CS4624 Multimedia/Hypertext/Information Access

Final Project Report

Spring 2018

Tweet URL Extraction

Carter W. Tat
David C. Chun
Christopher A. Bridges

Department of Computer Science
Virginia Tech
Blacksburg, VA 24061

Date: 5/2/2018
Instructor: Edward A. Fox

Table of Contents

Table of Figures

Table of Tables

1. Executive Summary

In this report we document our work on the tweet URL extraction project for CS4624 (Multimedia/Hypertext/Information Access) during the spring 2018 semester at Virginia Tech. The purpose of this project is to support our client Liuqing Li with his research in archiving digital content, part of the Global Event and Trend Archive Research (GETAR) project supported by NSF (IIS-1619028 and 1619371). The project requires tweet collections to be processed to find links most relevant to their respective events, which can be integrated into the digital library. The client has more than 1,400 tweet collections with over two billion tweets and our team found a solution that used machine learning to deliver event related representative URLs.

Our client requested that we use a fast scripting language to build middleware to connect a large tweet collection to an event focused URL crawler. To make sure we had a representative data set during development, much of our development was centered around a specific tweet collection, which focuses on the school shooting that occurred at Marshall High School in Kentucky, USA on January 23, 2018. The event focused crawler will take the links we provide and crawl them for the purpose of collecting and archiving them in a digital library/archive system [1].

Our deliverables contain the following programs: extract.py, model.py, create_model.py, and conversion.py. Using the client's tweet collection as input, extract.py scans the comma separated values (CSV) files and extracts the links from tweets containing them. Because Twitter enforces a character limit on each tweet, all links are initially shortened. Extract.py converts each link to a full URL, and then saves them to a file. The links at this stage are separate from the client's tweet collection and are ready to be made into testing and training data.

All of the crucial functionalities in our program are supported by open source libraries, so our program did not require any funds to develop. Further developments of our software could create a powerful solution for our client. We believe certain functions of our code could be reused and improved upon, such as the extractor, model, and the data we used for testing and training.

2. Introduction

The internet has over 1.5 billion websites, 200 million of which are active today [2]. To allow the client to collect and organize content related to a major event, the client requires a solution to archive tweets. Twitter is one of the biggest social media sites, with 11 million daily users [3]. Our client, Liuqing Li, requested that we write software that will support a much more ambitious project (Global Event and Trend Archive Research (GETAR)), that aims to archive web pages related to major events. Our group wrote a program in Python that extracts links from a massive tweet collection. Given a comma separated values (CSV) file, it extracts shortened URLs, converts them to full URLs, and saves them to a file. We used Python as our main implementation language since it was requested by our client.

In this paper, we first discuss the objective and problem specification by detailing the process we followed to finalize our solution. We describe the requirements, problem constraints, and detailed design specification. Further, we discuss the implementation process which covers each of the program's components. We then discuss how we tested, evaluated, and assessed the solution, which builds upon new machine learning concepts we learned this semester.

For the input that we received from the client, we wrote a program that went through every link in the collection and transferred them to a separate file for ease of processing. We observed that all links in the collection are shortened so they must be converted to their true URL. With the links in another file, we manually labeled them as relevant or non-relevant links. The relevant set includes links that are related to the Kentucky School Shooting while the non-relevant set includes links that are not related to the event. These links are important for the latter stage where model.py uses them to test and train the model.

Once we created the data for testing, model.py takes a corpus and tokenizes each document. TF-IDF stands for Term Frequency Inverse Document Frequency which measures importance instead of just frequency of words [4]. We used the built-in feature in scikit [5] to find the TF-IDF value for each URL.

We wrote a model.py program that integrates the previous component and also uses a classifier. The classifier is trained on the data from the previous steps, and is tested using cross-validation. Cross-validation is a technique that is used to evaluate different classifiers by repeatedly randomly splitting the data into k-equal subsets and testing along the way [6].

Once an effective classifier was selected, we provided the client with a Python program that utilizes the trained model to predict relevant URLs from a tweet collection. The final product allows the client to either input a single URL or a list of URLs from a file.

3. Objective

The objective of our team project is to provide our client with a tool that will classify and rank URLs based on estimated relevance to a given topic. Our deliverable will be written in Python. The application will demonstrate our knowledge of Natural Language Processing and Machine Learning practices.

4. Problem Specification

We need to create a tool for our client to extract URLs from various tweets within a given topic. These URLs will then be input into another tool called the Event Focused Crawler, which builds a collection of webpages about an event, given a list of seeds (i.e., URLs of webpages relevant to the event). Our project will help filter out URLs that are not relevant to a particular topic.

5. Requirement Specification

This section specifies the functional and nonfunctional requirements under which our software project application will be developed.

5.1 Functional Requirements

URL Extraction

1. The system shall have a component written in Python to parse through the tweet collections and extract all URLs from the tweets.
2. The URLs extracted must be a valid URL and will detect error status codes such as a 404 HTTP response.
3. The URLs must be converted from the shortened URL into the true URL which contains the domain name and relevant information within it.

Text Extraction

4. The system shall have a component that visits each linked webpage and extracts relevant text from within the web page.
5. The system shall detect sentences using the Natural Language Toolkit from Python.
6. The system shall also lemmatize words to clean up and refine our data.

Machine Learning

7. Each article must be represented as a vector to feed as input to the training model.
8. The model must be as efficient and accurate as possible. Therefore, many models will be used for testing and analysis.

Data Labeling

9. All URLs extracted must be labeled as either a “relevant” or a “not-relevant” link.
10. A list of relevant links shall be yielded after the program finishes.

5.2 Non-functional Requirements

1. The system shall be developed in Python 3 with maintained documentation for the client to follow along.
2. The software system shall be usable across all platforms including Windows, Apple, and Linux.
3. The system shall be developed and implemented without any paid services or applications and must use only open source software.

6. Design Specification

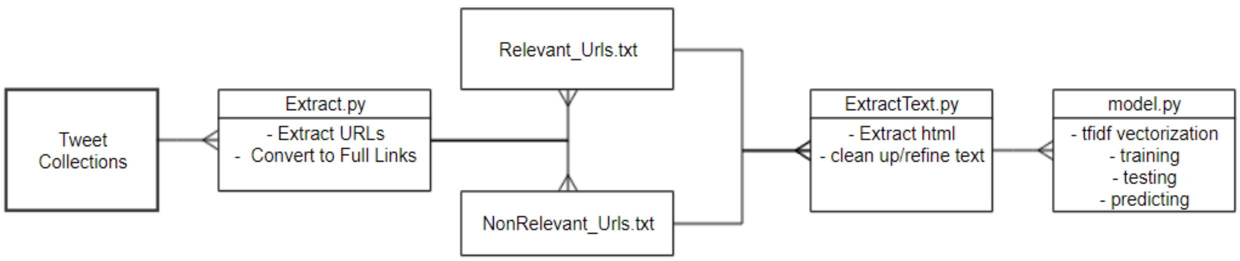

Figure 1. Design Diagram

Our initial design and its components are shown in Figure 1. The flow of data starts from the tweet collections and moves from left to right with arrows indicating connected components. From these tweet collections, we need to extract the URLs from each tweet. Since Twitter only handles 140 characters (280 starting in 2017) in tweets, most URLs were shortened, therefore we need to convert these shortened links into the full links. Next, we have to classify these URLs as either “good” or “bad” depending on whether or not the information on the webpage is relevant to the topic. This data will later be used to train the machine learning model to predict whether a URL is “good” or “bad”.

After we classified the links, we have to represent each URL’s webpage as a vector of values ranging between 0 and 1. To do this, we have to extract all the text from the webpage and apply natural language processing to filter out and clean the data. We shall use the Natural Language Toolkit in Python[7] to detect sentences, remove stop words, and lemmatize other words. After that is done, we will create a vector for each article using the Tfidfvectorizer class from scikit learn. This will essentially be fed into the machine learning algorithm to train and test the data. We will be testing different classifiers such as the decision tree classifier, support vector machine classifier, and random forest tree classifier. After that is done, we are going to evaluate and analyze different metrics to determine the best classifier for predicting a relevant URL vs. an irrelevant URL.

7. Implementation

All components created as part of our project are separate scripts written in Python 3. There is a detailed description for each file.

7.1 extract.py

Tweet collections come in a variety of formats. When working on the code with some given examples, we encountered tweet collections contained within both JSON and CSV files. As such, we decided to make our URL extractor capable of parsing and extracting URLs from both formats. Shown below in table 1, is a summary of the total number of URLs collected. We started with a total of about 27,000 tweets and about 10,000 of those tweets had a URL. After all the duplicate URLs and URLs with error codes have been removed, we were left with about 2,500 URLs. Lastly, we generated about 500 more URLs discussed later on in the report to have a total of about 3,000 URLs.

Total Number of Tweets	Total Number of Tweets with a URL	Total Number of URLs After Filtering (Removing Duplicates and Error Codes)	Final Total Number of URLs
27,251	10576	2,514	3133

Table 1. Summary of Number of Tweets

For CSV files, the first problem that we had to solve was how to extract all the URLs from a collection of tweets. Figure 2 shows a sample of what a raw CSV file of tweet collections from the Kentucky School Shooting event looked like. The CSV file also included other metadata fields as columns such as: the username, language, and the date and time of the tweet.

RT @piersmorgan: BREAKING: At least 2 dead 19 wounded in mass shooting at #Kentucky high school. Suspect is 15-yr-old student who will beâ€¦	Afam_Isol	9.56E+17	1.44E+08	en
RT @nytimes: The school shooting in Kentucky on Tuesday was at least the 11th so far this year â€" and the deadliest https://t.co/EUFimkkzoM	Had2GoTl	9.56E+17	19983237	en
RT @shannonrwatts: My God. Two Americans were killed and 15 were injured in a shooting today at a Kentucky high school. What is wrong withâ€¦	b0mbth3t	9.56E+17	4.7E+09	en
RT @shannonrwatts: My God. Two Americans were killed and 15 were injured in a shooting today at a Kentucky high school. What is wrong withâ€¦	worldwid	9.56E+17	16867624	en
RT @RemadnaWKRN: .@MauraSirianni and I will be in Marshall County Kentucky with continued coverage this morning after 2 students were killeâ€¦	WKRN	9.56E+17	16426346	en
RT @shannonrwatts: My God. Two Americans were killed and 15 were injured in a shooting today at a Kentucky high school. What is wrong withâ€¦	kstrebeljc	9.56E+17	1.67E+09	en
RT @JamilSmith: Itâ€™s January 23 and there have been 11 shootings on school property this year. â€œWe have absolutely become numb to these kiâ€	mahavast	9.56E+17	2.56E+09	en
RT @ValentinaInLA: Sparatoria in una scuola del Kentucky. Una vittima e diversi feriti. #BreakingNews high school shooting leaves at leastâ€¦	StellaSiric	9.56E+17	3.61E+09	en
RT @mattmfm: The Prime Minister of Canada tweeted out condolences about the school shooting in Kentucky before the President of the Unitedâ€¦	_chiefmoi	9.56E+17	88844012	en
RT @JonRiley7: The Kentucky school shooting is the 11th school shooting of 2018. Itâ€™s the 50th school shooting since the beginning of the sâ€¦	unStunne	9.56E+17	1.72E+08	en
RT @CyrusToulabi: My heart is w/ the victims & families of the KENTUCKY SCHOOL SHOOTING. KY GOP oppose common sense #GunControl:- Gov.â	LisaHause	9.56E+17	7.37E+17	en
RT @eugenegu: Was scrubbed into a kidney transplant when I heard several nurses prepare for the high school shooting victims helicoptered iâ€¦	MelindaT	9.56E+17	7.81E+17	en
Why has our own president said NOTHING about this mass shooting at a Kentucky school? https://t.co/kekZVbIqyn	susanash	9.56E+17	20465612	en
RT @shannonrwatts: Zero school shootings in Canada so far this year. Ten in America. Meanwhile our own President hasnâ€™t mentioned the shâ€¦	Siadasha	9.56E+17	8.24E+17	en
RT @shannonrwatts: My God. Two Americans were killed and 15 were injured in a shooting today at a Kentucky high school. What is wrong withâ€¦	Karefnp	9.56E+17	2.2E+09	en
.@MauraSirianni and I will be in Marshall County Kentucky with continued coverage this morning after 2 students werâ€¦ https://t.co/KbPZqphDt3	Remadna	9.56E+17	4.74E+09	en
RT @AlexMunday_2018: Eleven.11 school shootings so far this year. We aren't even through January.While the families & students of todayâ€¦	SlythSeek	9.56E+17	8.46E+17	en
RT @shannonrwatts: My God. Two Americans were killed and 15 were injured in a shooting today at a Kentucky high school. What is wrong withâ€¦	stab3216	9.56E+17	2.2E+09	en
RT @GavinNewsom: There has been a school shooting in Kentucky that has left two teenagers dead and 17 injured and our country has become nâ€¦	SBStarbuc	9.56E+17	1.38E+09	en
RT @shannonrwatts: Zero school shootings in Canada so far this year. Ten in America. Meanwhile our own President hasnâ€™t mentioned the shâ€¦	CorbinCol	9.56E+17	1.05E+09	en
RT @Bakari_Sellers: The #1 story is NOT a School shooting in Kentucky. Thatâ€™s where we are now....	jmhpayne	9.56E+17	3.44E+09	en
RT @MamaMilkSD: Again and again. It doesnâ€™t have to be this way. We are the only developed country in the world who have this problem. We aâ€	Siadasha	9.56E+17	8.24E+17	en
RT @shannonrwatts: My God. Two Americans were killed and 15 were injured in a shooting today at a Kentucky high school. What is wrong withâ€¦	HotHamW	9.56E+17	3.12E+08	en
RT @wsbtv: NEW DETAILS: Student hurt in Kentucky school shooting says the teenage shooter â€œdidnâ€™t think twiceâ€ Coverage at 5:34 on Channe	elle__moi	9.56E+17	7.25E+17	en
RT @iamTannenbaum: How bad is the gun violence problem in America?Well a school shooting just happened in Kentucky. The most recent schoâ€¦	BRIXING_	9.56E+17	1.94E+09	en

kentucky_school_shooting

Figure 2. CSV File of Raw Tweet Collections Data

As we can see, there are URLs highlighted in red at the end of certain tweets, but not every tweet has a URL attached to it. We had to write a Python script to parse through the CSV file and extract only the URLs from each tweet. For each tweet, a regular expression pattern was applied to match the format of a URL and then the URLs were grouped and saved. Once the URLs have been saved, each URL had to be validated and extended using the "requests" library in Python [8]. Using the "requests" library, each URL was converted from its shortened Twitter URL to its full URL, as shown in Figure 3. In order to validate each link we checked the status code return from the request and checked for "200 OK", to confirm a valid link.

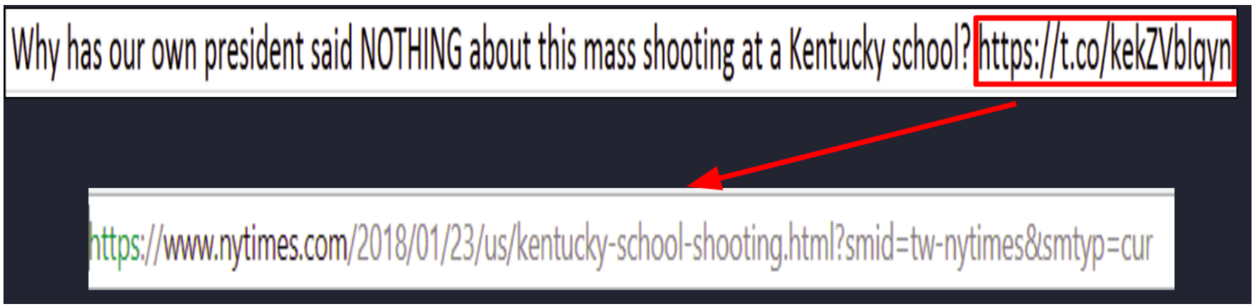

Figure 3. Conversion of short URLs to full URLs

The process for parsing tweet collections within JSON files was much more straightforward. The main problem with the JSON files exported from the tweet collections database is that each line in the generated JSON file is a separate JSON object. In order to account for this, the first step is to append a comma to the end of each line, and then encapsulate the contents of the file in square brackets. Once this is done, the file can be parsed by the Python JSON library [9]. Then, the script simply looks through the "URLs" entry of each recorded tweet. Any expanded URLs found are recorded, along with information about the tweet they correspond to, so they can be referenced back to, at a later time, if needed. An example of the URLs in JSON format can be seen in Figure 4.

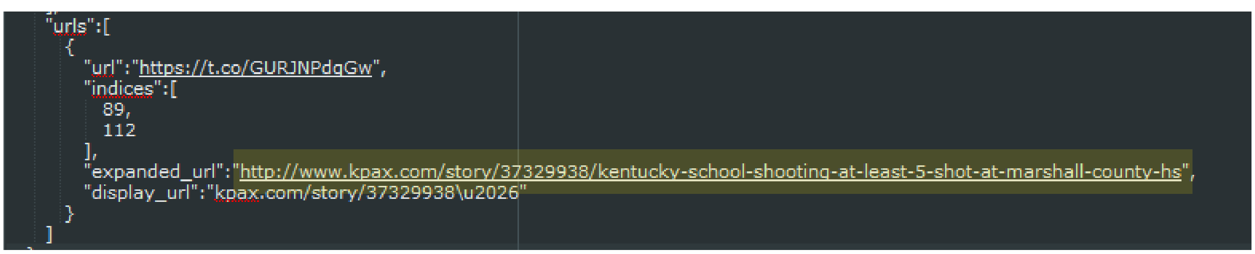

Figure 4. An example of the sort of entry our script is looking for

The next part of the pipeline is to extract webpage text for each of the detected URLs present in the tweet collection. Initially, each webpage is visited, and the full HTML for that URL is stored. This allows us to process the HTML of each page, extracting only the pieces of text related to the article. Therefore, we removed all HTML tags and JavaScript code from the raw HTML. While multiple approaches for doing so were tested, the most accurate and reliable of these seemed to be using a Python library called "newspaper," which attempts to extract text from only relevant sections of the webpage [10].

After the extraction of the text from the web pages, we had to clean up and refine the text to filter out unwanted data. For example, in Figure 5, you can see that while there is a lot of text present on the webpage, we only wanted to focus on relevant article text (surrounded by red). To filter out other words and nonrelevant information, we applied natural language processing techniques to each article. First, we had to use a Python library called the Natural Language Toolkit (NLTK) [11]. The NLTK library has a module called "sent_tokenize" that takes in a string as input and outputs a list of sentences within that string. We used this module to remove any phrases or words that occur on the web page that may be irrelevant to the tweet collection theme.

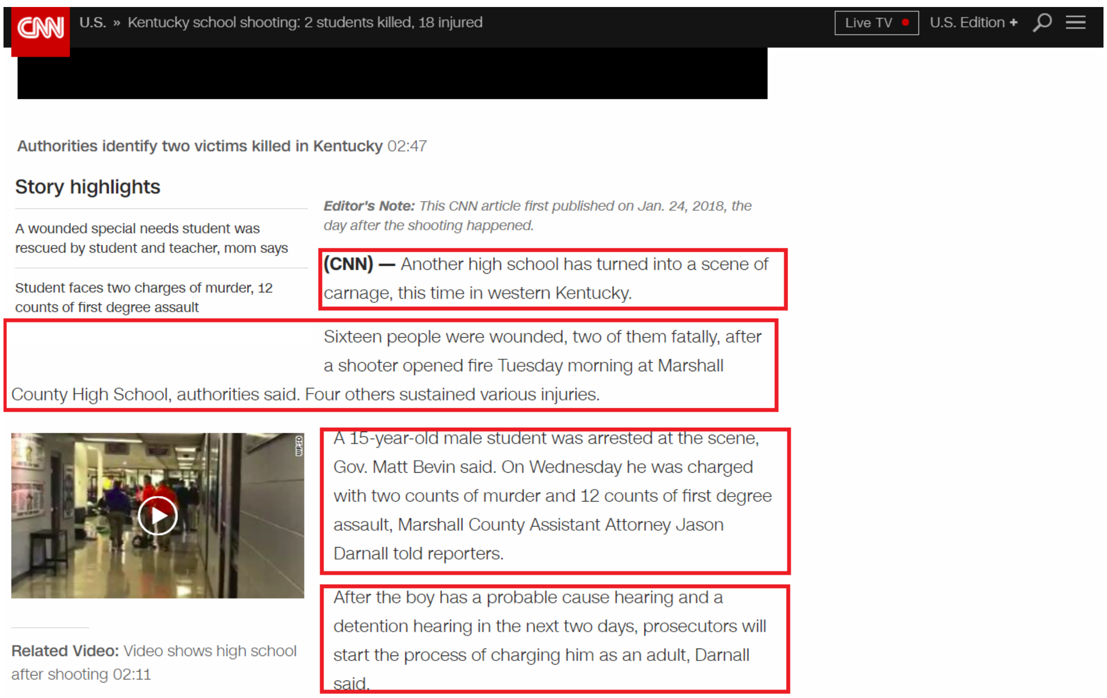

Figure 5. Example of Text Extraction from Article

Next, we wanted to refine our text even more by lemmatizing the words. The NLTK library has a module called "WordNetLemmatizer" that takes in a word and outputs the base form of the word. For example, lemmatizing the world "cats" would yield a result of "cat". More examples of lemmatization are shown in Figure 6.

```
>>> l.lemmatize("rocks")
'rock'
>>> l.lemmatize("cacti")
'cactus'
>>> l.lemmatize("geese")
'goose'
>>> l.lemmatize("fishes")
'fish'
```

Figure 6. Lemmatization Examples

7.2 model.py

In order to implement and train a classifier, we first had to convert the extracted text related to a URL into a vector. This vector is essentially an array of values ranging from 0 to 1 that represents the relevance of each word in the text. From the text extracted in the previous step, we use the scikit learn class called "TfidfVectorizer" to transform each string of text into a vector [12]. The vector has an entry for each term present, with a TF-IDF weight that reflects how important a certain word is to a particular document, as shown in Figure 7. Once we have the vector of the article, we can use this vector to train our specific model.

Benton, Kentucky	0.262517749327
Marshall County High School	0.195714190727
hospital	0.155514689834
handgun	0.152278806372
injured	0.130417524234
police	0.124411751867
shooting	0.122924977859
shooter	0.103793246466
Gabe Parker	0.098137987500
killed	0.095342360412
Atheism	0.091733665357
opened fire	0.086462298198
18 injured	0.078295921554
2 students killed	0.070282526017
Tuesday morning	0.069392769155
15-year old	0.058568771681
January 23, 2018	0.058568771681
Students	0.058210117288
Gun	0.054365052979
deadly school shooting	0.054365052979

Figure 7. Example TF-IDF scores showing relevance of each set of words in the corpus

Next, we had to decide on which classifier would be the most accurate for our situation. Therefore, we tested four different classifiers: Gaussian Naive Bayes, Support Vector Classifier, Random Decision Forest, and Decision Tree. After testing each classifier on our labeled data set, we found that the Random Decision Forest yielded the highest accuracy when it came to predicting if a URL was relevant to the topic. More extensive testing and results are shown in the next section.

7.3 conversion.py

We have created a final Python script that consolidates all the steps described above. The client has the option to run this script and input one URL or input a file with a list of URLs. If the client wishes to input one URL, then the output will be whether or not that URL is relevant based on the classifier's prediction. If the client inputs a list of URLs from a file, then the output will be every URL that is classified as relevant, so these links can be used in the Event Focused Crawler.

In terms of performance, we noticed that the script was taking a long time to run due to the training and vectorization of each webpage. Therefore, we used the pickle module in Python to "save" these objects to disk. The pickle module can serialize objects into a binary file or de-serialize it with the methods "dump" and "load" [13]. We used the pickle module to save the fitted Tfidfvectorizer into a file called "vectorizer.pickle" and we also saved the trained classifier into a file called "model.pickle". Therefore, when the script is executed, these objects will be loaded from the files and will not have to be re-fitted or re-trained.

8. Testing/Evaluation/Assessment

For our testing purposes, we have split our collected data into two parts, one for training and one for testing. We decided that we would have 80% of our data used for training our classifier, and 20% of our data would be used for testing and evaluation. Therefore, from our 3,000 labeled URLs, about 2,400 URLs were used for training and 600 URLs were used for testing. A summary of the test accuracy results for each classifier is displayed in table 2 below.

Classifier	Decision Tree	Random Forest	Support Vector (SVC)	Gaussian Naive Bayes
Test Accuracy	0.970967	**0.974193**	0.969354	0.790322
Cross Validation Accuracy	0.94 (+/- 0.06)	**0.95 (+/- 0.06)**	0.95 (+/- 0.06)	0.75 (+/- 0.29)

Table 2. Prediction Accuracy of Classifiers

As we can see, for a single test with no cross-validation the random forest classifier yielded the highest percentage for accuracy. After deciding that the random forest classifier would be the most optimal choice, we decided to extensively test which parameters would increase our accuracy even more. Therefore, we thoroughly tested the n_estimators, min_sample_leaves, max_depth, max_features, and random_state. To test these parameters, we created multiple nested for loops that iterate through each parameter and prints out the accuracy for a single test. After the parameters have been tested, we chose the score with the highest accuracy of 0.98, shown in Figure 8, and set the parameters of our classifier accordingly.

```
0.972580645161 ---- depth: 19, n_estimators: 7, max_features: auto, min_samples_leaf: 1, random_state: 14
0.98064516129 ---- depth: 19, n_estimators: 7, max_features: auto, min_samples_leaf: 1, random_state: 15
0.972580645161 ---- depth: 19, n_estimators: 7, max_features: auto, min_samples_leaf: 1, random_state: 16
0.970967741935 ---- depth: 19, n_estimators: 7, max_features: auto, min_samples_leaf: 1, random_state: 17
0.974193548387 ---- depth: 19, n_estimators: 7, max_features: auto, min_samples_leaf: 1, random_state: 18
```

Figure 8. Example of Parameter Testing Output

We also performed some additional testing to check how easy it was to apply these trained classifiers to URLs related to various other school shootings. As such, we obtained links relating to other school shootings, and found whether each classifier determined the article was relevant or not. The results for a few sample links are listed in table 3 below.

URL	Decision Tree		Random Forest		Gaussian Naive Bayes	
	Probability link is irrelevant	**Probability link is relevant**	**Probability link is irrelevant**	**Probability link is relevant**	**Probability link is irrelevant**	**Probability link is relevant**
https://www.cbsnews.com/news/florida-shooting-marjory-stoneman-douglas-high-school-today-2018-02-14/	97.14%	2.85%	22.25%	77.74%	100%	0%
https://en.wikipedia.org/wiki/Umpqua_Community_College_shooting	97.14%	2.85%	62.37%	37.62%	100%	0%
https://www.nytimes.com/2015/10/02/us/oregon-shooting-umpqua-community-college.html	97.14%	2.85%	71.42%	28.57%	0%	100%

Table 3. Models' Predictions on Links Related to Other School Shootings

As evident from table 3, it appears the decision tree classifier consistently classified all links tested as irrelevant with a high degree of confidence. Upon further analysis, it became clear why the Decision Tree classifier was so much worse at generalizing to be able to classify other school shootings. Because of the way decision tree classifiers work, each time we trained our decision tree classifier, it was choosing words most commonly present in the relevant links but not in the irrelevant links. As these words were often specific to that school shooting, as can be seen in Figure 9 below, all of our trained decision tree classifiers were terrible at generalizing to other school shootings.

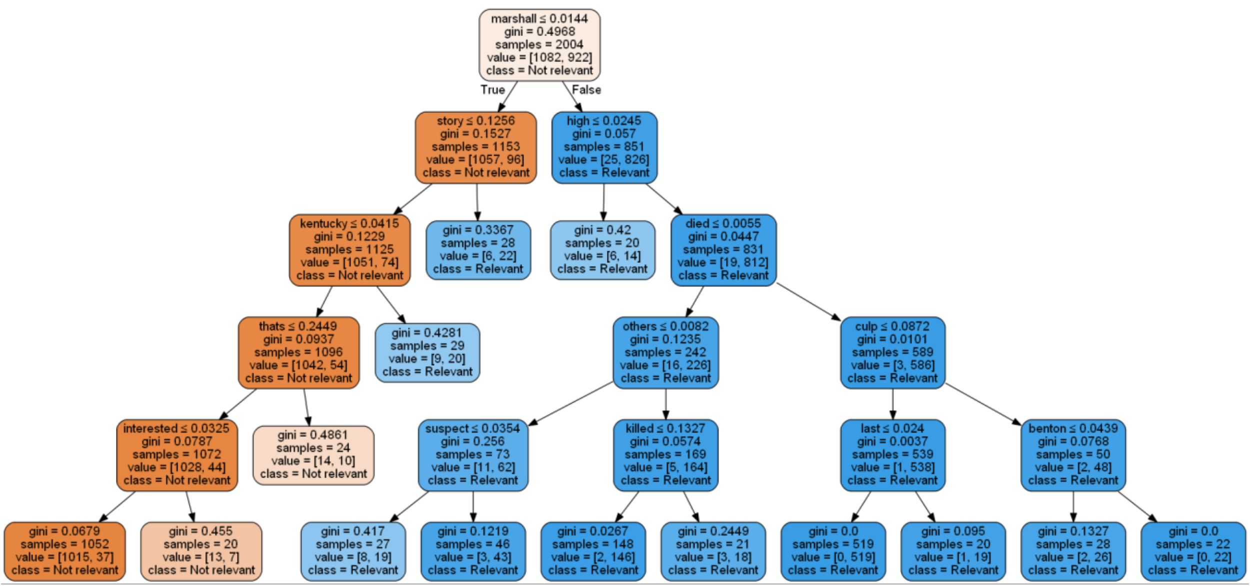

Figure 9. Example of Decision Tree Classifier Generated Through Our Approach*

Note that keywords related to this specific school shooting, including “marshall,” “kentucky,” and “benton” are all used in the decision tree to help determine whether the link is relevant or not.

9. User’s Manual

9.1 Environment Setup & Installation

All of our project code is written in Python 3, which should make it easy to run on most common operating systems. In order to run the code, you first need to install Python 3, which can be downloaded from https://www.python.org/downloads/. Figure 10 shows where the user can download Python 3. Note that the code has not been tested with Python 2, so we recommend installing Python 3 even if you already have Python 2 on your system.

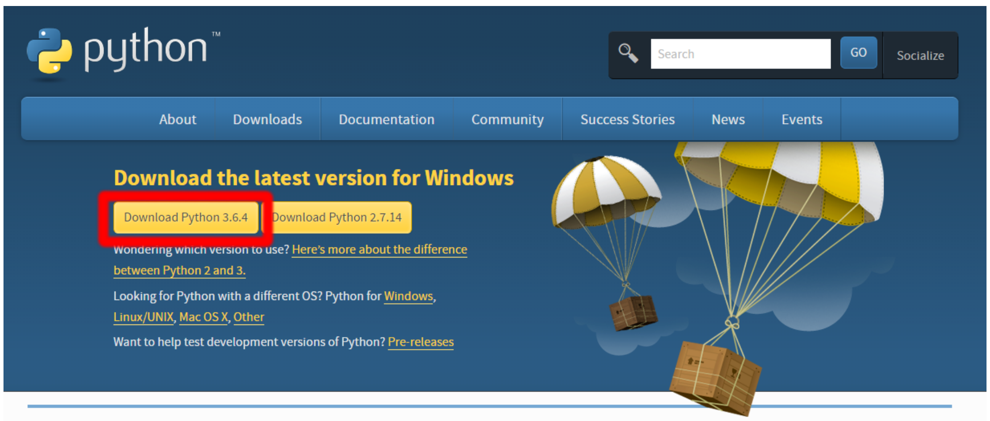

Figure 10. The Python download page. Button users should click is marked in red.

Once Python is installed on your system, you should next make sure pip - a Python package management system - is also installed. To do so, type pip -V into your preferred console. If you get output detailing the version of pip on your system, it should be installed. Figure 11 shows an example command a user can run to test if pip was installed successfully.

```
> pip -V
pip 9.0.1 from C:\ProgramData\Anaconda3\lib\site-packages (python 3.6)
```

Figure 11. An example of the sort of output expected when pip is installed

If you instead get an error stating that pip was not found, you should install it. If you are having trouble doing so, the following page has an informative guide on installing it: https://pip.pypa.io/en/stable/installing/.

With pip installed, you can now install all of the Python libraries our code relies on. First, navigate to the directory containing all of our project’s code in your preferred console. Assuming you're in the right folder, there should be a file present called requirements.txt, which contains a list of our code's requirements. To install all of them, enter the following command:

```
pip install -r requirements.txt
```

Wait for this command to finish. Upon completion, you should now have all of the requirements necessary to run the code yourself!

9.2 Obtaining Tweet Collections

Our code supports tweet collections exported from [yourTwapperKeeper](https://github.com/540co/yourTwapperKeeper), an open source tweet collection platform, in JSON format. Please consult the official project repository, which can be found at https://github.com/540co/yourTwapperKeeper, for more information about this tool.

For more information about Virginia Tech's Tweet collection efforts, visit http://www.eventsarchive.org/. An example *yourTwapperKeeper* deployment can be found at http://ytk1.dlib.vt.edu/twitter/index.php.

9.3 Extracting & Resolving URLs from Tweet Collections

Once a tweet collection is obtained, the next step in the process is to extract all URLs from it. A script we created as part of our project called *extract.py* handles this process. Input to the script should be a JSON file as obtained from yourTwapperKeeper (see above). This is passed with the *-t* or *--tweetcollection* parameter. The script will extract URLs from this file. Once these URLs are resolved, they will be output to a separate file. This output file is specified with the *-o* or *--output* parameter. Additionally, the script also generates metadata about each resolved URL along the way as well. If a user would like to view this information in addition to the simple list of URLs, they can specify the *-eo* or *--extendedoutput* parameter to specify where this extended information should be saved.

For example, if a user had an exported collection of tweets called *collection.json* and wanted to save a list of resolved URLs to a file called *resolved_links.txt*, and also wanted to save all of the additional metadata to another file called *tweet_url_info.csv*, they would enter the following command:

```
python extract.py -t collection.json -o resolved_links -eo tweet_url_info.csv
```

After running, output similar to Figure 12 should be observed.

```
> python extract.py -t collection.json -o resolved_links -eo tweet_url_info.csv
Loading Tweets
Extracting URLs from the Tweet collection
Resolving shortened URLs
Saving results
```

Figure 12. Example Output from *extract.py*

Note that the extended output parameter is optional, not required. For complete information regarding the supported and required arguments, please consult the appendix.

9.4 Analyzing Tweet Collections

One of the final deliverables is the Python script that is used to determine whether or not a URL is relevant to the Kentucky School Shooting that occurred on January 23rd, 2018. The user can run the following command to test a single URL:

```
python conversion.py -u [URL]
```

```
C:\Users\toms_000\Desktop\TweetURLExtraction\code>python conversion.py -u https:
//www.cnn.com/2018/01/23/us/kentucky-high-school-shooting/index.html
[[ 0.28571429  0.71428571]]
[1]
```

Figure 13. Example Output for a Single URL

Shown above in Figure 13, we can see that the script will output two lines. In this case, the first line will output two values in an array. These two values in the array indicate the percentage of how likely it is a relevant or irrelevant URL. An example of a relevant URL will have the first value in the array closer to 0 and the second value of the array will have a value closer to 1. The second line the program outputs is whether or not the URL is relevant. A value of 1 on the second line of output represents a relevant URL, and a value of 0 represents an irrelevant URL.

Additionally, a user can test multiple URLs from a file by running the following command:

```
python conversion.py -f [file_name]
```

The output of this command will be a list of all relevant URLs found inside the file. Figure 14 shows an example of the output when this command is run. The user could pipe these URLs into a separate file or may wish to use it as input for the Event Focused Crawler.

```
C:\Users\toms_000\Desktop\TweetURLExtraction\code>python conversion.py -f ../dat
a/good.txt
http://13wham.com/news/nation-world/reports-possible-school-shooting-in-marshall
-county-kentucky
http://5newsonline.com/2018/01/23/kentucky-school-shooting-one-dead-at-least-5-s
hot/
http://6abc.com/1-dead-many-wounded-in-kentucky-school-shooting/2980258/
```

Figure 14. Example Output for a List of URLs

By default, the classification process makes use of two provided files, *model.pickle* and *vectorizer.pickle*. However, the model and vectorizer to use within the classification process can also be specified manually by the *-m* (or *--model*) and *-v* (or *--vectorizer*) parameters, respectively. This allows users to use models and vectorizers obtained from *create_model.py* instead of the provided ones.

For more information regarding the supported and required arguments to this script, please see the appendix.

9.5 Training Custom Models

While we have provided a trained model for a sample tweet collection relating to the recent Kentucky School Shooting, we understand that users may want to generate their own classification model(s) for other tweet collections. As such, we have provided code for generating custom models, which can be trained on other datasets.

The first step in the training process is to generate a list of relevant and non-relevant links, so that the classifier can be trained to determine whether a link is relevant or not. This process is fairly straightforward. Users should create two separate text files, one of which will contain relevant links, and another which will contain non-relevant links. An example is shown in Figures 15 and 16 below. Note the file names do not matter, as long as you are able to identify which file contains the relevant links and which file contains the non-relevant links.

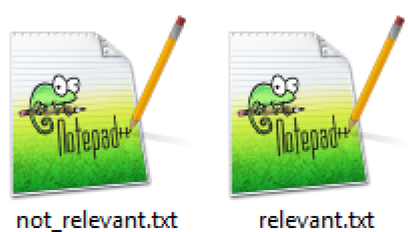

Figure 15. An example of the files needed for training the model

Within these files, users should place a list of URLs they deem relevant and not relevant, respectively.

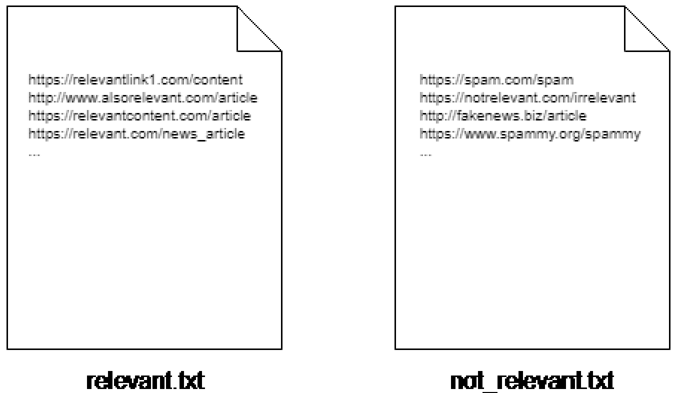

Figure 16. An example of the contents in each file

Then, all that's left is to invoke the script to generate a model from these links. The script is contained within *create_model.py*, and requires several parameters. A full printout of the supported and required arguments can be found in the appendix.

Following along with the example, users can train the model on the two created text files (relevant.txt and not_relevant.txt) by passing them as arguments to the program. Additionally, the user must supply a list of stop words, and file names for the generated model and vectorizer to be stored in. Assuming relevant links are stored in a file called *relevant.txt*, non-relevant links are stored in a file called *non_relevant.txt*, the list of stop words is stored in a file called *stop_words.txt*, and you want the model to be stored in *model.pickle* and the vectorizer to be stored in *vectorizer.pickle*, the following command should be entered:

```
python create_model.py -r relevant.txt -n not_relevant.txt -sw stop_words.txt
-mo model.pickle -vo vectorizer.pickle
```

After running, the output should be similar to Figure 17, although it will of course vary based off of the input provided.

```
> python create_model.py -r relevant.txt -n not_relevant.txt -sw stop_words.txt -mo model.pickle -vo vectorizer.pickle
Scraping articles
Extracting text from each article
Normalizing text
Creating corpus
Number of relevant links: #
Number of not relevant links: #
Creating TFIDF vectors
Training RandomForestClassifier
Classifier accuracy: #
Dumping model and vectorizer
```

Figure 17. An example of the output expected after running the model generation script

If a user wants additional debug output to be printed, they can add the -v flag to the script arguments. When added, this flag will make the script print out additional information about its progress throughout the script.

10. Developer's Manual

A github repository will be provided in the appendix, to aid future developers, for open source collaboration. A developer could either clone or fork depending on whether they want to contribute to the project or use it for their own purposes. For requirements, the developer should have the latest version of Python 3.6 installed including the latest version of pip. The developer could then run "pip install -r requirements.txt" to install all packages and imported libraries necessary for the project.

The final Python script to be delivered is named conversion.py. The developer could clone the repository and make modifications to the script as needed. Each component in this script has been separated into different functions.

A list of all of the major provided scripts and other files, along with a detailed description of what each of them does, can be found in table 4 below.

File	Description
extract.py	Extracts links from a tweet collection and resolves them to full URLs.
create_model.py	Allows users to generate their own model with training data they provide. Generates a model and vectorizer which can be used in *conversion.py*.
conversion.py	Predicts whether the given links are relevant or not. Uses a trained model and vectorizer to make these predictions.
model.py	Trains a classifier model that can be used to predict whether links are relevant or not.
kentucky school shooting.json	A JSON file containing an example of the sort of JSON needed as input for *extract.py*.
model.pickle	The model used to classify articles as good or bad. Trained by *model.py* using the training dataset, *good.txt* and *bad.txt*.
vectorizer.pickle	The vectorizer used to convert articles to vectors. Generated by *model.py* using the training dataset, *good.txt* and *bad.txt*.
good.txt	A list of URLs that were labeled as relevant. Used in the training process of *model.py*.
bad.txt	A list of URLs that were labeled as not relevant. Used in the training process of *model.py*.
stop_words.txt	A list of words that are ignored when generating the TF-IDF scores in *model.py*.
randomurls.py	Generates a list of random URLs. Useful for creating additional non-relevant links for training purposes.

Table 4. Description of Major Project Files

10.1 Future Work

Due to time constraints, much of this project was focused on generating classifiers for a very specific collection of URLs taken from tweets about the recent Kentucky school shooting. In the presence of more time, we would have liked to work on generalizing our classification approach, making it usable for not just Kentucky school shootings, but possibly school shootings in general, or simply valid news articles vs. spam articles and/or generally not newsworthy articles. As such, one suggested direction for others interested in continuing the work covered by our project is to try and generalize the classification technique to encompass a wider variety of articles.

11. Lessons Learned

During the work of this project, we learned many new programming concepts, tools, and ways to find new solutions. In the starting stages, our group came across small issues that imposed obstacles on our work. With many of these issues, we asked our client many questions and he was helpful in offering his advice, and so we found a solution very quickly.

For example, we found that there weren't enough irrelevant URLs in the data set given to us. After extracting and analyzing the URLs from the original tweet collections files, we found that there were about 3,000 URLs. From that set of URLs, there were about 1,500 (50%) relevant URLs and 1,000 (33%) irrelevant URLs, and the remaining 500 (16%) URLs were thrown out due to different status code errors. Therefore, in order to generate a large number of random links, we created a Python script to do so (*randomurls.py*). The script utilizes module named "RandomWords" [14] to generate a list of random words. For each of the words in the list of random words, we created a Google search query with that specific word that searches Google News and retrieves the first URL of the search result. In total, we generated the remaining 500 URLs with the script. This script can be found in the github repository named "randomurls.py". We used these randomly generated URLs as our irrelevant URLs.

Another lesson that we learned was that certain URLs resolved to different status codes. For example, some URLs had a status code of 404 which equates to "Not Found". We weren't sure whether or not to consider these URLs as irrelevant or just throw them out of our data set. Therefore, when we contacted the client, he recommended us to just remove them from the data set.

During spring break our group didn't have a meeting and we learned that before a long hiatus it's a good idea to have a meeting to keep the productivity going even when working remotely. Momentum definitely slowed down afterwards, as can be seen in Figure 18 of commits to the project repository:

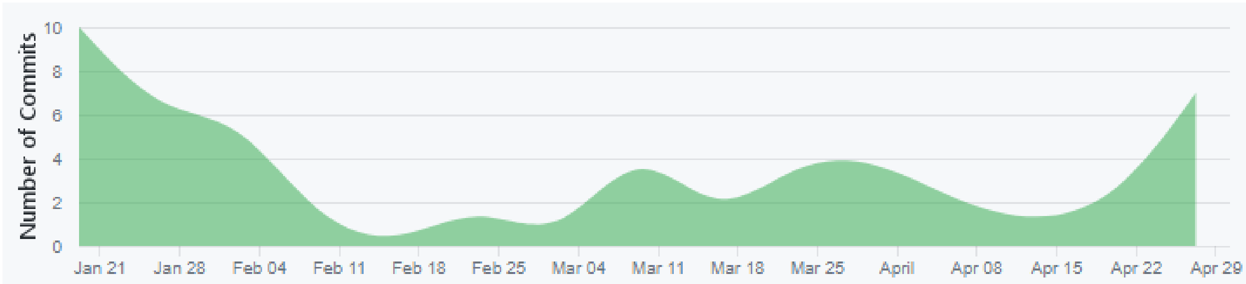

Figure 18. Number of Commits by Week

Data standardization was another issue our team had to deal with. Throughout the course of the project, we all worked on different parts of the pipeline, often generating data to be passed to the next stage. However, in several cases, we had different expectations of what format the data would be in when sent/received, which meant we had to go back and change code to support a different data format than anticipated.

Acknowledgements

We want to give our thanks to Dr. Fox for providing guidance and offering his support. We would also like to thank our client Liuqing Li for meeting with our team and offering his support. We are extremely grateful for all of the very helpful advice they both provided throughout the semester. We would also like to acknowledge the Global Event and Trend Archive Research team (GETAR) which is supported by NSF (IIS-1619028 and 1619371).

Dr. Fox can be reached via email at fox@vt.edu.
Liuqing Li can be reached via email at liuqing@vt.edu.

References

[1]: "Events Archive Invitation, Funding | Events Archiving." http://www.eventsarchive.org/. Accessed 1 May 2018.

[2]: “Total Number of Websites.” *Total Number of Websites - Internet Live Stats*, InternetLiveStats, www.internetlivestats.com/total-number-of-websites/. Accessed 28 Apr. 2018

[3]: salman.aslam.mughal. “Twitter by the Numbers: Stats, Demographics & Fun Facts.” • *Twitter by the Numbers (2018): Stats, Demographics & Fun Facts*, 26 Jan. 2018, www.omnicoreagency.com/twitter-statistics/.

[4]: Nicholson, Chris V., and Adam Gibson. “Bag of Words & TF-IDF.” Bag of Words - TF-IDF - Deeplearning4j: Open-Source, Distributed Deep Learning for the JVM, SkyMind, www.deeplearning4j.org/bagofwords-tf-idf. Accessed 3 Mar. 2018

[5]: “Sklearn.feature_extraction.Text.TfidfVectorizer.” 1.4. Support Vector Machines - Scikit-Learn 0.19.1 Documentation, scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html. Accessed 5 May 2018.

[6]: Vanschoren, Joaquin. “OpenML.” *OpenML: Exploring Machine Learning Better, Together.*, www.openml.org/a/estimation-procedures/1. Accessed 3 Mar. 2018

[7]: “Natural Language Toolkit.” NLTK 3.3 Documentation, NLTK Project, www.nltk.org/.

[8]: Reitz, Kenneth. “Quickstart.” *Quickstart - Requests 2.18.4 Documentation*, MMXVIII, www.docs.python-requests.org/en/master/user/quickstart/. Accessed 1 May 2018

[9]: “19.2. Json - JSON Encoder and Decoder.” *19.2. Json - JSON Encoder and Decoder - Python 3.6.5 Documentation*, Python Software Foundation, 2 May 2018, www.docs.python.org/3/library/json.html. Accessed 1 May 2018

[10]: Ou-Yang, Lucas. “Newspaper3k: Article Scraping & Curation.” *Newspaper3k: Article Scraping & Curation - Newspaper 0.0.2 Documentation*, 2013, www.newspaper.readthedocs.io/en/latest/. Accessed 1 May 2018

[11]: Bird, Steven, Edward Loper and Ewan Klein (2009), *Natural Language Processing with Python*. O’Reilly Media Inc. https://www.nltk.org/. Accessed 30 Apr. 2018

[12]: Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011. Accessed 30 Apr. 2018

[13]: “12.1. Pickle - Python Object Serialization.” *12.1. Pickle - Python Object Serialization - Python 3.6.5 Documentation*, Python Software Foundation, 2 May 2018, www.docs.python.org/3/library/pickle.html. Accessed 1 May 2018

[14]: Święcicki, Tomek. “RandomWords.” *PyPI*, Python Software Foundation, www.pypi.org/project/RandomWords/. Accessed 1 May 2018

Appendices

A1. Repository Information

A github repository of all code related to this project can be found at https://github.com/carter144/TweetURLExtraction.

If you are interested in contributing to this project, please reach out via Github to one of the project members:

- Carter Tat (carter144)
- Chris Bridges (chris-bridges)
- David Chun (cdavid0)

A1. Arguments for Various Programs

extract.py

```
usage: extract.py [-h] [-eo EXTENDEDOUTPUT] [-t TWEETCOLLECTION] [-o OUTPUT]

Extracts and resolves URLs within a tweet collection

optional arguments:
  -h, --help            show this help message and exit
  -eo EXTENDEDOUTPUT, --extendedoutput EXTENDEDOUTPUT
                        The file to store the results and additional metadata
                        in.

required arguments:
  -t TWEETCOLLECTION, --tweetcollection TWEETCOLLECTION
                        A JSON file containing the list of tweets to process
                        (exported from yourTwapperKeeper)
  -o OUTPUT, --output OUTPUT
                        The file to store a list of resolved URLs in.
```

conversion.py

```
usage: conversion.py [-h] [-u URL] [-f FILE] [-m MODEL] [-v VECTORIZER]

Process some integers.

optional arguments:
  -h, --help            show this help message and exit
  -u URL, --url URL     A single URL to classify as relevant or not
  -f FILE, --file FILE  An input file containing a list of URLs to check the
                        relevance of
  -m MODEL, --model MODEL
                        A pickle object containing a trained model used for
                        classification
  -v VECTORIZER, --vectorizer VECTORIZER
                        A pickle object containing a vectorizer used to
                        vectorize the text in each link
```

create_model.py

```
usage: create_model.py [-h] [-v] [-r RELEVANT] [-n NONRELEVANT]
                       [-sw STOPWORDS] [-mo MODELOUT] [-vo VECTOROUT]

Generates a classifier model from training data

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         If provided, prints extra debug information

required arguments:
  -r RELEVANT, --relevant RELEVANT
                        A file containing the relevant URLs
  -n NONRELEVANT, --nonrelevant NONRELEVANT
                        A file containing non-relevant URLs
  -sw STOPWORDS, --stopwords STOPWORDS
                        A file containing stop words
  -mo MODELOUT, --modelout MODELOUT
                        The file to store the generated model in
  -vo VECTOROUT, --vectorout VECTOROUT
                        The file to store the generated vectorizer in
```

A3. Project Timeline

Table 5 indicates our milestones and plans for the project, on a weekly basis.

Week	**Actions Performed**
Jan. 22 - Jan. 26	Assign Roles Initialize Github repo
Jan. 29 - Feb. 2	Obtain sample dataset Plan architecture/framework Extract & resolve all URLs from tweets Start labeling a dataset of URLs for training purposes
Feb. 5 - Feb. 9	Install dependencies Scrape and parse web pages Create module to retrieve and store data from URLs
Feb. 12 - Feb. 16	Research NLP and Scikit learn for machine learning portions
Feb. 19 - Feb. 24	Research additional machine learning technologies, including Tensorflow/Pytorch and Stanford entity name recognizer
Feb. 26 - Mar. 2	Create sample machine learning module
Mar. 5 - Mar. 9	SPRING BREAK
Mar. 12 - Mar. 16	Implement natural language processing and data labeling code
Mar. 19 - Mar. 23	Refine created code
Mar. 26 - Mar. 30	Test code with various datasets

Apr. 2 - Apr. 6	Create manual and other supporting documentation
Apr. 9 - Apr. 13	Finalize prototype code
Apr. 16 - Apr. 20	Code cleaning Inline code documentation
Apr. 23 - Apr. 27	Test application modules to ensure they’re functioning correctly
Apr. 30 - May 2	Finalize project report

Table 5. Finalized Project Timeline