**Big Data – Group Project – Team 5**

**Project Phase II:**

**Introduction**

In the growing age of social media discussion, where everyone and anyone can post their thoughts instantly to the masses, it can be tricky to avoid spoilers for the next episode of your favorite tv show or the newest big blockbuster superhero franchise film. People go so far as to completely go off the grid, cutting themselves off from friends and family by throwing their phones in a lake with a cinderblock or simply deleting Twitter for a few days. This creates a problem for Twitter, as when huge pop culture events take place, they may inadvertently see a decline in user activity (though perhaps followed by a spike once enough people have seen, for example, the latest Marvel movie and feel comfortable diving into discourse).

There becomes a need for the average entertainment media enthusiast to find a way to avoid spoiler content while staying connected online. As of today, the technology does exist on Twitter, but it isn’t as simple as clicking a button. However, it could be. Over the course of this report, we will outline the steps we took to develop a new proposed feature on Twitter called “Spoiler Alert” that can automatically flag and filter potential spoiler content from a user’s social media feed.

Before collecting our data, **we hypothesized a few theories as to where the most common sources of spoilers on Twitter would lie.** We **narrowed it down to three main locations**: within the text of a tweet, in the comments of a tweet (“retweets” and “mentions”), and the attached images, videos, and links of a tweet. Tweeters may not always post a spoiler warning, and tweets that become popular are more likely to show up in a user’s feed. Popular tweets may even be retweeted multiple times and spread like wildfire, and those retweets may include further commentary that may lead into spoiler territory as other users provide their own opinions. While comment restrictions by the original tweeter may limit this issue, that feature is not always utilized. Avoiding posts that may naturally lead into spoiler discussion may not always be enough, as “internet trolls” are prone to leave spoiler comments on completely unrelated tweets. Lastly, images or videos in tweets may depict “spoiler” scenes such as surprise cameos or character deaths.

**Data Process**

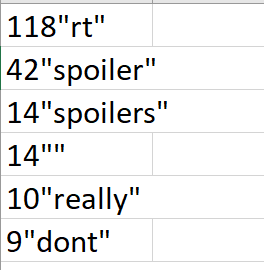
Our data collection, analysis, and implementation process consisted of two branches. The first step of both was to collect the data from Nifi with the Twitter API. Twitter’s popularity as a social media platform, while providing a wealth of data, also came with a challenge. To combat this issue, we decided to filter out irrelevant tweets with a regex expression before saving the remaining data to a csv file. For the purposes of our business problem, we agreed that filtering for tweets that contained the word “spoiler” would leave us with data most likely to contain spoiler content. For regex simplicity, we turned off the filter’s whitespace and case sensitivity. This means that both tweets with “spoiler” and “SPOILER” in them would be kept. Later, when we wanted to cast a wider net to analyze spoilers from specific media, we added regex expressions based on the media name (Ex. The newest Marvel show, “Moon Knight,” we filtered on tweets containing “moon knight”). For versatility, we kept any tweets that adhered to at least one of our key words. Doing this meant that we could collect Moon Knight spoilers that did not explicitly label themselves as spoilers, as well as spoiler content from other popular content non-Moon Knight related. Later, when we ran a neural network to predict spoiler tweets, this would hopefully prevent the model from overfitting for explicitly Moon Knight spoilers (we later tested our model with spoiler tweets from the latest Harry Potter spin-off movie).

Once we collected our desired tweets, the first thing we did was run the data through a few different MapReduce programs. For example, we ordered the words in all the tweets from most frequent to least frequent. The code below took unlabeled data and found and sorted the words by frequency count.

Text

Description automatically generated

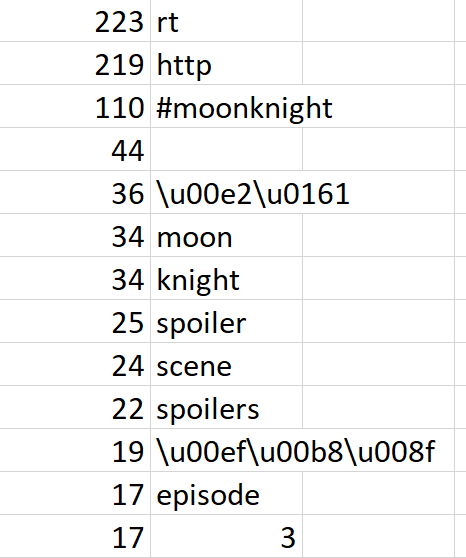
While running these programs, you’ll notice our MapReduce scripts also cleaned the data by removing unnecessary punctuation and converting everything to lowercase. This would help MapReduce aggregate words that are realistically the same but may have been left separate because perhaps one is uppercase or the other has a question mark at the end. We also converted any word with “http” in it to “http” so we could aggregate all the links together to see how often a tweet contained a link at all. Before doing this, each link was unique and therefore at the bottom of our sorted list. Our last step was to remove filler words such as “the” and “and”. We figured these words would be common in any tweet and would clutter our Top-N ranking while not providing any distinction between spoiler and non-spoiler tweets. Once the data was cleaned, the output of this code when run through a docker container and sent to a csv file looks like this:



We used a similar code to find the top words of labeled data both for Marvel’s Moon Knight and Fantastic Beasts, looking at the snippet of code, you’ll notice the only difference is we’ve filtered to only count words in tweets labeled as spoilers by adding an extra if statement.



The output csv file looks similar but here the words are guaranteed to be from spoiler tweets rather than just tweets containing the word “spoiler”. There are some issues such as null values and non-word output that may be due to unfamiliar characters or emojis.



We also found the percentage of tweets that were retweets and the percentage of tweets that contained images and links. We could then compare those values to the percentages from a datafile containing an unfiltered collection of twitter data. The code below was for unlabeled spoiler data which finds the percentage of links, which we ran for both miscellaneous data (pictured below) and for our filtered Nifi output data containing the word “spoiler”. You’ll notice here cleaning up the text was unnecessary as the only thing that needed to be verified was if the tweet contained “http”. The final output values are calculated by keeping a running count of both the total tweets in the dataset, as well as a count of just the tweets with links. Total links are divided by total tweets, providing the final percentage value.

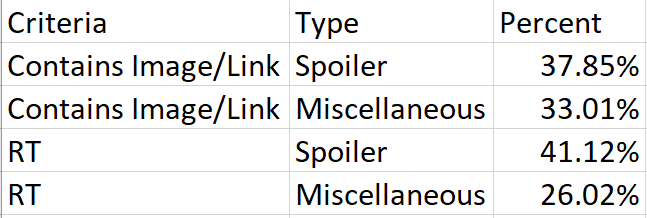
Text

Description automatically generated

A similar code was run to count retweets, however instead of counting if “http” was in the text, we found if the first two characters of the tweet started with “RT”. Aside from that, a few variable names and output string labels were changed but the code was fundamentally the same.

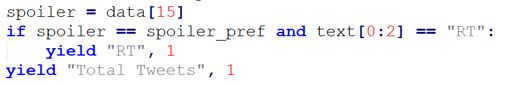


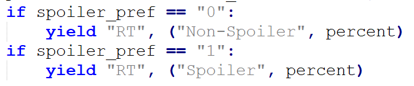
We ran each of these files on our spoiler and miscellaneous data and appended our results to the same csv file from the docker command prompt.



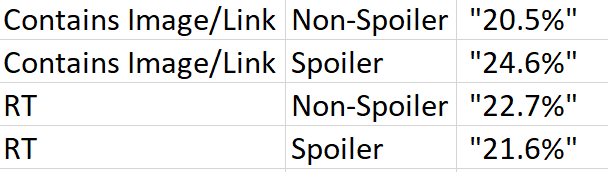
Ultimately, this comparison supported our hypothesis that “spoiler” tweets are more likely to contain images and links, as well as more likely to be retweets. We also built versions of these two scripts for the labeled data, this time outputting values for either the “spoiler tweets”, should the value of the “spoiler\_pref” variable be set to “1”, or “non-spoiler” if the variable is set to “0”. Depending on this variable, we filtered the tweets differently, and labeled the output percentage differently. The added lines of code are below for the retweet script.







The output for these scripts appears when written to the same CSV file appear as follows.



**Data Visualization**

Once we had appropriately aggregated data, we could send it to Tableau to visualize the Top-10 words and their frequencies in spoiler tweets for Moon Knight and Fantastic Beasts (the Harry Potter movie). We also ran the data through a python script to reformat it for a word cloud generator which creates a cluster of words whose size depends on their frequency.

Graphical user interface, text, application

Description automatically generated

This python script takes the MapReduce top-N data and essentially uses the word key and the count to build a list of words repeating the number of times given the frequency. The output csv file looks as follows.



Unsurprisingly, the most common words were “http” and “rt”, followed by “spoilers” and “moonknight” (however, we filtered tweets based on these criteria, so this is expected). Other common words were character names, episode numbers, and scene references.

The Top-10 words for the Fantastic Beast data was less focused on key words from the movie title (albeit still present) as the movie having just released to lackluster review has had limited buzz leading to limited viewership. Furthermore, unlike a show on streaming, screenshots are harder to capture making photo spoiler leaks much less likely to circulate.

**Data Implementation**

After analyzing our data and confirming our hypotheses, we could then take this information and proceed to implementing the data in ways that could solve our business problem. The first step was to see how our learnings worked with Twitter’s current filtering tools. As of today, there are two main methods to filter tweets, however for the greatest effect, we recommend they be used in conjunction. The first is a feature called “Muted Words”. Users can specify words they don’t want to see in their notifications or main feed, and any tweet containing these will be filtered out. They can even set a time limit on this filter, such as 24 hours or days at a time. However, these words must be specified one at a time, and will not affect searching. This brings us to the second method, “Advanced Search”. Users can specify key words they want to see in tweets as well as key words that must not be in tweets. They can even select to not include retweets or tweets with images or media. For example, we filtered on the words “Moon”, “Knight”, and “Review” while avoiding words like “spoiler” and “spoilers” as well as retweets and media. This was rather effective. However, in order for a user to do this, they must already know Twitter’s advanced search syntax or have already submitted a search for the user-friendly interface to appear. In the case of the later, it may be too late by then to avoid spoilers.

This dilemma motivated our final process: creating a recurrent neural network to predict when a tweet is a spoiler or not. In order to do this, we needed labeled data. This would have been incredibly tedious even for just the 1000 tweets to which we limited our train-test dataset. A final model ready to integrate into Twitter’s infrastructure would need to be run with a much larger dataset, but for the sake of time and simplicity we decided this was appropriate for a demo. We built an algorithm in our excel CSV file which would evaluate each tweet for a link. Tweets with links could be opened automatically so a user could properly evaluate if the link or media attached contained spoiler content. Without this link, a message box containing the text of the tweet would be displayed.

Graphical user interface, text

Description automatically generated

Following this, a user form prompted the user to determine if the tweet was a spoiler or not and their selection would automatically append a “1” for spoiler or “0” for non-spoiler to the label column of the CSV file.

Table

Description automatically generated

The labeling algorithm loops through each tweet until every single one is labeled. Finally, this data is run through the neural network. The code for this is as follows. Similarly, to the sorted word count files, we must clean the tweet data to remove punctuation, turn everything to lowercase, and reformat links. This will hopefully ensure the model trains smoother as the data is more standardized and easier to comprehend and compare. The tweets are then embedded using “glove” and then split into training and validation sets.

Text

Description automatically generated

Once this data is prepared properly, we can build the model, run it with our data, and then plot a few charts to visualize the model’s performance.

Application

Description automatically generated with low confidence

After several runs, we determined that the average accuracy of validation data was around 80% (a minimum around 75% and a maximum around 85%).

Chart

Description automatically generated

Chart

Description automatically generated

There was one caveat, however, as both the training and validation data consisted of a blend of any moon knight tweets and other spoilers. This model was potentially overfit to function for moon knight spoilers specifically (this television show was chosen because 1. Marvel shows are extremely popular and the most likely source of spoiler content; 2. The newest episode premiered far enough in advance to build and workshop our neural network). Therefore, we decided to use the last batch of data we collected (tweets filtered on the Fantastic Beasts movie) to retest the models predictive accuracy.

Perhaps unsurprisingly, this brand-new data performed worse. While the total accuracy was at 75%, around the lowest the models tested with the split moon knight data, true spoilers were only predicted accurately around 15.25% of the time.

Chart

Description automatically generated

**This means that the ability of this neural network to hypothetically function as intended (accurately flagging spoilers) is lackluster**. However, this can be partially attributed to limited time and limited data. This exercise is intended as more of a demonstration of what is possible rather than perfected execution. Other reasons for this underperformance may be an overfit model. This is especially troublesome because the trends of entertainment discourse are constantly changing and given the tendency of key spoiler words to be topic specific, such as character names, there is limited overlap between popular content. Ultimately, more work must be done to improve the model’s dependency.

This model, while certainly not ready to be onboarded to Twitter, at least proved that an automatic spoiler filter is feasible, which leads us to “Spoiler Alert”. We propose with a more extensive collection of data and neural network training, Twitter could provide this new feature allowing users to avoid spoiler content without having to give up social media, and most importantly Twitter, altogether. With the press of a button, users could transform their feed into a virtual safe haven. Any photos the algorithm determines to be risky will be flagged and censored. Any comments from internet trolls will be removed from sight, and, once the user has stepped out of the theater and is ready to dive into discourse about obscure comic book easter eggs or shocking plot twists, all they need to do is press the button again and their feed is back to normal, no hassle. That’s why we believe “Spoiler Alert” is the future of Twitter.