 tweets that consist of conspiracy theories and non conspiracy theories which were collected around the COVID-19 pandemic.

Data Cleaning:

Since the data needed for our model was text I used the Twitter API alongside the Tweepy python library to extract the tweets on the dataset. I ran a script where I tweets from private accounts as these tweet IDs returned an empty string when being retrieved.

Process:

We want to do conspiracy text classification - pass in a youtube comment that we want to classify and have the model return out a label (class value) which tells us if it is a conspiracy or not.

To do this we use the distilBERT language model, a smaller more operable version of the popular BERT language model (with very similar performance). The model is pre-trained on a large corpus of varied data, so it has pre-learned natural language capabilities via masked language modelling (MLM).

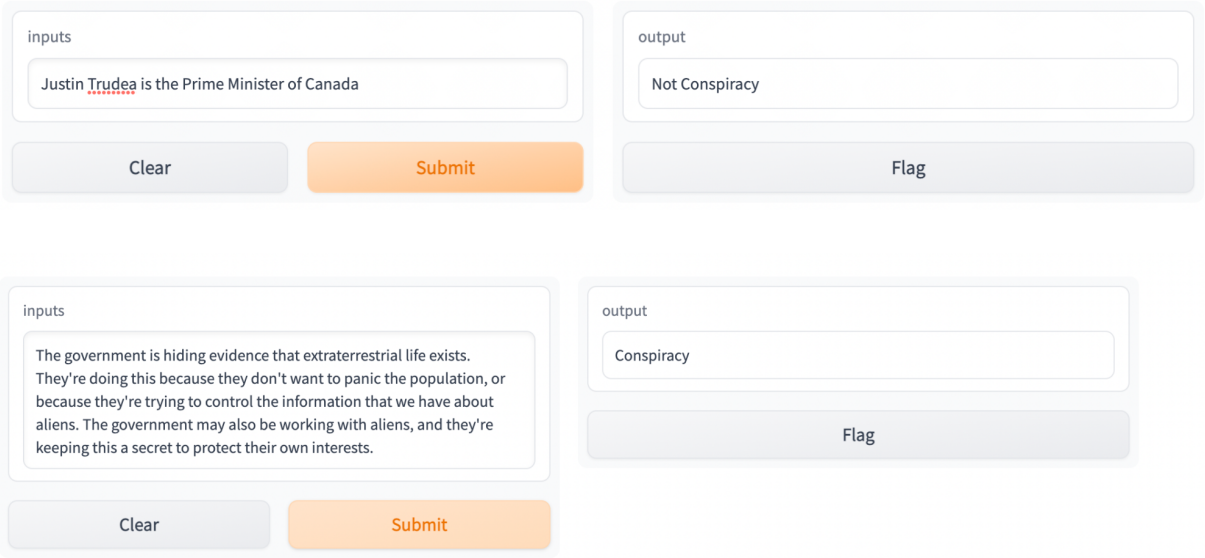
This pre-training procedure means we hide words in a text and have the model guess the output, and tune the model based on if it is correct or wrong.

Using this pre-trained model, we will then fine-tune it on our Twitter database that we scraped. This teaches the model to classify these tweets (and as a result other texts) as conspiracy or not-conspiracy.

To pre-train, we will use the PyTorch and Huggingface libraries

We can further tune model hyperparameters such as loss coefficients and layer sizes to improve the performance. Once we have an acceptable model we will be able to run it on scraped data and extract instances of conspiracy

The next page has images with a UML flowchart for the cleaning and training process. It is not exactly following the class/attribute style in OOP UML charts but it conveys the key points of the programs involved with this section of this experiment. The imports of the python libraries are in the first section of each container and the flow communicates how the classification is handled. The diagram below illustrates the interface I made for the model.



Code/Python Notebook:

https://github.com/ManethKulatunge/COMP400_YoutubeResearch/blob/master/text_classification/Final_Binary_Text_Classification_Conspiracy_COMP400.ipynb

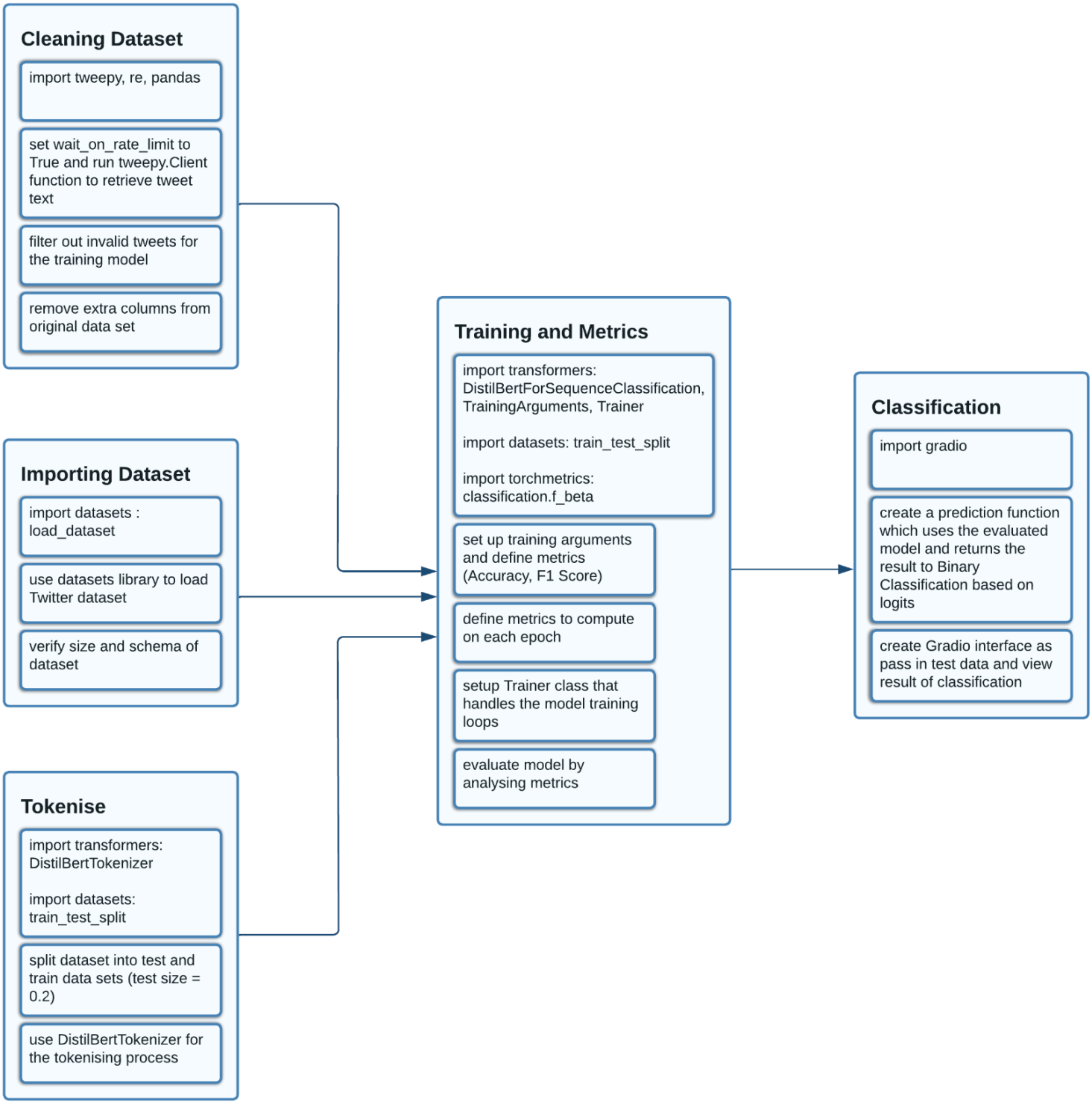
Verification:

Since we want to verify if the Text Classification is functioning as expected, I ran a few tests to see if the intended results are received.

I generate ~50 Conspiracy theories examples using a GPT-3 playground for both positive and negative cases. I labelled each text to reflect if it's considered conspiracy or not according to my understanding. Then I pass this data frame through the prediction/classification function and compare both my prediction to the prediction from the model.

The results for the verification process can be found here: https://github.com/ManethKulatunge/COMP400_YoutubeResearch/blob/master/datasets/classification/final_check.csv

Professor Vybihal gave me feedback about the aforementioned verification method and unfortunately since my analysis for the depth first and breadth first algorithm took longer than I accounted for, I could not document my alternative verification procedure, but I will mention the alternative procedure in Section 6 so that in case someone does take up the project, they could make my training process more efficient.

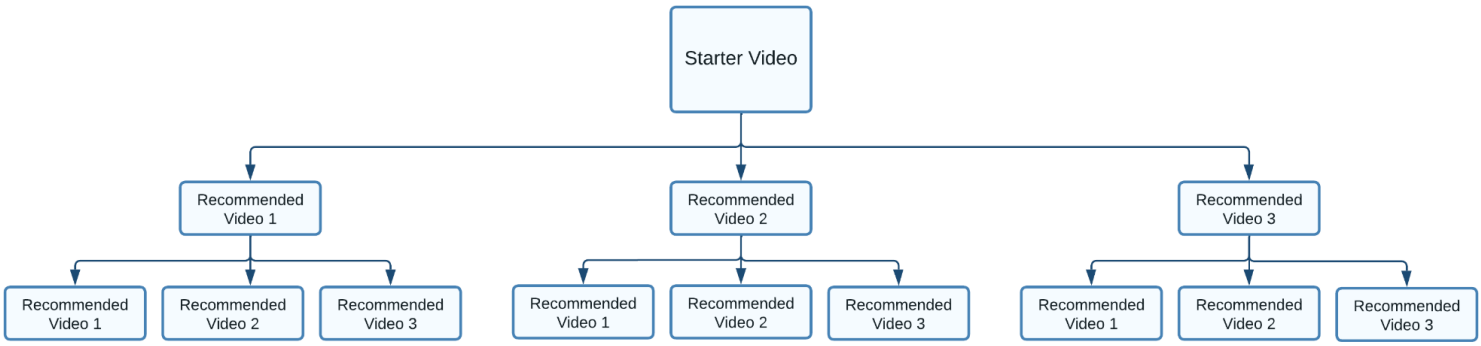


UML Diagram For Text Classification Process

Breadth First Scraper

Description

My scraper algorithms are inspired by Mika Desblanc's research on the Youtube recommendation system. In the breadth-first scraper, the algorithm begins with a video from a seed list that I created which is primarily populated with conspiracy-based videos. For this experiment, the starter video explores initially explores three recommendations. We will visit every child video in that resultant list of recommended videos (the ply) before continuing to the next ply. Each of those 3 videos becomes the root of another expansion level. In this way, the tree expands ply by ply. The algorithm continues until the expansion tree has a depth of 5. My analysis will be exploring each ply and how the proportion of conspiracy-related videos (silo percentage) changes for each recursive expansion.



Flowchart for BFS Video Recommendation Scraper

Implementation

In terms of implementation, I use the Selenium library alongside Python as means of automated scraping. The Selenium Library has a Webdriver which helps us identify elements such as Video Descriptions, Titles and URLs that will be useful for extracting information. I perform an iterative breadth first using a Python list to represent the queue which will hold the URLs of the recommended videos. There are procedures in place to avoid graph cycles and invalid data. The Algorithm utilises the Node class in Python to build out the Breadth First expansion. And this same class can be used to build a level order list where we can access the results for each ply in an efficient manner. A visualisation of such a tree built programmatically can be seen below.

After building out this tree, we need to convert it into a form that is easier for analysis so we convert the tree of recommended videos into a .json file where each key will represent the ply depth and the values will be information related to each recommended video in the ply. Then we move onto the analysis stage where we explore each of these created .json files and find the average proportions of conspiracy and non-conspiracy-based videos by passing all the results of these .json files through the Text classification model we have implemented already. The results for each .json file and saved in a .csv file which will be attached below. As the image shows, we can see the number of videos for each category. Since the seed list consists of ~50 videos we generate 50 .csv files and this enables us to get an average silo to get the average silo percentage per ply.

bfs_1

ply	conspiracy	non_conspiracy
0	1	0
1	3	0
2	8	1
3	20	6
4	59	19

bfs_2

ply	conspiracy	non_conspiracy
0	1	0
1	3	0
2	5	4
3	18	9
4	50	31

The above images are from the generated CSV files and this will be used for our analysis.

Breadth First Scraper:
https://github.com/ManethKulatunge/COMP400_YoutubeResearch/blob/master/breadth_first/scraper_breadth.py

JSON File Generator:
https://github.com/ManethKulatunge/COMP400_YoutubeResearch/blob/master/breadth_first/analysis.py

Breadth-First Search Folder (with resultant files):
https://github.com/ManethKulatunge/COMP400_YoutubeResearch/tree/master/breadth_first

Depth First Scraper

Description

The Depth First algorithm follows a similar pathway to the breadth-first algorithm that we established earlier. The main difference would be that for each ply we take a depth-first approach where instead of experimenting with n number of elements (where n=3 in our actual experiment for breadth-first), we only consider the first recommendation of the Youtube recommendation system, and each recursive ply is built by diving into the very first recommendation. In our experiment, we limit the number of dives to 3 which means that our depth-first expansion tree would be a k-nary tree where k=3 resulting in three branches. However, we will need to keep track of the top 5 recommendations for each ply, including the recommendation that we are exploring depth first because we need to consider how the silo percentage varies per dive. The tree depth would be 6 we will observe how the recommendation change for each level of the tree when taking a depth-first approach

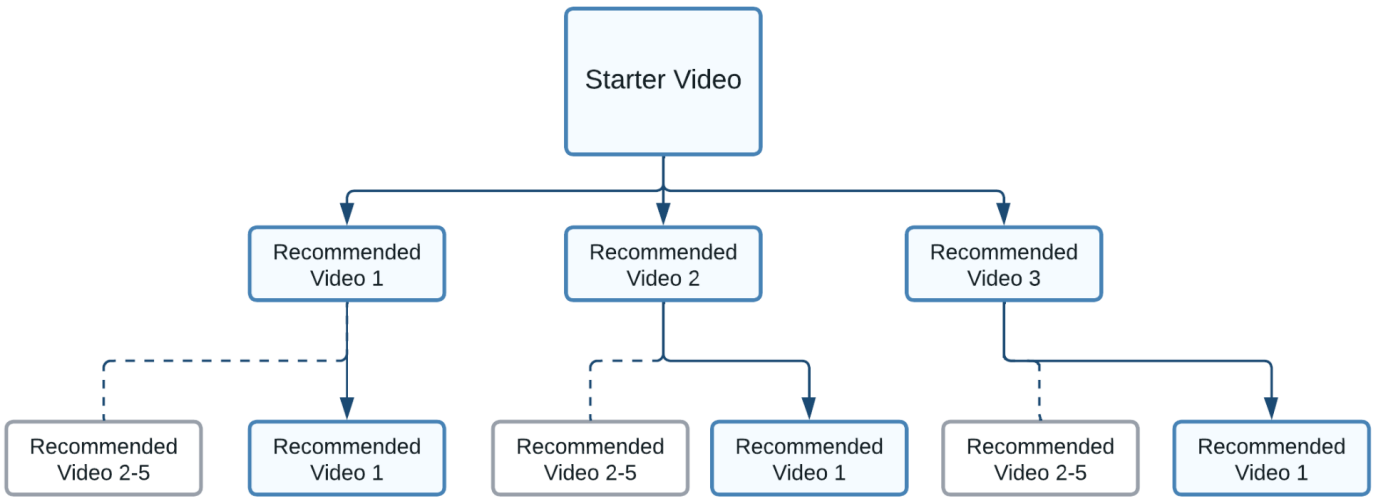
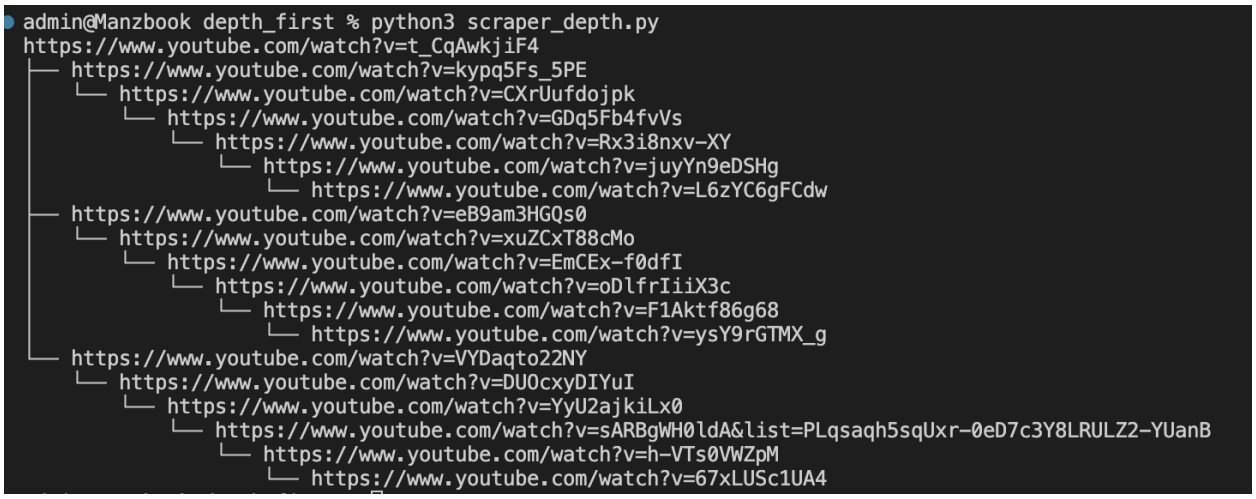


Diagram explaining the Depth First Algorithm for fetching Youtube Recommendations

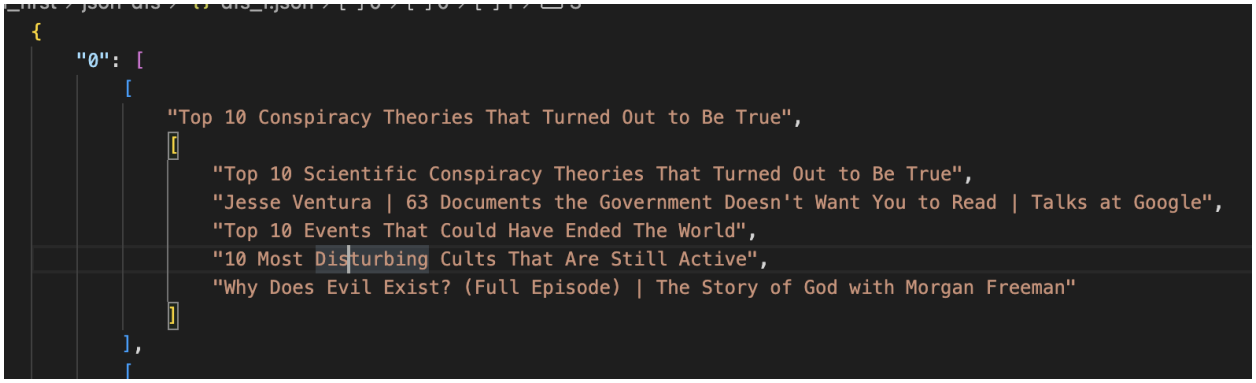
Implementation

The Implementation of the Depth First scraper uses the same libraries being used in the Breadth First Scraper but it was easier to implement a strictly recursive approach for the Depth First Scraper. The base case would be when each explored branch reaches a depth of 6. The first ply is organised in a list and the program recursively populates the resultant expansion tree while keeping track of 5 recommendations per recommended URL in a python dictionary. We make sure that there are no cycles in each dive and that we account for invalid information when we collect data through the Chrome Webdriver. The image below represents the expansion tree that's generated programmatically.



Result of Tree after running Depth First Scraper

The second segment of the implementation for the depth-first scraper is to store the results from the ~50 searches for the starter videos. We once again store these in a .json file where the key for the dictionary would be the number of the dive which will indicate the branch of the depth-first search tree. The tree will also contain the titles of the recommended videos per explored URL. We use the properties of the Node class such as the children attribute to demarcate the dives so that the JSON file can be organised accurately. The image below represents an example of a resultant JSON file.



We will then convert this JSON file to a CSV file for easier analysis allowing us to use dive somewhat as an index to track the proportions of conspiracy-leaning videos per ply of the depth-first expansion tree by passing the metadata of the needed videos through the text classification algorithm I established earlier on. The images below show examples of how the data is being stored in CSV for analysis.

dfs_1

dive	ply	conspiracy	non_conspiracy
0	1	4	2
0	2	3	3
0	3	1	5
0	4	3	3
0	5	1	5
1	1	5	1
1	2	1	5
1	3	4	2
1	4	4	2
1	5	5	1
2	1	6	0
2	2	6	0
2	3	5	1
2	4	3	3
2	5	1	5

dfs_23

dive	ply	conspiracy	non_conspiracy
0	1	1	5
0	2	2	4
0	3	2	4
0	4	3	3
0	5	1	5
1	1	3	3
1	2	2	4
1	3	1	5
1	4	4	2
1	5	5	1
2	1	4	2
2	2	2	4
2	3	2	4
2	4	1	5
2	5	1	5

Depth First Search Algo:
https://github.com/ManethKulatunge/COMP400_YoutubeResearch/blob/master/depth_first/scraper_depth.py

Complete Folder (with the program for prepping result tree for analysis):
https://github.com/ManethKulatunge/COMP400_YoutubeResearch/tree/master/depth_first

Results and Analysis

Breadth-First Search

I made a program for the analysis of the results I received for the breadth-first scrapers where I populate two Python dictionaries with the total number of conspiracy and non-conspiracy related videos after iterating through the CSV files to retrieve the average per ply values for conspiracy and non-conspiracy related videos. The general breakdown of the video count per level is

Level 0:1 Level 1:3 Level 2:9 Level 3:27 Level 4:81

And since we collect data for 50 starter videos we are getting results for a total of 6050 videos [(1*50) + (3*50) + (9*50) + (27*50) + (81*50)].

After receiving the results I can conclude that there is a reduction of the proportion of recommended conspiracy-leaning videos as the ply depth increase. I made two runs on the text classification algorithm to verify these findings and the results turned out to be similar. I have plotted the results below for reference. The list of numbers above each plot represents the average Silo Percentage value per ply depth

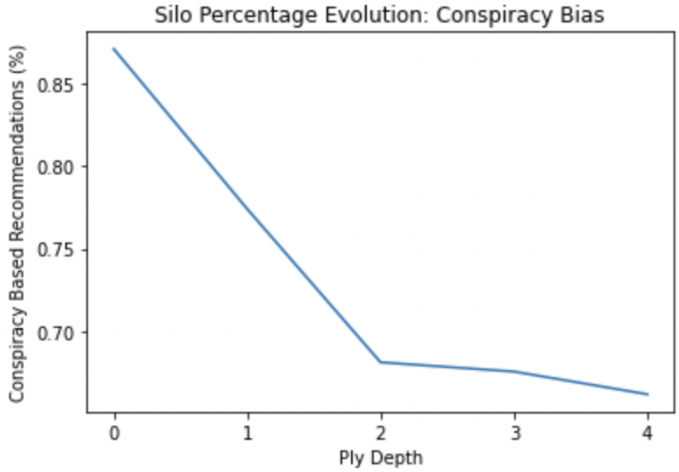
Data:

https://github.com/ManethKulatunge/COMP400_YoutubeResearch/tree/master/breadth_first/csv_results

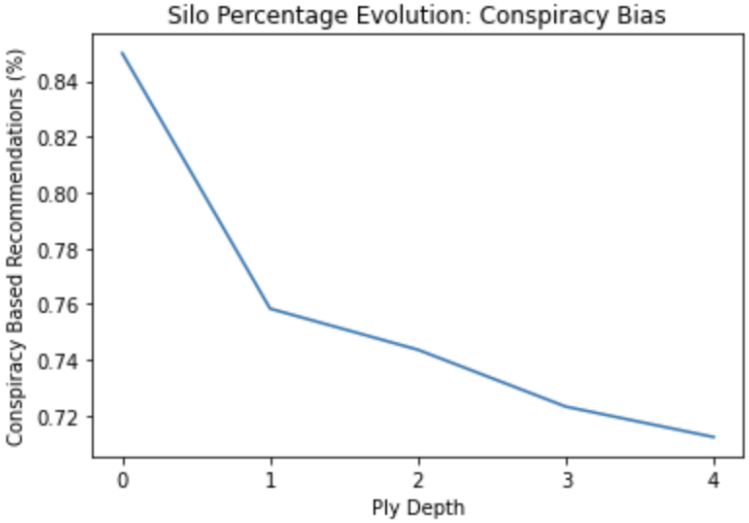
Analysis:

https://github.com/ManethKulatunge/COMP400_YoutubeResearch/blob/master/analysis/Analysis_BreadthFirst.ipynb

[0.8709677419354839, 0.7741935483870968, 0.6813186813186813, 0.6757090012330457, 0.661965989216093]



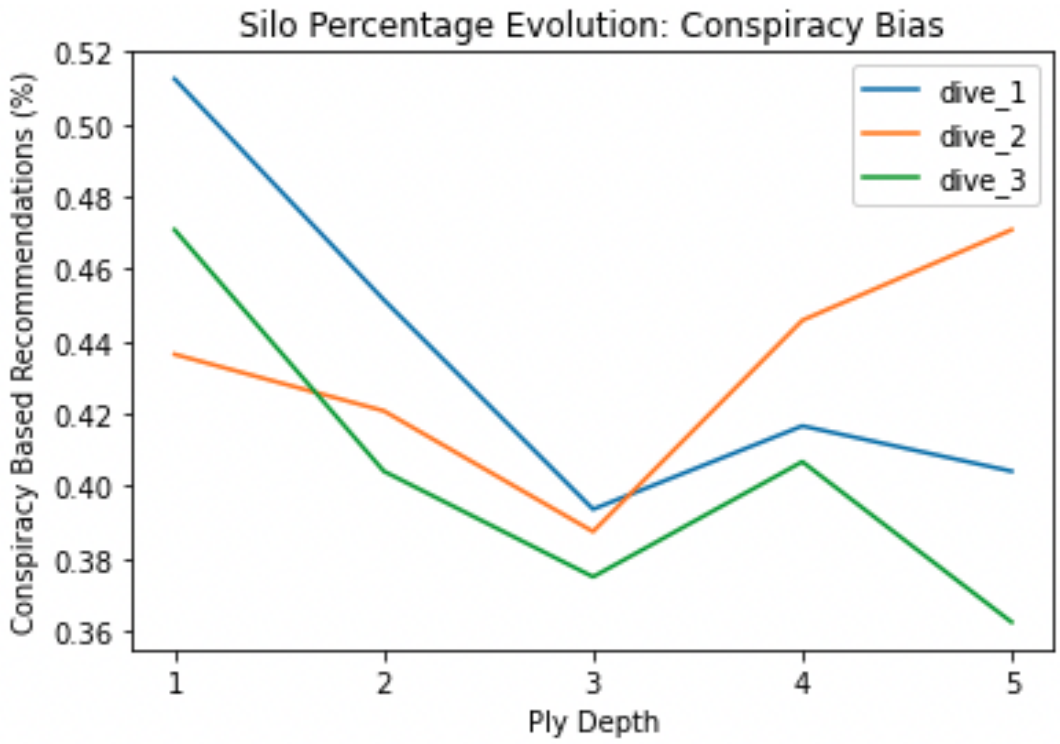
[0.85, 0.7583333333333333, 0.7435897435897436, 0.7231800766283525, 0.712258064516129]



Depth First Search

My analysis algorithm for the Depth First Search algorithm was examining the three dives separately to identify how the Silo Percentages are changing for every ply depth in each dive. I used python dictionaries in a similar way as my Breadth First analysis but I had to be more granular with my approach so I kept track of all the values for each dive and subsequently for the levels of each dive. Essentially had to use a nested dictionary to keep track of the values.

I calculated the averages from the data I collected above and the plot looks for each dive looks like the following:



Interestingly, it seems like the second dive acts slightly like an anomaly compared to the other two dives. But generally, I think it's fair to assess this graph and come to the understanding that conspiracy-leaning recommendations are reducing in percentage as the ply depth increases. There is a small increase between ply depths 3 and 4 but the overall trend of the graph is in an inverse direction. I compared my results to Mika's and they seemed to have a similar trend with the breadth-first search results. Mika noted that the results were probably due to an artifact of their algorithm and I believe this same reasoning should apply to my depth-first algorithm. There needs to be more research done to see if this pattern is persistent.

Data:

https://github.com/ManethKulatunge/COMP400_YoutubeResearch/tree/master/depth_first/csv-dfs

Analysis:

https://github.com/ManethKulatunge/COMP400_YoutubeResearch/blob/master/analysis/Analysis_DepthFirst.ipynb

Project Feedback

What would I do differently

1. I would have populated the seed video list for the breadth-first search with more varied data so that I could have gotten a variety of results. My current seed list is heavily leaning towards conspiracy-based content so the starting silo percentage for the BFS analysis is a little too high which could have been an inefficiency with my experiment

2. I would have improved the verification process for my machine-learning model by picking a subset of the annotated data and passing each record through the text classification algorithm on it to see if the desired results have been achieved for each record. I think that would be a better way to judge if the text classification algorithm has been trained correctly as opposed to generating sentences using GPT-3.

3. I would add more depth to the text classification algorithm and show more data on how the algorithm processed passed text. As of now, it returns a label but it would be helpful to know more in-depth data about the metadata analysis of each video as opposed to a binary classification

4. As of now the text passed into the classification algorithm is the title of the video and the recommended video. I would add more metadata such as description and maybe even comments to get a more holistic understanding of whether a video is conspiracy-leaning or not

What would I do again

1. I would train the text classification algorithm again and maybe try other models outside distillBERT to examine how the classification varies according to the algorithm. I would also more data sources to create the data set used for the training of our model.

2. I would run the breadth and depth-first searches again with different metadata to examine if the results vary. A great start would be the usage of comments instead of or alongside the video titles as this might capture public opinion related to the video's conspiracy bias.

Future Work

1. Make the code base (especially breadth first scraper) more modular
2. Utilise the survey results retrieved by Professor Vybihal to mimic how users utilise the Youtube recommendation system and perform analysis on this data
3. Examine how the home page changes with newly added/removed videos
4. Improve verification process for ML model