BIG DATA TEXT SUMMARIZATION

Hurricane Harvey

Address: Virginia Tech, Blacksburg, VA 24061
Instructor: Edward A. Fox

Course: CS4984/5984
Members: Jack Geissinger, Theo Long, James Jung, Jordan Parent, and Robert Rizzo

Date: 12/10/2018

Table of Contents

List of Figures

List of Tables

1. Executive Summary

Natural language processing (NLP) has advanced in recent years. Accordingly, we present progressively more complex generated text summaries on the topic Hurricane Harvey.

We utilized TextRank, which is an unsupervised extractive summarization algorithm. TextRank is computationally expensive, and the sentences generated by the algorithm aren't always directly related or essential to the topic at hand. When evaluating TextRank, we found that a single sentence interjected and ruined the flow of the summary. We also found that ROUGE evaluation for our TextRank summary was quite low compared to a golden standard that was prepared for us. However, the TextRank summary had high marks for ROUGE evaluation compared to the Wikipedia article lead for Hurricane Harvey.

To improve upon the TextRank algorithm, we utilized template summarization with named entities. Template summarization takes less time to run than TextRank but is supervised by the author of the template and script to choose valuable named entities. Thus, it is highly dependent on human intervention to produce reasonable and readable summaries that aren't error-prone. As expected, the template summary evaluated well compared to the Gold Standard and the Wikipedia article lead. This result is mainly due to our ability to include named entities we thought were pertinent to the summary.

Beyond extractive summaries like TextRank and template summarization, we pursued abstractive summarization using pointer-generator networks and multi-document summarization with pointer-generator networks and maximal marginal relevance. The benefit of using abstractive summarization is that it is more in-line with how humans summarize documents. Pointer-generator networks, however, require GPUs to run properly and a large amount of training data. Luckily, we were able to use a pre-trained network to generate summaries. The pointer-generator network is the centerpiece of our abstractive methods and allowed us to create summaries in the first place.

NLP is at an inflection point due to deep learning, and our generated summaries using a state-of-the-art pointer-generator neural network are filled with details about Hurricane Harvey, including damage incurred, the average amount of rainfall, and the locations it affected the most. The summary is also free of grammatical errors. We also use a novel Python library, written by Logan Lebanoff at the University of Central Florida, for multi-document summarization using deep learning to summarize our Hurricane Harvey dataset of 500 articles and the Wikipedia article for Hurricane Harvey. The summary of the Wikipedia article is our final summary and has the highest ROUGE scores that we could attain.

2. Introduction

This report is a culmination of a semester-long team project targeted at generating simple summaries of Hurricane Harvey as part of the Big Data Text Summarization course taught by Dr. Edward A. Fox. We produced many summaries over the semester with different techniques including TextRank, templating, and deep learning.

We operated mainly on the small dataset from ArchiveSpark, which contained around 500 articles, and the Wikipedia article for Hurricane Harvey. The reasoning behind this was that we wanted to prototype as quickly as possible and reduce computation time, and the small dataset and the Wikipedia article allowed for this. The small dataset was used with the TextRank algorithm and the template summary, while the Wikipedia article was mainly used with the deep learning approaches.

We combined our summaries from the template approach and the deep learning approach to create a readable summary that covers the main facts of the event while remaining grammatically correct.

Finally, we generated summaries for the small dataset and the Wikipedia article using multi-document summarization with deep learning. The summaries are mostly grammatically correct with some mistakes. The multi-document summary for the Wikipedia article has the best ROUGE score we could generate and is filled with facts about Hurricane Harvey. It is our best summary by far.

To make as much progress as possible, we avoided some Units in the course material that didn't directly support the end goal of creating extractive and abstractive summaries. We also made use of many team collaboration tools such as GroupMe and Google Drive. We used these tools mainly to keep tabs on progress in different parts of the project and to keep all of our files in a single location.

3. Literature Review

Our literature review took place near the beginning of the semester shortly after being assigned to teams. We mainly focused on deep learning at first because this aspect of the project was most interesting to us. We found a number of articles and tutorials on deep learning that we found helpful such as [1], [2], [3], [4], [5]. Deep learning for NLP is also a problem that is not solved and is most challenging currently, which is why we studied this to a greater extent than other topics.

We also investigated tools for classical NLP, but to a lesser degree than deep learning. We found a number of libraries for this task that were also recommended by Dr. Fox. These include the NLTK library [6] and the spaCy library [7]. We personally made use of both and this will be explained more in section 6. For template summaries, we gained insight and perspective from chapter 17 in Speech and Language Processing, 3rd edition by Daniel Jurafsky and James Martin [8].

Near the end of the semester, we found another library published in 2018 for multi-document summarization using deep learning [9], [10]. The library enabled us to summarize longer documents, including our small dataset and the Wikipedia article for Hurricane Harvey.

4. Users' Manual

Our code, apart from the pointer-generator network, is fairly simple to use. It requires a machine with Python 3.7 and Python 2.7. We recommend creating an Anaconda environment to isolate the requirements for this project from others. Using the requirements.txt file, most of the dependencies can be installed easily in an Anaconda environment. In addition, many scripts require NLTK to be installed and the NLTK corpora through nltk.download(). The usage of "template_summarizer.py" requires the usage of spaCy that can be used after downloading the 'en_core_web_md' model. Some libraries such as textblob may require the use of pip to install. These libraries should become apparent when running a given file because it will fail due to that library not being available. All of the scripts were developed locally and don't require a cluster to run.

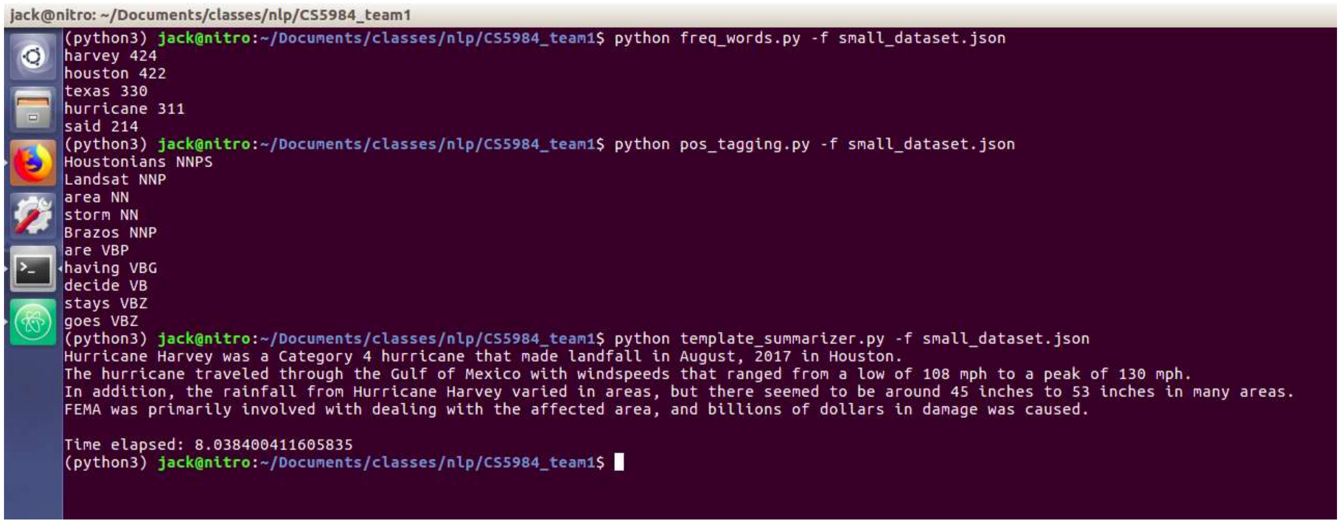
Files for users:

1. freq_words.py - Finds the most frequent words in a JSON file that contains a sentences field. Requires a file to be passed through the -f option.
2. pos_tagging.py - Performs basic part-of-speech tagging on a JSON file that contains a sentences field. Requires a file to be passed through the -f option.
3. textrank_summarizer.py - Performs TextRank summarization with a JSON file that contains a sentences field. Requires a file to be passed through the -f option.
4. template_summarizer.py - Performs template summarization with a JSON file that contains a sentences field. Requires a file to be passed through the -f option.
5. wikipedia_content.py - Extracts content from a Wikipedia page given a topic and formats the information for the pointer-generator network using the "make_datafiles.py" script. Requires a topic to be given in the -t option and an output directory for "make_datafiles.py" to read from with the -o option.

The pointer-generator network has its own README.md which can be studied to understand how to run with the output of wikipedia_content.py. There are a number of requirements, including a pre-trained network, a vocab file, and the pointer-generator network software [11]. However, keep in mind that deep learning is very slow on CPUs, and the computer used for running the pointer-generator network had a Nvidia 1050 GPU. We recommend referring to the pointer-generator network README.md file for reference on how to run it. It is extensive and too detailed to relay here.

We also made use of a multi-document summarization library [10]. This library utilizes the pointer-generator network, but is much easier to use. It summarizes multiple documents at a time. To discover more about how to use it, please refer to the README.md for the repository. Keep in mind that it also uses deep learning, so a GPU will be required to run it efficiently.

To use the files effectively, we share a few screenshots of us running the files in the terminal on Ubuntu 16.04 in a Anaconda environment. For the freq_words.py, pos_tagging.py, and template_summarizer.py files, we demonstrate how to run the scripts in Figure 1 below.

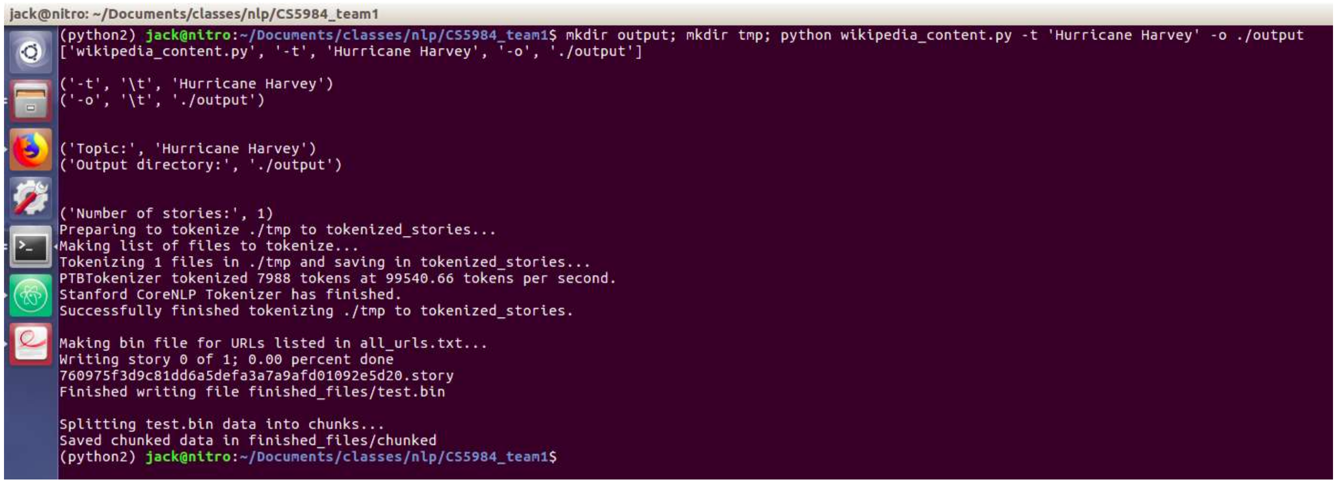


```
jack@nitro: ~/Documents/classes/nlp/CS5984_team1
(python3) jack@nitro:~/Documents/classes/nlp/CS5984_team1$ python freq_words.py -f small_dataset.json
harvey 424
houston 422
texas 330
hurricane 311
said 214
(python3) jack@nitro:~/Documents/classes/nlp/CS5984_team1$ python pos_tagging.py -f small_dataset.json
Houstonians NNPS
Landsat NNP
area NN
storm NN
Brazos NNP
are VBP
having VBG
decide VB
stays VBZ
goes VBZ
(python3) jack@nitro:~/Documents/classes/nlp/CS5984_team1$ python template_summarizer.py -f small_dataset.json
Hurricane Harvey was a Category 4 hurricane that made landfall in August, 2017 in Houston.
The hurricane traveled through the Gulf of Mexico with windspeeds that ranged from a low of 108 mph to a peak of 130 mph.
In addition, the rainfall from Hurricane Harvey varied in areas, but there seemed to be around 45 inches to 53 inches in many areas.
FEMA was primarily involved with dealing with the affected area, and billions of dollars in damage was caused.

Time elapsed: 8.038400411605835
(python3) jack@nitro:~/Documents/classes/nlp/CS5984_team1$ 
```

Figure 1: Examples of running Python 3.7 scripts using an Anaconda environment.

The pipeline for the pointer-generator network is all in Python 2.7. Thus, it is necessary to have a separate Anaconda environment for Python 2.7. In Figure 2 below, we demonstrate the running of the wikipedia_content.py script. Be aware that the Stanford CoreNLP library is required to run this script and the CLASSPATH must be set for the Stanford CoreNLP .jar file.



```
jack@nitro: ~/Documents/classes/nlp/CS5984_team1
(python2) jack@nitro:~/Documents/classes/nlp/CS5984_team1$ mkdir output; mkdir tmp; python wikipedia_content.py -t 'Hurricane Harvey' -o ./output
['wikipedia_content.py', '-t', 'Hurricane Harvey', '-o', './output']

('-t', '\t', 'Hurricane Harvey')
('-o', '\t', './output')


('Topic:', 'Hurricane Harvey')
('Output directory:', './output')


('Number of stories:', 1)
Preparing to tokenize ./tmp to tokenized_stories...
Making list of files to tokenize...
Tokenizing 1 files in ./tmp and saving in tokenized_stories...
PTBTokenizer tokenized 7988 tokens at 99540.66 tokens per second.
Stanford CoreNLP Tokenizer has finished.
Successfully finished tokenizing ./tmp to tokenized_stories.

Making bin file for URLs listed in all_urls.txt...
Writing story 0 of 1; 0.00 percent done
760975f3d9c81dd6a5defa3a7a9afd01092e5d20.story
Finished writing file finished_files/test.bin

Splitting test.bin data into chunks...
Saved chunked data in finished_files/chunked
(python2) jack@nitro:~/Documents/classes/nlp/CS5984_team1$
```

Figure 2: Example of running wikipedia_content.py in a Python 2.7 Anaconda environment.

5. Developers' Manual

A developer needs to have a machine with Python 3.7. We recommend creating an Anaconda environment to isolate the requirements for this project from others so that development is easier. In addition to the comments made in the User's Manual, we recommend also having a fast computer for development. The scripts were developed locally, but with larger datasets the processing may become cumbersome. **Files for developers are the same as for users.**

5.1 Dataflow for Developers

We give a brief description of the data flow for each file so that developers can understand how to extend the functionality further.

Most of our files follow the format shown in Figure 3. The JSON file of our small dataset is read into memory. Then the sentences from each article are taken from the JSON file and saved into a long string. Finally, we do any procedure we deem necessary to achieve a particular task.

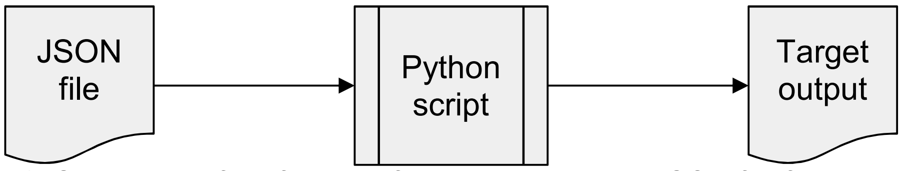


Figure 3: Simple data flow for most files that take in the JSON file for processing.

For some scripts such as finding the most frequent words and part-of-speech tags, the primary procedure is as simple as an API function call that does most of the heavy lifting. A developer should refer to the scripts to understand more about how they operate.

The deep learning procedures such as using the pointer-generator network and the multi-document summarization are more complicated. A developer should refer to the respective GitHub repositories for the minute details of each library, but we'd like to share the data flow for each library that we found to be helpful.

To use the pointer-generator network with Wikipedia, we made a simple script titled "wikipedia_content.py". This script takes in a topic and an output directory. It then strips the content from the Wikipedia article for that topic and stores a .story file in the output directory. The script also handles the call to the script "make_datafiles.py", which is responsible for using the .story file and creating a .bin file, as shown in Figure 4. The .bin file is what the pointer-generator network requires for generating summaries.

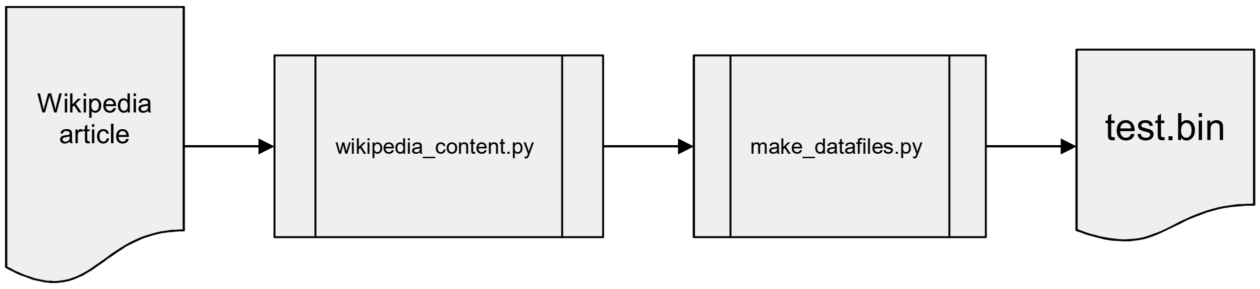


Figure 4: Our process for generating .bin files for the pointer-generator network.

After a .bin file is generated for the content a user wants to summarize, the pointer-generator network repository, a pre-trained network, and the accompanying vocab file can be used to summarize the .bin file. From here on, the procedure for using the pointer-generator network is much simpler. Since the main headaches are found in generating the necessary .bin file, this will make the process easier for future developers. As shown in Figure 5, the pointer-generator is run by using the “run_summarization.py” script with arguments detailed in the GitHub repository.

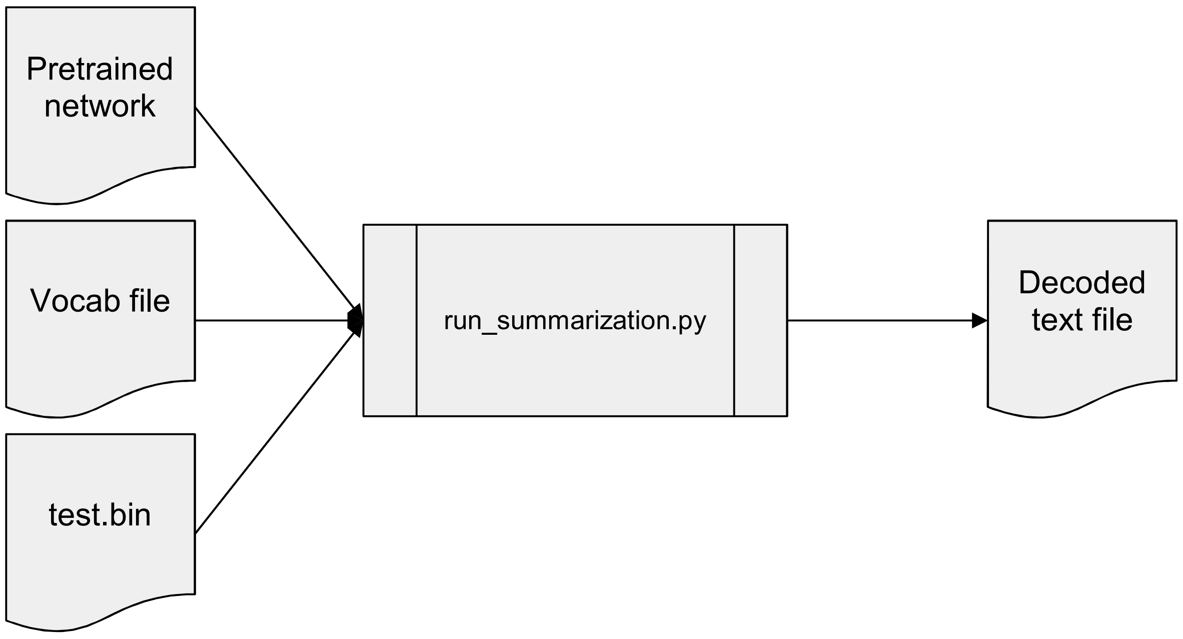


Figure 5: Our process for generating .bin files for the pointer-generator network.

The multi-document library aforementioned is useful for summarizing both large datasets and long articles. We used it to summarize both our small dataset and the Wikipedia article for Hurricane Harvey. As shown in Figure 6 below, the data flow for the multi-document summarization library consists of converting the content that needs to be summarized into a Tensorflow example. A Tensorflow example is a compact, flexible data format that Tensorflow uses for inference and training. It is used here for inference by run_summarization.py.

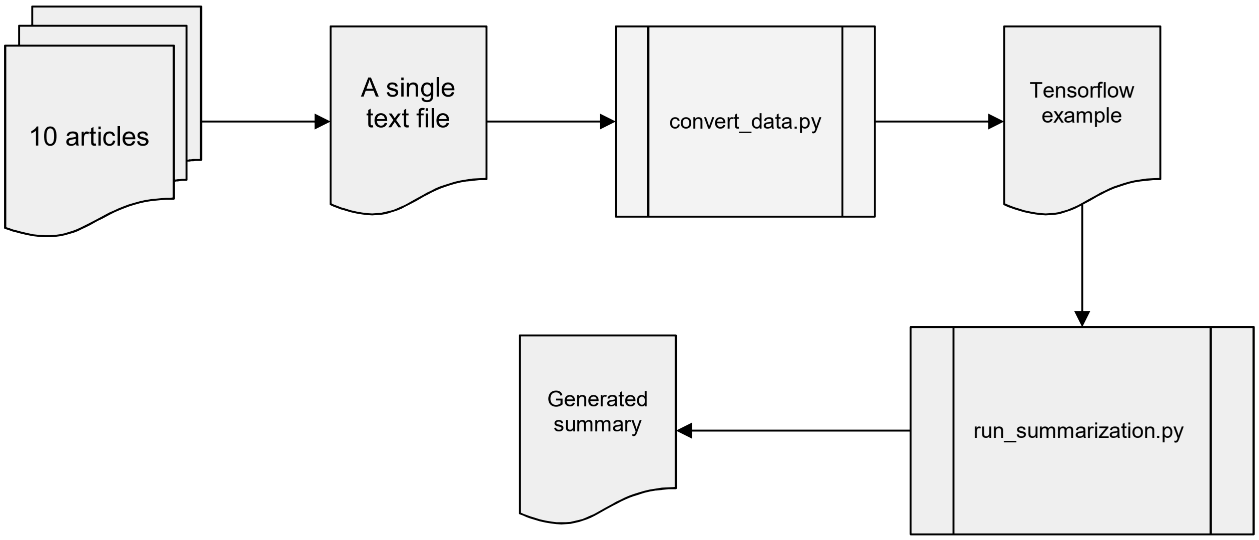


Figure 6: The data flow for the multi-document summarization library. There are many things going on in both of the scripts that a developer should refer to the repository to learn more about.

5.2 Areas of Improvement

We believe a number of areas can be improved in the codebase. First, we didn’t utilize object oriented programming. Since we didn’t have a real overarching layer of abstraction, we just wrote scripts that solved each task individually. This functionality, in hindsight, could be better improved by a Summarizer class, perhaps, that utilizes methodology present in the TextRank, template, pointer-generator network, and multi-document summarizer in a better way.

Secondly, the workflow we followed was prototypical and by no means advanced. In many areas, specifically the pointer-generator network summarization and multi-document summarization workflows, we simply moved files around from folder to folder as needed by parts of the pipeline. We found this to be the easiest, quickest way to get results we needed on a short time budget.

6. Approach, Design, Implementation

6.1 Solr Indexing and Data Cleaning

6.1.1 Conceptual Background

When searching and analyzing text data sets, direct text searches become increasingly resource-intensive and time-consuming as the data set magnitudes increase. A solution to this is to index the text data sets. This solution is analogous to using an index in the back of a book to find specific keywords rather than scanning the entire book.

Web pages scraped from the internet are sometimes not clean and include many websites that are no longer active, irrelevant to the topic we were given, or websites that are dedicated to hosting and captioning images that aren't relevant to text summarization. Also, sites that do contain text, which could be used in the summarization, may contain noise such as solicitation by the author to subscribe, click on other links, and so forth. Thus, web pages scraped from the internet have to be filtered in some form to remove all of the dead links, useless pages, and noise.

6.1.2 Tools

Our primary tool for indexing our data was Apache Solr. Solr is an enterprise search platform that has full-text search and real-time indexing. Solr was recommended to us by Dr. Edward A. Fox; it has good scalability and ease of use that made it an ideal starting tool with which to analyze our data set. Additionally, instructions were provided step by step in the initial weeks of the course.

We used two main tools for cleaning our data: (1) using jusText to remove useless webpages, and (2) removing sentences that contain words we deemed to be noise. jusText is a tool for removing boilerplate content; boilerplate content is material such as headers, footers, navigation links, and ads.

6.1.3 Methodology and Results

For Solr indexing, we used infrastructure prepared for us by the course instructors. To load data from the WARC and CDX files into a Solr index, we first created a Solr friendly JSON file. To do this, we used ArchiveSpark to extract data from the WARC/CDX files and append them to a JSON output. Then we used the output from ArchiveSpark and formatted it using json_formatter.py to make it fit Solr input syntax.

Afterward, we queued the JSON file for indexing into the Solr server using curl, into our designated core. The results were accessible via an admin console located at http://blacklight.cs.vt.edu:8983/solr. We were able to index our small and big data sets into Solr hosted on Blacklight. These results were available to all teams and were used by another team to aid in writing the golden standard for the Hurricane Harvey data set.

For data cleaning we cleaned our data with jusText. Specifically, we made a request to each website URI in the WARC file and passed it into the jusText Python API. When we first tried to implement this, it took way too long to make individual requests, so we applied a multi-threaded approach to the script which made it much faster.

From there, we eliminated stopwords and focused on just the HTML headings and paragraphs of each page to create a JSON output with the format : {“URL”: url, “Title”:title, “Sentences”:sentences}. For an example outline, see Figure 7 below.

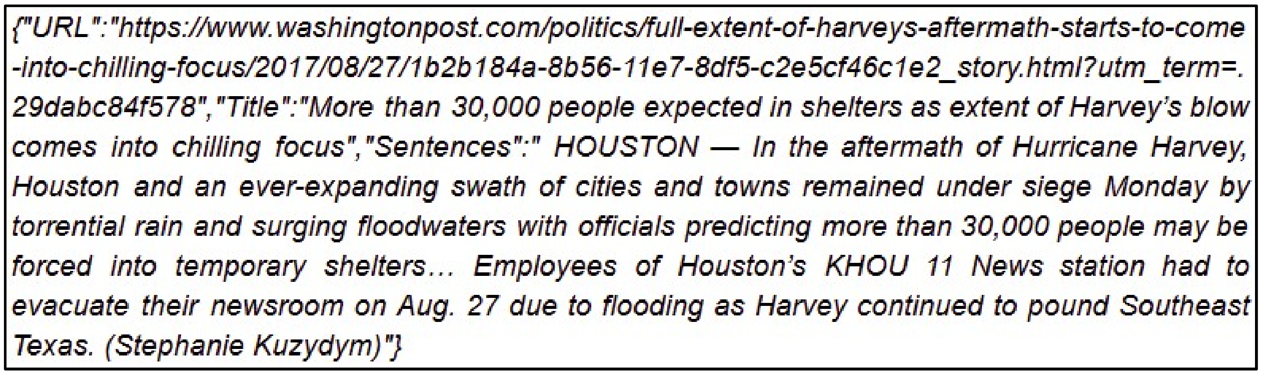

{"URL":"https://www.washingtonpost.com/politics/full-extent-of-harveys-aftermath-starts-to-come-into-chilling-focus/2017/08/27/1b2b184a-8b56-11e7-8df5-c2e5cf46c1e2_story.html?utm_term=.29dabc84f578","Title":"More than 30,000 people expected in shelters as extent of Harvey’s blow comes into chilling focus","Sentences":" HOUSTON — In the aftermath of Hurricane Harvey, Houston and an ever-expanding swath of cities and towns remained under siege Monday by torrential rain and surging floodwaters with officials predicting more than 30,000 people may be forced into temporary shelters… Employees of Houston’s KHOU 11 News station had to evacuate their newsroom on Aug. 27 due to flooding as Harvey continued to pound Southeast Texas. (Stephanie Kuzydym)"}

Figure 7: An example outline for the JSON output from jusText

jusText also analyzed the density of stopwords for each sentence and eliminated the ones with too high of a stopword density. Another additional benefit to jusText is that it was easy to take out web pages that timed out during the request section, which we assumed indicated websites that had been taken down since the creation of the WARC file. In addition, it removed web pages with very short or no paragraphs, which we assumed indicated a removal of an article from a functioning page. Finally, jusText weeded out pages which were not in English.

Our results came in the form of a JSON file named “jusText_output.json.” It had weeded out about 30% of the small WARC set and about 80% of the big WARC set. We were frankly surprised to see how much of the big WARC set the script eliminated.

6.1.4 Functionality

The functionality of this script requires the installation of NLTK, jusText, requests, and WARC in Python. We also require the location of the warc.gz file to process on line 12 of our script.

6.1.5 Deliverables

Our deliverable for this section is our "jusText.py" script as well as “jusText_output.json” which is the output of our script.

6.2 Most Frequent Words

6.2.1 Conceptual Background

Counting the frequency of words in a set of webpages is an important first step for summarization tasks and getting started with natural language processing in general. The most frequent words task is really a precursor to part-of-speech tagging, named entity recognition, and template filling tasks that were completed later on.

A good example to demonstrate this task’s usefulness is counting the most frequent words in the first two sentences of the Wikipedia page for Hurricane Harvey shown in Figure 8 below [13].

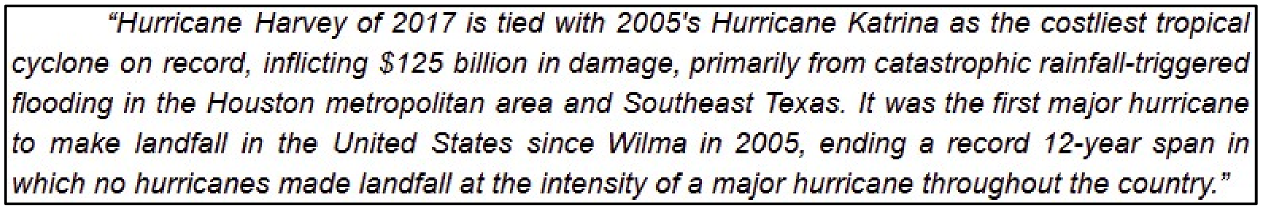

“Hurricane Harvey of 2017 is tied with 2005's Hurricane Katrina as the costliest tropical cyclone on record, inflicting $125 billion in damage, primarily from catastrophic rainfall-triggered flooding in the Houston metropolitan area and Southeast Texas. It was the first major hurricane to make landfall in the United States since Wilma in 2005, ending a record 12-year span in which no hurricanes made landfall at the intensity of a major hurricane throughout the country.”

Figure 8: The first two sentences from the Wikipedia page for Hurricane Harvey.

By reading over this paragraph, certain words occur frequently, including the word *hurricane*, which is the most frequent non-stopword. Thus, it may be useful as a first step to determine the most frequent words in our dataset.

6.2.2 Tools

For reading in the small dataset, which we used in this section, we used the JSON Python library. The JSON library eased the usability of .json files and helped us speed up the development process throughout the entire semester.

Another tool used in this step was the Natural Language Toolkit (NLTK) library. The NLTK library is an open source tool for natural language processing and supported the required operations necessary to count our most frequent words. NLTK was the default library recommended to us, but there are other libraries used later which perform better on specific tasks.

6.2.3 Methodology and Results

In developing our most frequent words script, we first tried operating on the raw text stored in the JSON file. The JSON file contains around 500 articles with sentences, including punctuation, stop words, and other information. We simply concatenated all of the text from each article together into a single string.

However, when counting the most frequent words in this string using the FreqDist class in the NLTK library, we came across many stop words such as *a*, *in*, and *the*, which, though very frequent, were not relevant to our task. Thus, the first change we made to our methodology was to filter out those stopwords with Python. To do this, we used the NLTK stopword corpus that contains many of the stopwords in the English language. By taking only the words in our string that did not belong to the stopword corpus we were able to filter them out successfully.

Another issue with this procedure is that punctuation remains that will cause the NLTK library to count “Harvey” and “Harvey.” as different words. To filter out punctuation we had to make a translator with Python using the maketrans function and a set of relevant punctuation, and then use this translator to remove all of the punctuation from our text. As an example for steps 1 and 2 in our methodology, take the following string:

“The cat jumped over the yellow fence.”

This string, through the process we implemented, would become as follows:

“cat jumped over yellow fence”

After removing stop words and punctuation, we can now correctly operate on the text to get the following output for the five most frequent words in our small dataset. As shown in Table 1, we found highly relevant words to be most frequent, which we believe signifies our methodology works. However, *said* is not relevant to Hurricane Harvey and appeared most likely because many people were quoted for news articles.

Table 1: Top 5 most frequent words with the number of times they appeared in the dataset

Word	**Frequency**
harvey	424
houston	422
texas	330
hurricane	311
said	214

6.2.4 Functionality

The functionality of our script for processing the most frequent words requires some assumptions. The first is that we are operating on a JSON file that contains data labeled with “sentences,” which we use to retrieve the data. To make sure this worked properly we only ever used the JSON file we retrieved with ArchiveSpark. Thus, the second assumption is that the user runs the script with the "small_dataset.json" file in the same directory. We did not extend the script to be generalizable because we wanted to focus on more difficult and rewarding tasks.

6.2.5 Deliverables

Our deliverable for this section is our “freq_words.py” script, which is usable alongside the “small_dataset.json” file.

6.3 POS Tagging

6.3.1 Conceptual Background

Part-of-speech (POS) tagging is an important methodology in natural language processing that allows for categorizing words into categories known as parts-of-speech. When a word belongs to a given part-of-speech category, it has similar grammatical properties as other words in that same category. The problem isn’t as easy as using a table with words and a respective part-of-speech tag because words can have several different part-of-speech tags based on how they are used in as sentence. Because of this, POS tagging is not a trivial problem and has a number of edge cases that are difficult to resolve. But for the purpose of using this technique on news articles, we consider POS tagging to be a solved problem for all of the cases we come across.

As an example of POS tagging, we can consider the example we used in the previous section:

“The cat jumped over the yellow fence.”

In this sentence, we can label “cat” as a noun, “jumped” as a verb, “yellow” as an adjective, and “fence” as a noun. This list isn’t exhaustive, but it demonstrates the principle. However, in NLP tasks this is not good enough, and we typically would use more specific labels. For example, our sentence from before can be labeled more extensively, as shown in Table 2.

Table 2: More specific part-of-speech tags using part-of-speech tagging library TextBlob.

Word	**POS**
The	DT
cat	NN
jumped	VBD
over	IN
the	DT
yellow	JJ
fence	NN

These tags are more useful for computational tasks. For example, NN means noun singular. It is helpful to recognize that nouns like cat and fence are plural or singular, which this form of POS tagging handles for us. The meaning of the sentence changes if the nouns change from singular to plural. Also, VBD means verb past-tense. So in addition to recognizing that “jumped” is a verb it also determines that this occurred in the past. Thus, we are beginning to find semantics in our POS tags for this sentence, which could be useful in summarization.

6.3.2 Tools

In addition to the NLTK and JSON libraries for Python, we also used the library TextBlob. TextBlob is a simple library for some NLP tasks such as POS tagging, sentiment analysis, and classification. TextBlob is easy to use. All that is required is text from the combined sentences we used in the previous section.

6.3.3 Methodology and Results

Unlike before, we do not need to remove punctuation or stopwords. Instead, we make a TextBlob out of our sentences and then look at the TextBlob tags property to find the POS tags.

In doing this, we looked specifically for all of the nouns and all of the verbs. In Table 3, we present the first five nouns with their POS tags found in the small dataset. Also, in Table 4, we show the first five verbs with their POS tags.

Table 3: The first 5 nouns in the small dataset with their respective POS tags using TextBlob.

Houstonians	NNPS
Landsat	NNP
area	NN
storm	NN
Brazos	NNP

Table 4: The first 5 verbs in the small dataset with their respective POS tags using TextBlob.

are	VBP
having	VBG
decide	VB
stays	VBZ
goes	VBZ

The primary purpose of this task was to gain knowledge in how to apply POS tagging to our dataset. We used similar programming tactics in named entity recognition for our templated summary.

6.3.4 Functionality

The function of our "pos_tagging.py" script is to take the nouns and verbs from the tags property of the TextBlob object that takes in the sentences as input. Further functionality could be implemented, but we believed that this task was instrumental in gaining more experience for actually generating summaries later on. Thus, the function takes in the "small_dataset.json" file and returns that nouns and verbs. It is not generalized or extended to other data.

6.3.5 Deliverables

For this section, our deliverable is the "pos_tagging.py" script which is usable with the "small_dataset.json"

6.4 Cloud Computing (ARC and Hadoop)

6.4.1 Conceptual Background

To generate good results, thousands of documents require preparation and processing. The data also needs to be shared among teammates. Cloud computing plays a vital role in big data analysis. With ARC, we planned to run training and processing quickly for days without worrying about a power outage, and with Hadoop, we planned on leveraging the power of the cluster to perform our cleaning much quicker than on one of our local machines.

6.4.2 Tools

ARC's Cascades system provides us with a powerful platform with more than 40 GPUs for machine learning. We used Cascades to experiment with the pointer-generator network to see how extensive the computational requirements were.

To clean thousands of pages of data, we expected to need to leverage the power of the Hadoop Cluster to complete the cleaning process promptly. The DLRL Hadoop Cluster is a powerful machine leveraging 88 CPU cores and 704GB of memory. We experimented with using the Hadoop Cluster to perform our cleaning and some processing but ultimately decided against it.

6.4.3 Methodology and Results

To run codes using TensorFlow for machine learning, the user will need to create a job and submit it to the job queue. By default, ARC will collect all standard output and standard error messages and store them in a file once the session is done. This makes debugging very inefficient. The program could not be running at all but generating an error message. The only way to tell is ending the job session and checking out the error log. Further details about this situation are mentioned down in the challenges session.

To run code on the Hadoop Cluster, we needed to utilize PySpark. PySpark's libraries for reading .json files do so by creating an object which is queried like a SQL database. Since all of our code was already written to utilize Python JSON libraries, we would need to refactor all of our code to be able to use the product of the cleaning on the Hadoop Cluster. Also, we would have to wait several minutes between script executions, making debugging slow. More details on the situation and our solutions to the challenges are mentioned down in Section 8.2.2 and Section 8.3.2, respectively.

6.5 TextRank Summary

6.5.1 Conceptual Background

TextRank is an unsupervised graph-based algorithm for sentence extraction [12]. It is useful in dealing with performing extractive summarization with raw data because it doesn't require training data. PageRank, used by Google search, is the precursor for TextRank and has shown success in giving structure to the internet.

Since TextRank is a graph-based algorithm, the data structure it deals with is a graph with vertices (nodes) and edges (connections between nodes). It consists of an algorithmic determination of the most useful vertices in a graph by finding the vertices with the most votes. Votes are just the number of connections to other nodes, meaning the most important vertices will have the highest number of links.

The algorithm begins by assigning random values to each of the sentences in our collection. It then iterates, changing node assignments, until some threshold is reached. The final values of the nodes do not depend on the random initial assignment, but on the number of iterations. In the end, there will be a set of sentences we can choose from which have the highest values. Using these sentences allows us to generate a summary of the text.

6.5.2 Tools

In this section, we used the Summa library that has a TextRank summarizer. The Summa library is simple to use and contains a summarizer class that only requires the sentences as a string and the number of words we want in our summary.

In addition to the Summa library, we also used the JSON library. Neither the NLTK library nor the TextBlob library were required.

6.5.3 Methodology and Results

We combined all of the sentences into one long string and then used the summarize method that is part of the summarizer class to summarize the text with a word limit of 100 words. There was some experimentation that went into this task because we had to find a good number of target words through testing. We first started with a word limit of 500 words, but this contained many repeated sentences. By reducing the number of words to 100, we found less repeated sentences appeared and the overall summary was more consistent. In addition, we had an issue where “100-year floodplain” was included in every sentence, so we decided to remove any sentence that contained that phrase and this reduced repetition. A diagram of the final process is shown in Figure 9.

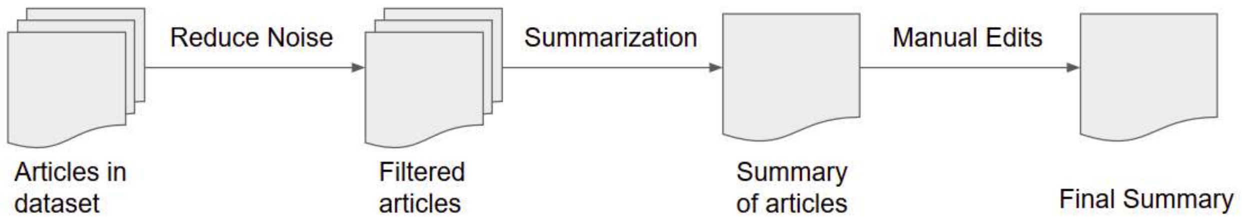
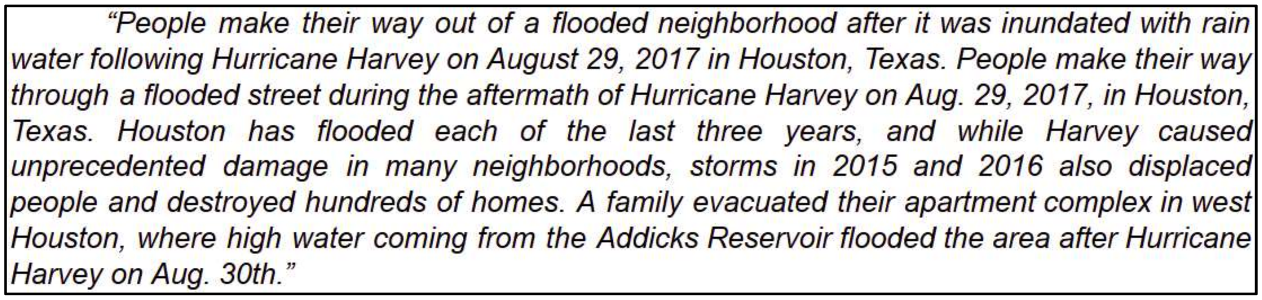


Figure 9: Details of the TextRank summarization process.

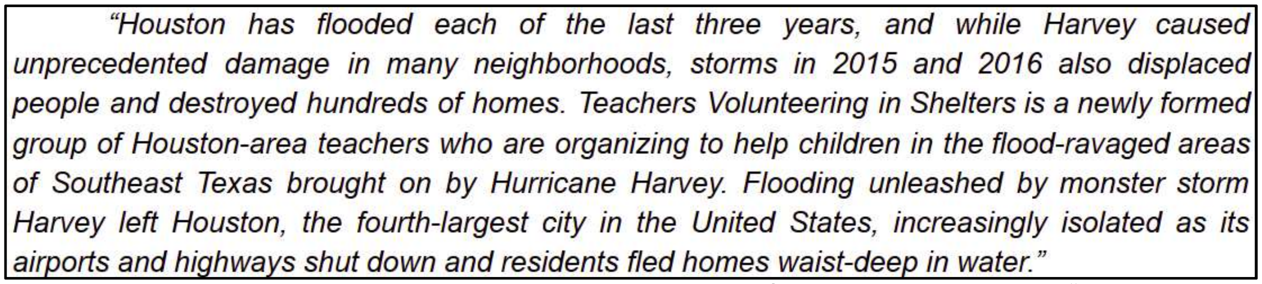
We first started with a 500-word limit, but this created extensive, jumbled summaries that contained single sentences. We also had an issue where certain sentences would repeat themselves because they were so similar, as shown in the text summary below in Figure 10.



"People make their way out of a flooded neighborhood after it was inundated with rain water following Hurricane Harvey on August 29, 2017 in Houston, Texas. People make their way through a flooded street during the aftermath of Hurricane Harvey on Aug. 29, 2017, in Houston, Texas. Houston has flooded each of the last three years, and while Harvey caused unprecedented damage in many neighborhoods, storms in 2015 and 2016 also displaced people and destroyed hundreds of homes. A family evacuated their apartment complex in west Houston, where high water coming from the Addicks Reservoir flooded the area after Hurricane Harvey on Aug. 30th."

Figure 10: The first summary generated by TextRank with a 100 word limit.

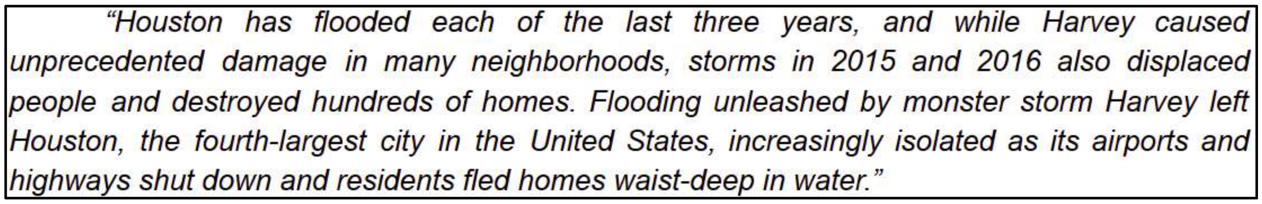
To avoid this, we just removed the keyword "People" from our string, and that removed those sentences from the dataset. After we did this extra processing step, we were able to get a better summary shown below in Figure 11.



"Houston has flooded each of the last three years, and while Harvey caused unprecedented damage in many neighborhoods, storms in 2015 and 2016 also displaced people and destroyed hundreds of homes. Teachers Volunteering in Shelters is a newly formed group of Houston-area teachers who are organizing to help children in the flood-ravaged areas of Southeast Texas brought on by Hurricane Harvey. Flooding unleashed by monster storm Harvey left Houston, the fourth-largest city in the United States, increasingly isolated as its airports and highways shut down and residents fled homes waist-deep in water."

Figure 11: The second summary using TextRank after removing the word "People".

This is a decent summary, considering it was done with an unsupervised method and just had raw sentences as input. It has a total of 3 sentences, but only the first and last related to one another. The first and last are about the damage seen in Houston during Hurricane Harvey, but the second deals with a group helping the recovery efforts. Thus, the unsupervised TextRank method is decent, but not good enough. If we manually filter out the second sentence, we can get a better summary as shown below in Figure 12.



"Houston has flooded each of the last three years, and while Harvey caused unprecedented damage in many neighborhoods, storms in 2015 and 2016 also displaced people and destroyed hundreds of homes. Flooding unleashed by monster storm Harvey left Houston, the fourth-largest city in the United States, increasingly isolated as its airports and highways shut down and residents fled homes waist-deep in water."

Figure 12: The TextRank summary in Figure 11 with manual editing to remove off-topic sentences.

The above summary is better but had manual editing, so it is not entirely unsupervised.

This trade-off between summary quality and supervision seems to be a common theme in the other two summarization methods as well. In the template-based summary, we supervise the summarization by creating the template and choosing proper slots that go into those templates, and in deep learning, we require a large amount of labeled data that we can use to train a neural network.

6.5.4 Functionality

Our script, “textrank_summarizer.py” is rather simple. It takes in the JSON file for our small dataset and then inputs the text extracted from the JSON file into the Summa library’s summarizer class. Beyond this, there isn’t much generalizability at play here apart from the potential to fill in the sentences with whatever is desired.

6.5.5 Deliverables

Our deliverables for this section include “textrank_summarizer.py” and the summaries listed above.

6.6 Template Summary

6.6.1 Conceptual Background

Templates are summaries with missing entities that must be filled in by searching through data. The missing entities are typically found with either regular expressions or named entity recognition. Once the information is filled in, there is a generated summary that has templated information supplied by the user, which is rounded out with important entities filled in with data. Conceptually, the template summary is much simpler to understand than TextRank or deep learning. It is like a form filled out by a computer.

6.6.2 Tools

One of the tools used in this section is regular expressions. A regular expression is a guideline used to find patterns in text that we think would be useful for some entity in our template. Regular expressions are used mainly in this part to count the number of occurrences of words and phrases such as “Hurricane Harvey” and “Category 4”. We can construct regular expressions which find occurrences of “Hurricane” plus a random string that follows and then count up the number of cases we find. For example, we could find “Hurricane Katrina” mentioned five times and “Hurricane Harvey” mentioned 40 times. We would take “Hurricane Harvey” as the value for an entity in our template.

The other tool we used is named entity recognition that can be done through the spaCy library. Named entity recognition is similar to POS tagging in that you will retrieve a word or phrase with a tag associated with it. However, the labels are entirely different and refer to entities instead of parts of speech. Some examples of entities that can be named with the spaCy library are ORG, GPE, and MONEY. As an example of how this would work, one could feed in a text about Google and values such as Google could be associated with ORG and $30 billion could be associated with MONEY. We can see how this might be useful in our case, since many organizations like FEMA (ORG) may have responded to Hurricane Harvey, it made landfall in Houston, Texas (GPE), and incurred billions of dollars (MONEY) in damage.

6.6.3 Methodology and Results

Our approach for this section is based on using regular expressions or named entity recognition to find the occurrence of essential entities in our data. Once we have seen those entities, we make sure we keep track of how many there are. We then use the most frequent occurrence in our template-based summary. Figure 13 below demonstrates this process.

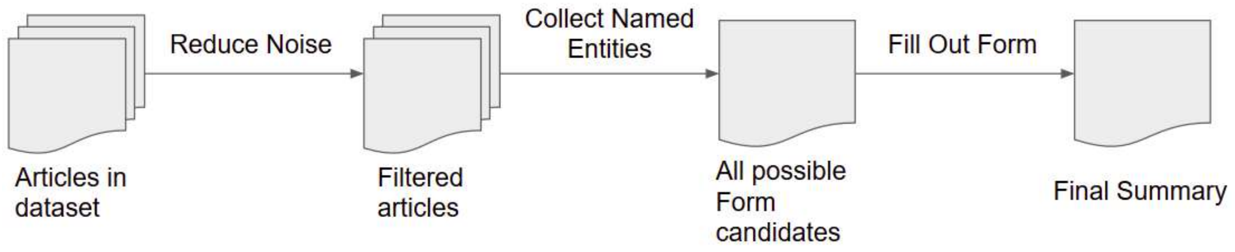
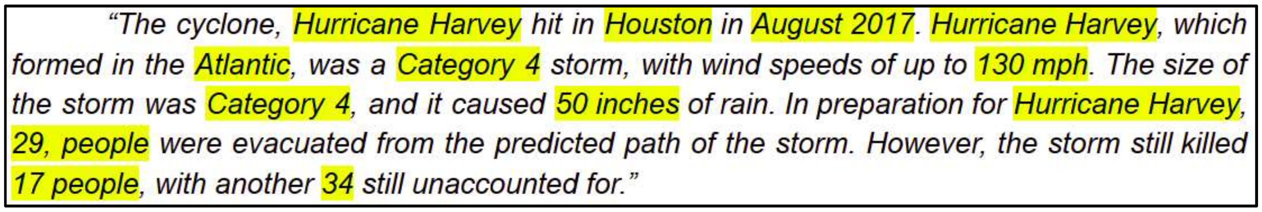


Figure 13: A diagram demonstrating template based summarization.

As an example, we generated a summary with regular expressions using a previous team's source code that we modified [16]. As shown in the summary below in Figure 14, entities which are highlighted represent those which were found with regular expressions.



"The cyclone, Hurricane Harvey hit in Houston in August 2017. Hurricane Harvey, which formed in the Atlantic, was a Category 4 storm, with wind speeds of up to 130 mph. The size of the storm was Category 4, and it caused 50 inches of rain. In preparation for Hurricane Harvey, 29, people were evacuated from the predicted path of the storm. However, the storm still killed 17 people, with another 34 still unaccounted for."

Figure 14: The initial template summary using a previous team's software.

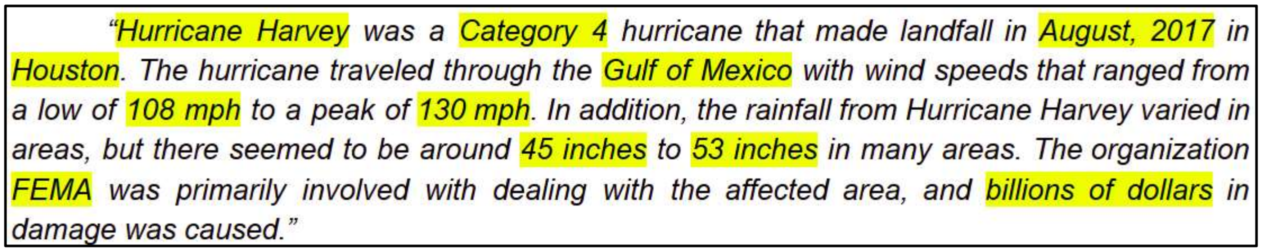
The above summary was generated by scanning the data with regular expressions and keeping track of the best (most frequent) occurrences for a given regular expression. For example, the regular expression for finding the name of the hurricane is:

"(Hurricane)\s*[A-Z][a-z]*"

This regular expression is run through the text and when it comes across an occurrence of "Hurricane __________" it will add to the count of that string in a dictionary. Thus, every mention of Hurricane Harvey, Hurricane Katrina, and Hurricane Irma will be caught when scanning the text, but only the one mentioned the most would contribute to the summary at the end.

There are some issues with regular expressions, however. The above summary has some errors such as “29, people” as the number of people evacuated, and “17 people” as the number of people killed. Also, “34” as the number of people unaccounted for is inaccurate. Apart from inaccuracies, we are also missing important information that may be relevant, such as the amount of damage Hurricane Harvey caused and the organizations involved in the process.

Named entity recognition helped us overcome these challenges by making it easier to find entities for damage and the organizations, as well as a range for the amount of rainfall and wind speeds. Our new summary shown below in Figure 15 demonstrates this.



“Hurricane Harvey was a Category 4 hurricane that made landfall in August, 2017 in Houston. The hurricane traveled through the Gulf of Mexico with wind speeds that ranged from a low of 108 mph to a peak of 130 mph. In addition, the rainfall from Hurricane Harvey varied in areas, but there seemed to be around 45 inches to 53 inches in many areas. The organization FEMA was primarily involved with dealing with the affected area, and billions of dollars in damage was caused.”

Figure 15: The improved template summary using named entities and spaCy.

In addition to no longer having incorrect information, more information is present in the summary above, including a mention of FEMA and the amount of damage caused. We also now have ranges for the wind speeds and amount of rainfall. Searching through our small dataset, we could not find numbers for how many people died or were evacuated. We don’t think these values are present, so we cannot include them in our summary.

To construct the above summary a similar methodology to the regular expressions approach was followed. We found named entities in our dataset and then grouped entities that were the same and kept track of how many times they appeared. Whereas before it was difficult to find organizations or cost, there is a named entity for both (ORG and MONEY) that can be used in our methodology. For ranges, we used matplotlib’s hist function to auto-generate bounds and chose the most frequently filled bin’s ranges for the summary. The ranges are accurate, and they serve as good ranges for the quantities in our summary above.

6.6.4 Functionality

The script we wrote for this section is “template_summarizer.py”. It can be run when it is given the path to “small_dataset.json”.

6.6.5 Deliverables

Our deliverables for this section are the summaries above and the file “template_summarizer.py”.

6.7 Abstractive Summary with the Pointer-Generator Network

6.7.1 Conceptual Background

Pointer-generator networks are a recent development in natural language processing. The networks are abstractive and are nearly the state-of-the-art in abstractive summarization.

Pointer-generators are built upon sequence-to-sequence models with attention. Similarly to sequence-to-sequence models, pointer-generators have encoders and decoders. According to a post made by the author of pointer-generator networks [5], the encoder first acts to read the source text and outputs a sequence of encoder hidden states. Once the encoder has read the entire document, the decoder begins to print out a series of words that form the summary.

To begin with, the decoder prints out a <START> token. Afterward, we start to look at the previously printed word to figure out what word we should write next. An attention distribution is formed using the decoder hidden state. The decoder hidden state is updated based on the previous word that was printed, and this, in turn, is used to form the attention distribution over the words in the source text. The decoder hidden state and a context vector based on the encoder hidden states are then used to construct the vocabulary distribution. At this stage in sequence-to-sequence models, we typically choose the highest probability word and print it out to continue our summary. However, in pointer-generator networks, we introduce what is known as the generation probability.

The generation probability is what makes the pointer-generator network special. It allows for the combination of the attention distribution and the vocabulary distribution so that we can sometimes choose from our attention distribution as well as from our vocabulary distribution.

Thus, pointer-generator networks allow for us to extract words from the attention distribution, which is the source text. This method enables a pointer-generator network to operate with a smaller vocabulary and out-of-vocabulary words because it doesn’t have to rely on only the vocabulary distribution. In addition to these improvements, adding a mechanism known as coverage can ensure that the pointer-generator does not repeat itself. Coverage is a trick for the pointer-generator network to keep track of what it has already printed. If this is active, then the pointer-generator network doesn’t repeat itself, and robust, readable summaries can be generated.

6.7.2 Tools

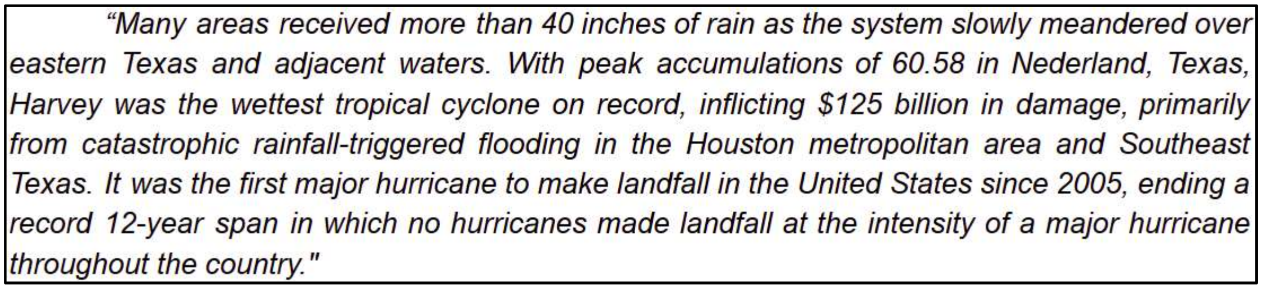
The tools we used in this section is the GitHub repo that originated the pointer-generator network [11]. We also used the Wikipedia library for Python to extract content from Wikipedia articles. There is a tool developed by the author of the pointer-generator network for converting text into a .bin file for processing by the pointer-generator network [14]. We modified this tool to handle Wikipedia content that we extracted. The two files for this purpose are “wikipedia_content.py” and “make_datafiles.py”.

6.7.3 Methodology and Results

The small and large datasets are difficult to summarize with the pointer-generator network because it is designed to operate on single articles. Our initial attempts at synthesizing the small and large datasets did not go well. The summaries were off-topic, jumbled, and generally incoherent. In particular, the pointer-generator network is not meant to summarize large amounts of content. We used the neural network in a naively to see what summary it would produce. The summary we generated focused on content available in a single article, but ignored the remaining content in the other articles.

To fix this, we simplified our problem only to summarize the Wikipedia page of Hurricane Harvey. This change simplifies the problem greatly by allowing us to summarize a single article, instead of an extensive database. We simplified the problem mainly to speed up development and avoid building a significant pipeline for churning through the small or large datasets.

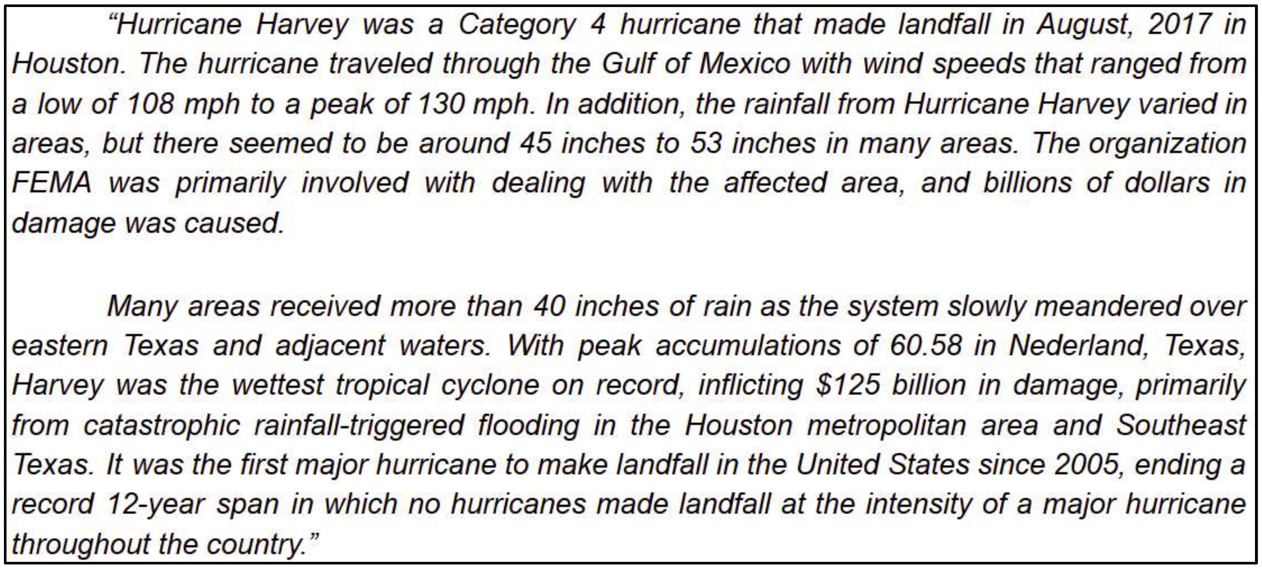
Using the Wikipedia article, we generated a summary that we believe is concise and explains the topic well, as shown in Figure 16. It is also readable and on-topic, instead of the summaries we generated from the small and large datasets with the pointer-generator network.



"Many areas received more than 40 inches of rain as the system slowly meandered over eastern Texas and adjacent waters. With peak accumulations of 60.58 in Nederland, Texas, Harvey was the wettest tropical cyclone on record, inflicting $125 billion in damage, primarily from catastrophic rainfall-triggered flooding in the Houston metropolitan area and Southeast Texas. It was the first major hurricane to make landfall in the United States since 2005, ending a record 12-year span in which no hurricanes made landfall at the intensity of a major hurricane throughout the country."

Figure 16: The summary generated by the pointer-generator network with the Wikipedia article as the input.

We think this summary is readable and has good coverage of the topic at hand. Similarly to the summary before, there are mentions of rainfall amounts and the amount of damage the storm incurred. However, there are some issues with this summary. First, there is no mention of Hurricane Harvey until the second sentence, so it is hard to know what the summary is talking about. Even when the name of the storm is mentioned, it is only called Harvey and referred to as a tropical cyclone. Thus, we believe to improve this summary we can combine it with our previous extractive summary that was mentioned in the previous section, as shown in Figure 17.



"Hurricane Harvey was a Category 4 hurricane that made landfall in August, 2017 in Houston. The hurricane traveled through the Gulf of Mexico with wind speeds that ranged from a low of 108 mph to a peak of 130 mph. In addition, the rainfall from Hurricane Harvey varied in areas, but there seemed to be around 45 inches to 53 inches in many areas. The organization FEMA was primarily involved with dealing with the affected area, and billions of dollars in damage was caused.

Many areas received more than 40 inches of rain as the system slowly meandered over eastern Texas and adjacent waters. With peak accumulations of 60.58 in Nederland, Texas, Harvey was the wettest tropical cyclone on record, inflicting $125 billion in damage, primarily from catastrophic rainfall-triggered flooding in the Houston metropolitan area and Southeast Texas. It was the first major hurricane to make landfall in the United States since 2005, ending a record 12-year span in which no hurricanes made landfall at the intensity of a major hurricane throughout the country."

Figure 17: The template summary and the pointer-generator network summary combined.

This improved summary is much more comprehensive and covers many more facts about the storm. We believe this serves as a final summary that can aptly summarize the event. The template summary serves as an overall summary, whereas the pointer-generator network goes into more depth.

6.7.4 Functionality

The scripts we supply, "wikipedia_content.py" and "make_datafiles.py", allow for extracting content from Wikipedia for a given topic and saving it as a .bin file. It is necessary to use the pre-trained pointer-generator network, a vocab file, and other instructions from the GitHub repo for the code. Section 6 will explain this in more detail.

6.7.5 Deliverables

Our deliverables for this section include "wikipedia_content.py", "make_datafiles.py", and the summaries we presented above.

6.8 Multi-document Summary of the Small Dataset and Wikipedia Article

6.8.1 Conceptual Background

To summarize multiple documents, the multi-document summarization library uses an extractive algorithm called maximal marginal relevance or MMR, a near state-of-the-art extraction algorithm. MMR works iteratively by selecting a sentence at each iteration that maximizes a score of the difference between the importance of the sentence and its redundancy compared to what has already been chosen.

In [9], the authors designed the pointer-generator to work alongside MMR. Their new system is called PG-MMR. At each iteration, PG-MMR selects some sentences from the source material, summarizes them with the pointer-generator and then adjusts the remaining content in the source material to reflect new measures of importance and redundancy. MMR assists the pointer-generator network by extracting important and non-redundant information that the pointer-generator should summarize to make a better summary.

6.8.2 Tools

We used the multi-document summarization repository [9], [10]. It allows for the summarization of up to 10 related news articles.

6.8.3 Methodology and Results

To summarize the small dataset, we found every article in that which contained the word “Harvey.” After removing articles that didn’t contain the word "Harvey," we had approximately 160 articles. The multi-document summarization library takes in up to 10 articles per summary. Thus, we summarized the nearly 160 articles into 16 summaries. We then took those summaries and generated two summaries of the summaries. Finally, we summarized those two summaries go into a final summary. The process we followed is shown in Figure 18.

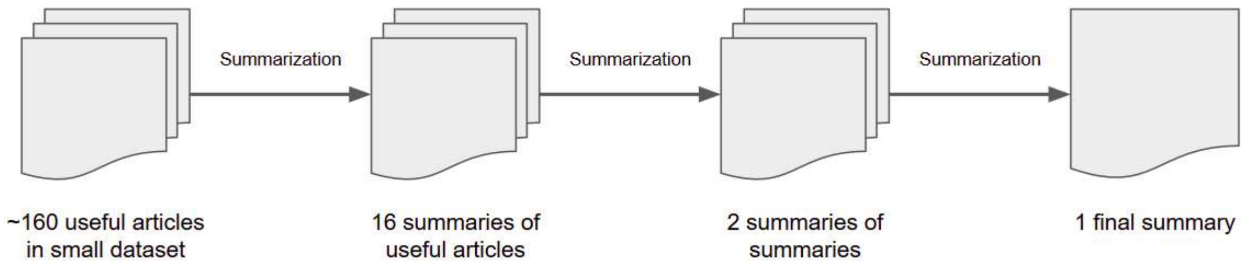


Figure 18: Processing steps for the small dataset summary using the multi-document summarization library.

The output of this procedure is shown below in Figure 19. Reading over it, it seems to focus on the topic of Hurricane Harvey, and mention how catastrophic and massive the storm was. However, it is devoid of facts about the hurricane and instead remains quite vague. Also, the final sentence doesn’t make much sense.

> *“Harvey is now the benchmark disaster of record in the United States. If past disasters are any indication, those numbers will only grow in the coming days and weeks. In fact, Harvey is likely already the worst rainstorm in U.S. history, but Harvey is in a class by itself. As a result, Harvey is the third 500-year flood to hit the houston area in the past three years, who have a federal flood policy in place. But a Category 4 hurricane and has lingered dropping heavy rain as a tropical storm.”*

Figure 19: The summary generated with the multi-document summarization library with the small dataset as the input.

Though this summary is short and doesn’t include many pertinent details about Hurricane Harvey, it is quite incredible that a readable output that stayed on-topic was generated at all. Very little pre-processing was used to reach this summary apart from making sure that “Harvey” was present in each article used as source material. It is also a significant reduction in the source material. If each item of the small dataset only contains 50 words, then it is a near 99% reduction in source material length.

Following this summary, we were interested in how much better the Wikipedia article would be in serving as a source of information for the multi-document summarization approach. Thus, we manually extracted content from sections of the Wikipedia article and used those as source material. In total, around 50 “articles” of content were generated from the Wikipedia article. We then utilized the library to summarize all of the information into 5 summaries. Once we had those summaries, we appended them together to generate the final summary. As shown in Figure 20 below, our procedure is rather simple.

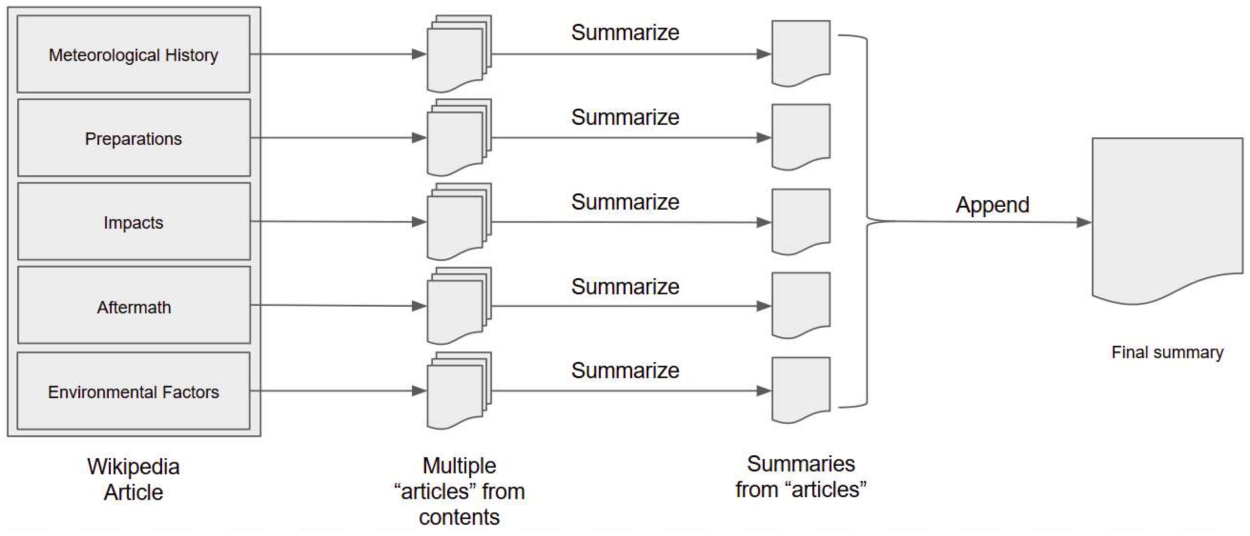


Figure 20: Processing steps for the Wikipedia article summary using the multi-document summarization library.

Below we present our final summary of Hurricane Harvey using the Wikipedia article and the multi-document summarization library. The total word count for the input was 4466 words, and the output contains 487 words, which is an 89% reduction. The summary is shown in Table 5 below.

Table 5: The summary generated by the multi-document summarization library given the Wikipedia article as input.

"Harvey developed from a tropical wave to the east of the lesser antilles, reaching tropical storm status on August 17. The storm crossed through the windward islands on the following day, making landfall on the southern end of barbados and a second landfall on Saint Vincent. Harvey began to weaken due to moderate wind shear, and degenerated into a tropical wave north of Colombia, late on August 19. Harvey continued to quickly strengthen over the next day, ultimately reaching peak intensity as a Category 4 hurricane. The system reached the bay of campeche in the Gulf of Mexico on August 23.

FEMA placed disaster response teams on standby at emergency posts in Austin, Texas, and Baton Rouge, Louisiana. Upon the NHC resuming advisories for Harvey at 15:00 UTC on August 23, a hurricane watch was issued in texas from port mansfield to San Luis pass, while a tropical storm watch was posted from Port Mansfield south to the mouth of the Rio Grande and from San Luis pass to high island. Additionally, a storm surge watch became in effect from Port Mansfield to the Rio Grande; a tropical storm warning was in effect for the entire Gulf Coast of Texas from high island northward.

An estimated 300,000 structures and 500,000 vehicles were damaged or destroyed in Texas alone. The National Oceanic and Atmospheric administration estimated total damage at $125 billion, with a 90 % confidence interval of $ 90–160 billion. This ranks harvey as the costliest tropical cyclone on record in the country alongside Hurricane Katrina in 2005. However, accounting for inflation and cost increases since 2005, the national hurricane center considers Harvey the second-costliest. Nationwide, 2017 approximately 13,000 people had been rescued across the state while an estimated 30,000 were displaced. More than 20 percent of refining capacity was affected.

President Trump made a formal request for $5.95 billion in federal funding on August 31, the vast majority of which would go to FEMA. Other states' national guard's have offered assistance, with several having already been sent. By August 30, corporations across the nation collectively donated more than $72 million to relief efforts. In September 2017, the Insurance Council of Texas estimated the total insured losses from Hurricane Harvey at $19 billion. This figure represents $ 11 billion in flood losses insured by the national flood insurance program.

The geography of Houston brings very heavy rainfall. There is a tendency for storms to move very slowly over the region. There have been repeated floods in the city ever since

2010. As Houston has expanded, rainwater infiltration in the region has lessened and aquifer extraction increased, causing the depletion of underground aquifers. When the saturated ground dries, the soil can be compressed and the land surface elevation decreases in a process called subsidence. Within a week of Harvey, Hurricane Irma formed in the eastern atlantic, due to be a factor in Harvey 's impact.”

Reading the summary, we can see that pertinent facts are present as derived from each section of the Wikipedia article included in the source material. Although this doesn’t utilize the small or large datasets, we believe it achieves the goal of generating a summary of Hurricane Harvey.

6.8.4 Functionality

The repository supports the summarization of a text file with sentences of articles on each line and each article separated by an empty line. For more information, see the repository’s README [10].

6.8.5 Deliverables

Our deliverables for this section are the two summaries presented above.

7. Evaluation and Assessment

7.1 Summary Evaluation and Assessment with Gold Standard

To evaluate our summaries, we used a script given to us by the instructors of the course. It utilizes the ROUGE evaluation metric. There are three main methods for summary evaluation that we followed using the script given to us.

The first is ROUGE evaluation, which tests ROUGE-1 (1-gram, words), ROUGE-2 (bigrams), ROUGE-L (longest common subsequence), and ROUGE-SU4 (skip-bigram and unigram). The scores for our summaries are shown in Table 6.

Table 6: ROUGE evaluation scores for our 5 summaries.

Method	**Type**	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**	**ROUGE-SU4**
TextRank with small dataset	Extractive	0.06897	0.0	0.06897	0.01266
Template with small dataset	Extractive	0.2069	0.07143	0.2069	0.05696
PGN with Wikipedia article	Abstractive	0.06897	0.0	0.06897	0.01266
Multi-document with small dataset	Abstractive	0.03448	0.0	0.03448	0.00633
Multi-document with Wikipedia article	Abstractive	**0.51724**	**0.21429**	**0.34483**	**0.24684**

Our evaluation metrics for the template summary is the best for the two extractive summaries, while our multi-document summarization of the Wikipedia article is the best for our abstractive summaries.

To improve the results of our other summaries, we believe it would be pertinent to extend the length of each summary. Each summary, excluding the multi-document summary of the Wikipedia article, are around 100 words in length, whereas the Gold Standard is approximately 500 words in length.

Although our multi-document summary of the small dataset scored low in ROUGE evaluation, we believe it is the summary that is most interesting. With little pre-processing, an on-topic, readable summary was generated from 160 articles, from utilizing MMR (developed in 1998) and a near state-of-the-art neural network.

We believe our template summary is reliable because we were able to set up the template ourselves and create slots for information we thought pertinent, which would increase the score. Thus, it performed better than TextRank, which is an unsupervised method.

The use of the Wikipedia article with the multi-document performed well for three reasons. First, we probably used a similar document that the team who made our gold standard used. However, we don't consider this a bad thing. This comparison is the traditional method of evaluating summarization techniques: humans write a gold standard based on source material, and a computer generates a summary to compare with the human.

Secondly, we believe the length of the multi-document Wikipedia article summary helped it in gaining 1-grams and 2-grams to compare with the Gold Standard. Since it was 500 words in length, it was the only summary to be comparable in length to the Gold Standard.

Finally, the pointer-generator network, when choosing a word that is not in its vocabulary, opts for extraction instead of abstraction. We believe that the pointer-generator since it was not trained on Wikipedia articles, may have opted for extraction of keywords in the Wikipedia article. We will see evidence for this in the next evaluation metric.

The second summary evaluation metric performs ROUGE-1 and ROUGE-2 comparisons for sentences. In Table 7 below, we share our evaluation results.

Table 7: ROUGE sentence evaluation metrics for our 5 summaries.

Method	**Type**	**Max ROUGE-1**	**Max ROUGE-2**
TextRank with small dataset	Extractive	0.444	0.375
Template with small dataset	Extractive	0.500	0.250
PGN with Wikipedia article	Abstractive	0.620	0.200
Multi-document with small dataset	Abstractive	0.333	0.170
Multi-document with Wikipedia article	Abstractive	**0.700**	**0.556**

As shown in Table 7, our results for ROUGE sentence evaluation are quite promising. Despite having low scores in the previous assessment in Table 6, the TextRank, pointer-generator network, and multi-document small dataset summary each have good ratings for max ROUGE-1 and max ROUGE-2 sentence evaluation. Also, the template summary and multi-document Wikipedia article summary both have strong showings again in this evaluation.

We believe the Wikipedia-based summaries have such high evaluations because the same source material was used for the Gold Standard. As mentioned before, this is fairly traditional as far as assessment goes. For the multi-document Wikipedia article summary, it was interesting that the summary contained nearly identical sentences, as shown below.

[Predicted Sentence]: An estimated 300,000 structures and 500,000 vehicles were damaged or destroyed in Texas alone.

[Golden Sentence]: An estimated 300,000 structures and half a million cars were damaged or destroyed in Texas.

This result is evidence that the pointer-generator network may have fallen back on extraction in generating summaries and acted more as an extractive summarizer for certain sentences, increasing the evaluation scores of the summary.

The final method of evaluation is based on entity coverage. We share our results below in Table 8.

Table 8: Entity coverage for each summary.

Method	**Type**	**Entity coverage (%)**
TextRank with small dataset	Extractive	4.44
Template with small dataset	Extractive	3.33
PGN with Wikipedia article	Abstractive	3.33
Multi-document with small dataset	Abstractive	0.00
Multi-document with Wikipedia article	Abstractive	**10.0**

The main result that is surprising to us is the TextRank extractive summary performing better than the template summary. We specifically designed the template summary to have named entities filled in appropriate slots. Thus, it is surprising that the TextRank summary was able to have higher entity coverage. Investigating the entity coverage further, it seems the mention of “2015”, “2016”, “Southeast Texas”, and “the last three years” gave the TextRank summary additional coverage compared to the template summary that did not account for the history of storms in the area or Southeast Texas as opposed to Houston in particular.

The multi-document summary results aren’t that surprising. The multi-document small dataset summary doesn't have many entities mentioned; instead the summary talks about Hurricane Harvey in a vague sense. On the other hand, the multi-document Wikipedia article summary contains many entities and has a strong evaluation score as a result.

7.2 Evaluation and Assessment with the Wikipedia Article Lead

Another summary that we wanted to compare against is the Wikipedia Article Lead for Hurricane Harvey. The reasons for this are twofold: (a), 3 of our summaries are not dependent on the Wikipedia article, and the Wikipedia article lead could act as a gold standard for those 3 summaries; (b), we believe a comparison to Wikipedia reflects the nature of the class project well. Thus, we evaluated the TextRank summary, the template summary, and the multi-document small dataset summary against the Wikipedia article lead. The first set of evaluation metrics are shown below in Table 9.

Table 9: ROUGE evaluation scores for 3 of our 5 summaries against the Wikipedia article lead.

Method (with small dataset)	**Type**	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**	**ROUGE-SU4**
TextRank	Extractive	0.152	0.000	0.121	0.033
Template	Extractive	**0.303**	**0.094**	**0.182**	**0.099**
Multi-document with small dataset	Abstractive	0.121	0.032	0.121	0.027

In this case, the template summary has the highest evaluation scores in each of the 4 metrics. This result makes sense, since the methods for making the template summary consisted of filling the summary with as many named entities as possible. Also, though TextRank performed better than the multi-document summary, we still believe the multi-document summary is clearer and more on-topic.

We also evaluated ROUGE-n sentence scores for each of the three summaries, as shown in Table 10.

Table 10: ROUGE sentence evaluation metrics for 3 of our 5 summaries with the Wikipedia article lead.

Method	**Type**	**Max ROUGE-1**	**Max ROUGE-2**
TextRank with small dataset	Extractive	**0.357**	0.125
Template with small dataset	Extractive	0.250	0.100
Multi-document with small dataset	Abstractive	0.300	**0.133**

In this case, TextRank performed the best in ROUGE-1 evaluation and the multi-document small dataset summary performed the best in ROUGE-2.

For the final evaluation metric, we performed the assessment for entity coverage, as shown in Table 11.

Table 11: Entity coverage for 3 of 5 summaries compared to Wikipedia article lead.

Method	**Type**	**Entity coverage (%)**
TextRank with small dataset	Extractive	**8.47**
Template with small dataset	Extractive	5.08
Multi-document with small dataset	Abstractive	5.08

Surprisingly, in this case the TextRank summary performed better than the template summary just as before. The explanation is similar to before, where the TextRank includes some mention of “2015”, “2016”, “Southeast Texas”, and “the last three years”.

8. Lessons Learned

8.1 Timeline & Milestones

Sep. 12:

- Created raw JSON output file from WARC file
- Indexed the JSON file using Solr
- Extracted important words from the raw JSON file using NLTK
- Figured out POS-tagging on the raw JSON file using TextBlob and NLTK
- Prototyped an extractive summarizer using TextRank
- Created a pre-trained word embedding layer prototype using FastText for deep learning

Sep. 19:

- Re-indexed the JSON file using Solr
- Developed our first summaries using an extractive summarizer
- Researched multiprocessing on the cloud because none of us have experience with it

Oct. 1:

- Moving data cleaning onto the Hadoop cluster
- Implementing Python code for Tasks 8 and 9
- Imported pointer-generator network into ARC Cascades

Oct. 16:

- Ran jusText with Python to clean up irrelevant data
    - Eliminated about 40% of small website collection
- Generated our first templated summary with regular expressions using a previous groups work

Oct. 25:

- Draft of Gold Standard
- Submitted on Tuesday
- Template summary for tasks 8 and 9 on small dataset
    - Used spaCy to generate an excellent templated summary using the small dataset

Nov. 6:

- Abstractive summary of Hurricane Harvey Wikipedia article with pointer-generator network

8.2 Challenges

8.2.1 Pointer-Generator Network:

1. Importing into ARC

Although the pointer-generator network has a GitHub repo and some instructions for how to run it, instructions are not clear. The first challenge for making the pointer-generator network work in ARC is uploading the training dataset. SCP is usually the way to copy files from local computer to remote machine. However, ARC is secured by two factor authentication, and this makes SCP transfer very tricky.

After importing the training network and cloned pointer-generator on ARC, we had to set up Tensorflow. The pointer-generator is designed to run with Python and input arguments. However, ARC requires qsub for submitting a job. This post a challenge in converting the Python main into a qsub friendly job script.

ARC does not generate the output of the submitted job until the end of the job. This fact poses a difficulty of how the training is doing. On one occasion, we waited for the neural network for 24 hours, and it turned out it was not working. We could not determine that the neural network was outputting errors because ARC did not make this information available.

8.2.2 Cloud Computing:

1. ARC Cascade:
   The biggest challenge for using ARC's Cascades system is getting used to the job queue and configuring the virtual environment for Python. Every script is needed to be error-proofed and well packaged into a job — fortunately, instructions for creating jobs and setting up a Python environment are available. The only downside is that the Python instruction was for Python 3 and research was needed to use Python 2 with the guidance provided.

2. DLRL Hadoop Cluster
   We faced two main challenges with using the Hadoop Cluster for our purposes. The first was needing to use PySpark to run our cleaning code, and the second was the slow debugging process.

   Since we were mistaken in thinking PySpark was a necessity, we needed to find a way to clean our input of a large .json file and receive a cleaned output of a .json file. We needed the output to be in JSON format because we had already coded the majority of our functionality on the assumption that we would be working with JSON files, so rewriting our code to account for this fact was only to be the last possible option. The first method we tried was to map the Hadoop Cluster JusText tutorial provided on the course website to our dataset which we uploaded to the cluster. However, the dataset being cleaned in the tutorial was in .txt format and did not include any libraries which could read or write .json files. We tried loading the .json file as a .txt file but as expected, we got an IncompatibleTypeError.

Next, we tried following an example from Apache Spark's Programming Guides [15]. This involved importing spark.sparkContext and calling sparkContext.read.json(path) to load our data. We tried using Python's JSON libraries on this loaded object, but this was unsuccessful. We discovered that the sparkContext.read.json(path) call loads the data from the HDFS (Hadoop Distributed File System) into a DataFrame object. Upon more investigation, we learned that the spark.sql library was required to read from the DataFrame object using SQL queries. Another option on the approach of SQL queries was to use the pyspark.sql.SQLContext library to read.json files. This was with the hope of having a more natural object to work with, but similarly to the sparkContext library, the data was loaded from the HDFS into a DataFrame object. For our code to use SQL queries, we would need to refactor our code from using the Python JSON library to using these SQL queries, so we investigated further options discussed in Section 8.3.2.

An alternative to loading the JSON file into a DataFrame, which can only be accessed by SQL query, was to load the data into RDD format, which separates each JSON object into a separate string. Unfortunately our JSON file was one object, so this was not only unhelpful but the string it loaded the data into was challenging to work with and similarly to the DataFrame object, did not give us a JSON object, so we could not use our planned approach.

The second challenge we faced was more of an ease of use challenge. The issues we encountered were the slow debugging process on the Hadoop Cluster and the frequent problems that would occur while editing files. Between each edit and execution of code on the cluster, it would take several minutes before we could see the output or error messages from the script. Even then, the error messages would be surrounded by volumes of miscellaneous text and irrelevant messages.

As for editing the files on the cluster, the vim command, in particular, was frustrating to use. Around 50% of vim commands would fail to execute, freezing the console or would take 10+ minutes to load even small files. And then during editing, if there were no input for 60 seconds or so, it would freeze and force you to quit the console and relog into the cluster. We primarily used a Git Bash v2.19.2 window on a Windows 10 environment. It is possible that using a different environment to log into the Hadoop Cluster would solve this issue.

8.3 Solutions

8.3.1 Pointer Generated Network:

We believed that it was necessary to run the pointer-generator network on the cluster, but this was not the case. Instead, we were able to run the pre-trained network locally. For 500 words, it turns out the network only takes 2 seconds to perform summarization, which is quite fast. Thus, we migrated to a local machine that could run the network easily without all of the configuration issues we faced with ARC.

8.3.2 Cloud Computing:

1. ARC's Cascades system:

   We eventually decided to use the existing pre-trained neural network as it performed well and we didn't need the power of cloud computing to do deep learning. We were encountering the most issues when dealing with ARC, and decided to run things locally to avoid the issues that were occurring.

2. DLRL Hadoop Cluster

   Rather than refactor a significant portion of our code to accommodate for SQL queries, rather than using the Python JSON library, we considered other options.

   The first option we considered was porting data cleaning over to the ARC Cascades system rather than the Hadoop Cluster. Since it seemed that we could run Python code on Cascades without using PySpark, this was a promising option, but one we did not test since the second option we considered worked with minimal effort.

   The second option was to clean the data on one of our local machines. Through some manual cleaning, entry elimination, and by running JusText on the large dataset, we were able to attain the result we were looking for promptly without using the Hadoop Cluster at all. The lesson learned here was that we should always consider running the code on a local machine before attempting the overhead of getting it to work on the cloud if there is only a small difference in running time between the two.

8.4 Future Work

8.4.1 Pointer Generated Network:

There is a definite need to simplify the pipeline for the pointer-generator network. This would include a simplified data handling format, instead of the text to story to binary file conversion at the moment. In addition there would be a number of benefits from using a pipeline for churning through big datasets for analyzing multiple articles. Currently, the pointer-generator only helps in summarizing single articles.

8.4.2 Cloud Computing:

1. DLRL Hadoop Cluster:

   There are three directions future work for cleaning data on the Hadoop Cluster could go for us. The first is refactoring our code to utilize the SQL queries necessitated by the spark.sqlContext library rather than the Python JSON library. However, this would be quite a lot of effort. It may be better to start over completely to take this approach, but it would allow us to clean data on the Hadoop Cluster.

The second direction would be to write a JSON conversion script for the RDD objects which the spark.sparkContext library can use. In our opinion, this would be less work than the first option and would result in more readable code.

The third option would be to manually convert the .json file into a .txt file and attempt to plug that into the Hadoop JusText example provided to us by the instructor. This is only a possible solution, but one that is worth investigating.

Both of these directions are optional in our case as we discovered that cloud computing was unnecessary to clean a dataset as small as ours. However, were we to need to clean a much larger dataset, we should consider either of these options. A third option we'll note here would also be to find a way to receive the data in .txt format so that we can plug it directly into the Hadoop JusText example provided to us.

9. Acknowledgements

Dr. Edward A. Fox, Email: fox@vt.edu

Liuqing Li, Email: liuqing@vt.edu

Chreston Miller, Email: chmille3@vt.edu

Global Event and Trend Archive Research (GETAR) funded by the National Science Foundation (IIS-1619028 and 1619371)

10. References

[1] R. Guthrie, "DEEP LEARNING FOR NLP WITH PYTORCH." [Online]. Available: PyTorch, https://pytorch.org/tutorials/beginner/deep_learning_nlp_tutorial.html. [Accessed August 23, 2018].

[2] L. Fei-Fei, “CS231n: Convolutional Neural Networks for Visual Recognition” [Online]. Available: Stanford, http://cs231n.stanford.edu/. [Accessed August 23, 2018].

[3] A. Karpathy, “The Unreasonable Effectiveness of Recurrent Neural Networks” [Online]. Available: GitHub, http://karpathy.github.io/2015/05/21/rnn-effectiveness/. [Accessed August 24, 2018].

[4] S. Ruder, “Deep Learning for NLP Best Practices” [Online]. Available: Ruder, http://ruder.io/deep-learning-nlp-best-practices/index.html#bestpractices. [Accessed August 24, 2018].

[5] A. See, “Taming Recurrent Neural Networks for Better Summarization” [Online]. Available: Abigailsee, http://www.abigailsee.com/2017/04/16/taming-rnns-for-better-summarization.html. [Accessed August 30, 2018].

[6] “Natural Language Toolkit” [Online]. Available: NLTK, https://www.nltk.org/. [Accessed September 6, 2018].

[7] “Industrial-Strength Natural Language Processing” [Online]. Available: spacy, https://spacy.io/. [Accessed September 6, 2018].

[8] D. Jurafsky and J. Martin, “Information Extraction,” in *Speech and Language Processing*, 3rd edition: pp 327-355. [Online]. Available: Stanford, https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf. [Accessed September 27, 2018].

[9] L. Lebanoff, K. Song, F. Liu, ‘Adapting the Neural Encoder-Decoder Framework from Single to Multi-Document Summarization’, in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP),* 2018.

[10] L. Lebanoff, “multidoc_summarization” [Online]. Available: GitHub, https://github.com/ucfnlp/multidoc_summarization. [Accessed November 21, 2018].

[11] A. See, “pointer-generator” [Online]. Available: GitHub, https://github.com/abisee/pointer-generator. [Accessed August 30, 2018].

[12] R. Mihalcea and P. Tarau, “TextRank: Bringing Order into Texts” [Online]. Available: University of Michigan, https://web.eecs.umich.edu/~mihalcea/papers/mihalcea.emnlp04.pdf. [Accessed November 26, 2018].

[13] “Hurricane Harvey” [Online]. Available: Wikipedia, https://en.wikipedia.org/wiki/Hurricane_Harvey. [Accessed November 25, 2018].

[14] A. See, “cnn-dailymail” [Online]. Available: GitHub, https://github.com/abisee/cnn-dailymail. [Accessed August 30, 2018].

[15] “JSON Files,” *Apache Spark™ - Unified Analytics Engine for Big Data*. [Online]. Available: https://spark.apache.org/docs/latest/sql-data-sources-json.html. [Accessed: November 27, 2018].

[16] N. Crowder, D. Nguyen, A. Hsu, W. Mecklenburg, and J. Morris, “Computational Linguistics Hurricane Group” [Online]. Available: VTechWorks, https://vtechworks.lib.vt.edu/handle/10919/51136. [Accessed September 15th, 2018].

11. Appendices

11.1 Appendix A: Gold Standard for Hurricane Matthew

By Team 1: Jack Geissinger, Theo Long, James Jung, Jordan Parent, Robert Rizzo
11/27/2018 - CS 5984/4984: Big Data Text Summarization

Table 12: Gold Standard for Hurricane Matthew

On September 22, 2016 a mass of thunderstorms formed off Africa. On September 28, moving away from the Antilles (where it caused extensive damage to landmasses), with sustained winds of 60 mph, the storm gained tropical cyclone status when southeast of St. Lucia. It became Hurricane Matthew on September 29, when north of Venezuela and Columbia. On September 30, Hurricane Matthew became a significant hurricane, reaching Category 5, with wind speeds of 165 miles per hour, before moving to the Caribbean. It was the first Category 5 Atlantic hurricane since Felix in 2007, the deadliest since Stan in 2005, and the costliest since Sandy in 2012. Preparations, cancellations, warnings, and watches proceeded in the Windward Islands, Columbia (where one person drowned in a swollen river), and Jamaica.

On October 4, Hurricane Matthew made landfall in Haiti's Tiburon Peninsula with peak winds up to 145 mph as a Category 4 hurricane, and rainfall reaching over 30 inches. Hurricane Matthew caused a humanitarian crisis in Haiti, affecting over 2 million people, many needing clean water, food, shelter, sanitation, household goods, and medical supplies; more than 800 people died as a result of the storm. Dozens of cholera treatment centers were destroyed which led to a fear that the cholera epidemic would worsen. The World Health Organization began a vaccination campaign against cholera to avoid further spreading of the disease following Hurricane Matthew's destruction. The disturbance caused by Hurricane Matthew caused the Haitian presidential election to be postponed. There were 546 fatalities in Haiti and damage costing $1.9 billion; this was Haiti's worst disaster since the 2010 earthquake. Experts say that it will take at least 5 years to restore plantations of export crops like coffee, cocoa, and fruits. Four were killed in the Dominican Republic, which had heavy rainfall.

On October 5, Hurricane Matthew hit eastern Cuba, resulting in 4 fatalities and $2.58 billion in losses, especially in the Guantánamo Province. The city of Baracoa suffered the most, with 25-foot waves crashing over the coast and many buildings collapsing. The Red Cross was ordered to rescue those trapped in collapsed buildings. On October 5 and 6, Matthew then moved through the northern islands of the Bahamas, directly hitting Grand Bahama. Peak-winds were 145 miles per hour, and peak rainfall was 20 inches. As Hurricane Matthew traveled through the Bahamas, the hurricane was classified as Category 3. There were no fatalities in the Bahamas.

In response to the destruction Hurricane Matthew had caused, Florida Governor Rick Scott ordered the evacuation of 1.5 million people living in potential danger zones. President Obama also declared a state of emergency in both Florida and South Carolina, calling in the Department of Homeland Security to prepare relief measures. On October 6 and 7, Matthew moved along the entire length of the Florida Atlantic coastline, first as Category 3. It was downgraded to Category 2 late on October 7, then to Category 1 on October 8. The storm killed 12 people in Florida and 3 in Georgia. Over 1 million lost power in Florida, and more than half a million in Georgia and South Carolina. It made landfall on October 8 at Cape Romain National Wildlife Refuge near McClellanville, South Carolina, with 85 mph winds. 30 people were killed in North and South Carolina and 2 in Virginia; these states suffered flooding due to torrential rainfall, with portions of Interstate 95 shut down. Robeson County schools did not reopen till 31 October. The peak rainfall for Hurricane Matthew in the United States was about 20 inches in North Carolina. On October 9, turning away from Cape Hatteras, North Carolina, Matthew reentered the Atlantic as a post-tropical storm and moved along the eastern coastline towards Canada. The total damage to the United States is estimated at more than $10 billion, with more than 40 people dead.

On October 10, Hurricane Matthew's remnants hit Canada, specifically Newfoundland and Cape Breton, causing flooding, strong winds (over 56 mph), and power outages. There were no fatalities, but the total damage to Canada is estimated at $7.6 million. Most of this damage affected roads in Newfoundland. In Canada, Hurricane Matthew was absorbed by a cold front, and completed its 3,000-mile-long journey of destruction. Recovery and reconstruction continued into 2017 and 2018.

11.2 Appendix B: References for Gold Standard for Hurricane Matthew

We used the method outlined in the TAC 2011 Guided Summarization provided for us via the class Canvas page to create our gold standard for Team 3 [1]. Following this method, we constructed concise summaries from a series of news articles that we manually pulled from the internet covering the topic of Hurricane Matthew. We then merged these summaries into one comprehensive gold standard. We decided to construct our gold summary in the order of the hurricane's impact. Initially, we read over the Hurricane Matthew Wikipedia page and made bullet points of the major events and topics. Through this process, we found that ordering our summary by the regions affected by Hurricane Matthew would be the most systematic way to organize our outline. We then used Google to retrieve news articles using “Hurricane Matthew” and the regions affected as keywords. From these articles, we gathered metadata about each major region affected and constructed summaries from this data. Below is a list of the articles used to create our gold standard.

[1] “TAC 2011 Guided Summarization Task Guidelines,” TAC 2011 Guided Summarization Task. [Online]. Available: https://tac.nist.gov//2011/Summarization/Guided-Summ.2011.guidelines.html. [Accessed: 02-Dec-2018].

[2] “Hurricane Matthew,” Wikipedia, 27-Nov-2018. [Online]. Available: https://en.wikipedia.org/wiki/Hurricane_Matthew. [Accessed: 02-Dec-2018].

[3] “Deadly Hurricane Matthew pounds the Caribbean,” CBS News, 03-Oct-2016. [Online]. Available: https://www.cbsnews.com/news/deadly-hurricane-matthew-pounds-the-caribbean/. [Accessed: 02-Dec-2018].

[4] J. Elliott, “Cleanup a struggle after hurricane remnants batter Canada's East Coast,” CTVNews, 12-Oct-2016. [Online]. Available: https://www.ctvnews.ca/canada/cleanup-a-struggle-after-hurricane-remnants-batter-canada-s-east-coast-1.3111612. [Accessed: 02-Dec-2018].

[5] M. Whitefield, “Eastern Cuba lashed by Matthew, crumbling homes and more,” Bradenton Herald. [Online]. Available: https://www.bradenton.com/news/weather/hurricane/severe-weather-blog/article106107817.html. [Accessed: 02-Dec-2018].

[6] P. Wright and R. D. Benaim, “Crowdfunding Campaigns Underway To Aid Cuba As Country Grapples With Hurricane Matthew Destruction,” The Weather Channel. [Online]. Available: https://weather.com/news/news/hurricane-matthew-cuba. [Accessed: 02-Dec-2018].

[7] A. Holpuch, “Haiti faces fresh cholera outbreak after Hurricane Matthew, aid agencies fear,” The Guardian, 14-Oct-2016. [Online]. Available: https://www.theguardian.com/world/2016/oct/14/haiti-cholera-hurricane-matthew-aid-agencies. [Accessed: 02-Dec-2018].

[8] “Hurricane Matthew: Haiti storm disaster kills hundreds,” BBC News, 07-Oct-2016. [Online]. Available: https://www.bbc.com/news/world-latin-america-37582009. [Accessed: 02-Dec-2018].

[9] M. Blau, H. Yan, and P. Oppmann, “Hurricane Matthew kills 276, tears through Haiti, Bahamas, Cuba,” CNN, 07-Oct-2016. [Online]. Available: https://www.cnn.com/2016/10/06/americas/hurricane-matthew-cuba-haiti/index.html. [Accessed: 02-Dec-2018].

[10] A. Sarkissian, J. D. Gallop, and D. Stanglin, “Hurricane Matthew: Florida governor says 'Evacuate, evacuate, evacuate',” USA Today, 07-Oct-2016. [Online]. Available: https://www.usatoday.com/story/news/nation/2016/10/06/hurricane-matthew-batters-bahamas-set-strengthen-florida-approach/91652096/. [Accessed: 02-Dec-2018].

[11] D. Stanglin and J. Dean, “Matthew makes landfall in SC, dumps rain on eastern Carolinas,” WCNC, 08-Oct-2016. [Online]. Available: https://www.wcnc.com/article/weather/matthew-makes-landfall-in-sc-dumps-rain-on-eastern-carolinas/275-330788246. [Accessed: 02-Dec-2018].

[12] R. W. Miller, “A state-by-state look at Hurricane Matthew damage,” USA Today, 09-Oct-2016. [Online]. Available: https://www.usatoday.com/story/weather/2016/10/09/how-hurricane-matthew-affected-each-state-hit/91823380/. [Accessed: 02-Dec-2018].