Critical Role Fan Art of the Week Predictive Model

S Smolenski

Introduction

"Critical Role is an American web series in which a group of professional voice actors play Dungeons & Dragons" [1]. With over half a million fans, the D&D livestream has caught the imagination of a vast number of artists. Every day, hundreds or thousands of pieces of Critical Role fan art are shared on Twitter. Each week, the Critical Role team chooses one piece of fan art to be showcased on thier talk show and their twitter; they also award the artist with merchendise from either their store or from their sponsors.

I have endeavored to create a predictive model, which, based on time-series data of likes and retweets for a particular tweet, determines the likelihood that it will be chosen as Fan Art of the Week. Unfortunately, data collection has introduced some difficulties in the process. As of yet I have not been able to collect a data set suitable for the training of a machine learning model.

Method and Procedures

My initial goal was to collect those tweets that have been declared Fan Art of the Week (FAW) from the Talks Machina twitter account and extract likes and retweets from those tweets. These would be "true" examples. For the "false" examples I would select a random assortment of fan art tweets from the same week and extract their likes and retweets.

However, this idea had several problems. The first being that, as soon as a tweet is declared FAW and shared by the Talks Machina twitter its likes and retweets skyrocket. If I collcted the current number of likes and retweets from all declared FAW tweets for the past two years, they would all have a much higher tweet and retweet count than they had before they were chosen. In order to train a predictive model I would need to know how many likes and retweets the FAW tweet had before it was chosen.

Since there is no way to tell which tweet will be chosen (barring the predictive model I am attempting to train) this would require that I know how many likes and retweets *every fan art tweet* submitted in the past week had. Once I had that information, I could retrospectively mark a particular tweet as FAW, thus acquiring a single "true" example.

Time quickly becomes an important factor: to gather enough "true" examples to train a predictive model, a script would be required to scrape twitter for months or years.

But, there being no time limit to this project, that is one of the lesser problems I faced. The next being that I would need to scrape twitter every single day throughout that time in order to ensure I found every FAW tweet before it was chosen (this estimation of once a day turned out to be very optimistic). Twitter's API has a download limit and, given the sheer magnitude of the number of fan art tweets, it would be impossible to scrape all the tweets from the previous week.

My script would thus be required to:

1. Scrape Twitter every day
2. Clean and store the relevant tweets
3. Search twitter for all previously stored tweets
4. Store the updated likes and retweets count as a new point
5. Repeat this for months or years

However, when I had built a script that would perform these steps, I found that most weeks on Tuesday when FAW was announced, the relevant tweet was *not* stored in my data. Thus, downloading tweets once per day was insufficient: I had gaps in my data.

I updated the script to run multiple times a day, going so far as to begin scraping on Thursday night (when the live show is broadcast) and scrape more often on Friday, when I expected more art to be released in th wake of fresh material. Yet still I encountered the same problem. In spite of frequent scraping, my data still contains holes. I suspect this is due to th fact that my computer is not left on at all hours and therefore unable to scrape Twitter indisctiminately every few hours, day and night. As of the time of this writing, I have not found a convenient solution for this.

Data

The main file is CRScrape.R, which calls all othr functions. Initially it loads in existing data (or, if there is none, creates a new data frame).

I include piece of the code below, but none of it will execute. This is partially due to the amount of data and the associated time required to run it, and partially because the bulk of the script requires connection to the TwitterAPI, for which I will not include my key.

```r
if(file.exists("data.Rda")){
    cat("Loading existing data file...\n")
    load("data.Rda")
    lastid <- as.character(data$ID[[nrow(data)]])
    lastdate <- as.Date(data$Date[nrow(data)])
    if(lastdate!= date){
        cat("Updating tweet data...\n")
        data <- Update(data)
    }
}else{
    cat("Creating new dataframe...\n")
    lastid <- 0
    data<-data.frame(   "ID"=character(),
                        "User"=character(),
                        "Date"=as.Date(character()),
                        "Likes"=numeric(),
                        "RTs"=numeric(),
                        ...
                        FAotW=logical())

}
```

Note that there are several unused columns in the data frame, each one labeled with the name of an actor on Critical Role. These are for an extension to th project, which I have not yet developed, and will be discussed in the “Future Work” section.

It then calls NewTweets, with a tweet limit set as the last Tweet ID in the existing data frame (if there is no previous data, its limits are set by the NewTweets function). In other words, provided the script runs frequently enough, it will scrape all tweets posted between now and the last time the script ran. If it does not run frequently enough, the number of tweets is limited (both by NewTweets and, eventually, by the Twitter API itself).

```r
cat("Retrieving new tweets...\n")
new <- NewTweets(lastid)
data <- rbind(data,new)
```

Finally, every Wednesday the script will find the tweet that was declared FAW and compare the ID number to the existing data. If the tweet is saved in the data frame, it sets the value “FAotW” to “TRUE”

```r
if(weekdays(date)=="Wednesday"){
    cat("Getting FAotW... \n")
    ID<-FAotW()
    data$FAotW[which(data$ID==ID)]=TRUE
}
```

Somewhat inefficiently, this will run every time the script is run on Wednesday. Thus if the script runs three times a day, this portion will be run redundantly twice. This should be fairly simple to fix, but I have not done so yet.

Finally, the script saves all the updated data, overwriting the previous data file if it was loaded in or creating a new one if this is the start of a new set.

Acquisition: NewTweets.R

NewTweets connects to an external function TwitConnect, which simply conncts to the API. It then searches Twitter for two hashtags: `#criticalrolefanart` and `#criticalroleart`. It will either download as many tweets as possible until it reaches the where it left off last time or 5000 of each hashtag.

Cleaning: NewTweets.RTs

NewTweets then cleans the data and compresses the Tweets into a useable form. First it removes all downloaded tweets that were retweets so that the only remaining are the original posts. `strip_retweets()` is a function in the `TwittR` package.

Next it checks whether or not each tweet contains an attachment using `withArt()`. Admittedly, this method is not foolproof, as I only check whether the tweet contains a custome Twitter-shorted url.

After that, NewTweets strips the cosplay tweets. As many people tag their cosplay pictures as Critical Role art, `withCos()` checks whether or not each tweet also contains the tag `#criticalrolecosplay`. Those tweets that return `TRUE` are removed from the data.

Finally NewTweets unpacks the relevant data from the tweets, extracting Tweet ID and Username from each tweet and storing them in a data frame. This information is all that is required for the next step.

XML Scraping

The TwittR package does not include any convenient way to determine how many likes or retweets a tweet has. Instead, I parsed the html into raw text, which the functions `getLikes()` and `gtRetweets()` scrape for the number of likes and retweets, respectively.

ie, from `getLikes()` this looks like:

```r
doc <- htmlParse(rawToChar(GET(url)$content))
    xpathSApply(doc,"//li[@class='js-stat-count js-stat-favorites stat-count']",xmlValue) %>%
    str_extract("[[:digit:]]+,?[[:digit:]]*") %>%
    gsub(",","", . ) %>%
    as.numeric() -> Likes
```

The Predictive Model

As mentioned above, so far I have been unable to collect sufficient data to train a model on. The data set would be heavily skewed toward "not FAW", which also presents potential problems.

Furthermore, there would be at least one more choice to make before training was done. By necessity the data collected is time series. The simplest option would be to train a model only on the number of likes and retweets immediately before FAW was announced so that the model has only two parameters. This may or may not turn out to be too simplistic, but I would see what sort of precision and accuracy a model trained off this data alon could achieve.

As it is a binary classification, the simplest option is to train a logistic regression model, which may work as a decent baseline. (Note that, if I were to to train multiple models and choose the best, I would split the data with sufficent data to train, choose a model, and then test. This would obviously require more data and thus more time).

References

[1] "Critical Role." Wikipedia, Wikimedia Foundation, 5 Aug. 2019, en.wikipedia.org/wiki/Critical_Role.