Big Data Text Summarization CS4984/5984

Team 12 Project Report: Automatic Summarization of News Articles about Hurricane Florence

Authors

Frank Wanye
Matt Tuckman
Joy Zhang
Fangzheng Zhang [1]
Samit Ganguli [2]

Instructor: Dr. Edward Fox

Department of Computer Science
Virginia Tech
Blacksburg, VA 24061

December 14, 2018

[1]Fangzheng dropped the class on September 25, 2018
[2]Samit joined our team on October 2, 2018

Table of Contents

List of Figures

List of Tables

1 Executive Summary

We present our approach for generating automatic summaries from a collection of news articles acquired from the World Wide Web relating to Hurricane Florence. Our approach consists of 10 distinct steps, at the end of which we produce three separate summaries using three distinct methods:

1. A template summary, in which we extract information from the web page collection to fill in blanks in a template.
2. An extractive summary, in which we extract the most important sentences from the web pages in the collection.
3. An abstractive summary, in which we use deep learning techniques to rephrase the contents of the web pages in the collection.

The first six steps of our approach involve extracting important words, synsets, words constrained by part of speech, a set of discriminating features, important named entities, and important topics from the collection. This information is then used by the algorithms that generate the automatic summaries.

To produce the template summary, we employed a modified version of the hurricane summary template provided to us by the instructor. For each blank space in the modified template, we used regular expression matching with selected keywords to filter out relevant sentences from the collection, and then a combination of regex matching and entity tagging to select the relevant information for filling in the blanks. Most values also required unit conversion to capture all values from the articles, not just values of a specific unit. Numerical analysis was then performed on these values to either get the mode or the mean from the set, and for some values such as rainfall the standard deviation was then used to estimate the maximum.

To produce the extractive summary, we employed existing extractive summarization libraries. In order to synthesize information from multiple articles, we use an iterative approach, concatenating generated summaries, and summarizing the concatenated summaries.

To produce the abstractive summary, we employed existing deep learning summarization techniques. In particular, we used a pre-trained Pointer-Generator neural network model. Similarly to the extractive summary, we cluster the web pages in the collection by topic, before running them through the neural network model, to reduce the amount of repeated information produced.

Out of the three summaries that we generated, the template summary is the best overall due to its coherence. The abstractive and extractive summaries both provide a fair amount of information, but are severely lacking in organization and readability. Additionally, they provide specific details that are irrelevant to the hurricane. All three of the summaries could be improved with further data cleaning, and the template summary could be easily extended to include more information about the event so that it would be more complete.

2 Introduction

This project is a part of the Big Data: Text Summarization class as taught at Virginia Tech in the Fall of 2018. In it, we are tasked with developing an automatic summary of a collection of news articles obtained from the web, relating to Hurricane Florence.

Our dataset is provided to us in a WARC file, which contains 10948 records. Each record is made up of 5 fields: the URL from which the article was obtained, the text obtained from the URL, the timestamp at which the article was retrieved, a unique identifier, and a version number.

We choose to approach this problem with a ten-step method intended to gradually introduce us to the natural language concepts needed to generate automatic summaries. The following are the steps we follow:

1. Generate a set of most frequent important words
2. Generate a set of synsets that cover the important words
3. Generate a set of words constrained by part of speech
4. Generate a set of discriminating features
5. Generate a set of frequent and important named entities
6. Generate a set of important topics
7. Generate an extractive summary, as a set of important sentences
8. Generate a set of values for each slot in a template, matching collection semantics
9. Generate a readable template summary
10. Generate a readable abstractive summary

3 Literature Review

Automatic summarization techniques can be grouped into single and multi-document summarization, based on the number of documents that the technique is meant to summarize. They can also be grouped into generic, domain-specific, and query-based techniques, based on whether they are meant to consider the topics of the texts they summarize, or whether the summary is supposed to change based on some user-defined query. They can also be grouped into abstractive and extractive techniques, based on whether they simply extract information from the texts, or rewrite the texts in a coherent manner.[1]

Abstractive summarization, where new words and sentences that may not have been present in the original document or collection of documents are generated in the summary, is a difficult task, because it requires an understanding of the contents of the documents. As such, more research has historically been done in extractive summarization techniques than in abstractive ones. However, with the recent success of sequence-to-sequence techniques, which are based on recurrent neural networks, there has been an increase in the amount of attention paid to automatic abstractive summarization techniques. [2]

Extractive summarization techniques generally attempt to select important sentences from the text to be summarized. There are three main approaches to selecting these sentences: frequency based approaches, feature based approaches, and machine-learning based approaches. Frequency based approaches rank sentences using techniques like word probabilities and term frequency - inverse document frequency (TF-IDF). Feature based approaches rank sentences based on their features, like their similarity to the document title, their similarity to the other sentences, their position in the text, the presence of proper nouns, etc. Machine-learning based approaches attempt to classify sentences into summary sentences and redundant sentences, given a training set of documents and their summary sentences. [1], [3]

In the domain of news summarization, a combination of template summaries and information extraction have been successfully applied to generate news article summaries. Sentence extraction incorporating both ML and Statistics have also been used in systems like MultiGen. Techniques based on ontologies, such as the Ontology-enriched Multi-Documentation Summarization (OMS) system, have also been successfully employed. Here, the sentences in the document collection are mapped to a domain-specific ontology. When a user issues a query, the ontology nodes that relate to the query are selected, and the sentences associated with those nodes are combined to form summaries. [1]

When performing multi-document summarization, there is an increased concern that information will be repeated in the summary. Three major types of methods have been used to circumvent this: cluster based, graph based, and discourse based methods. Cluster based methods attempt to cluster similar sentences together, and use the cluster centers to form summaries. Graph based approaches build graphs linking sentences together that have a similarity above a predetermined threshold. Important sentences are then selected based on the graph produced. Discourse based methods make use of semantic relationships between sentences. However, due to the difficulty of getting computers to identify these relationships, they have to be identified by humans. [1]

Two main methods exist for evaluating automatically generated summaries. The first, and simplest, is human evaluation. The second is called Recall-Oriented Understudy for Gisting Evaluation (ROUGE), and has various formats, which include ROUGE-n, ROUGE-L, and ROUGE-SU. ROUGE-n compares word n-grams between the summary and original text, ROUGE-L makes use of the concept of longest common subsequence, and ROUGE-SU compares word bigrams between the summary and original text, but allows the insertion of words between the words in the bigrams. [4]

4 Design and Implementation

4.1 A set of most frequent important words

To extract the most frequent important words, we employed a technique called Term Frequency - Inverse Document Frequency (TF-IDF). TF-IDF works by weighting the frequency of a word in a given document with the inverse document frequency of the word in the collection to produce an importance score [5].

Our implementation of TF-IDF has two key differences. The first is that, being concerned with the importance of the words in the entire collection as opposed to a single document, we calculate term frequency of words with respect to the entire collection. Our term-frequency tf score for each word w was thus calculated with the following equation:

$$tf_w = \frac{\text{number of occurrences of } w \text{ in collection}}{\text{total number of words in collection}}.$$

We initially implemented inverse document frequency idf for each word w using the traditional method, with the following equation:

$$idf_w = \frac{\text{number of documents in collection}}{\text{number of documents containing word } w}.$$

The TF-IDF score for each word $tfidf_w$ was then calculated as the product of tf_w and idf_w. However, due to the large number of words present in a collection as opposed to a single document, the resulting tf_w scores were typically orders of magnitude lower than their corresponding idf_w scores. As such, idf_w contributed disproportionately to $tfidf_w$, leading to words like fullscreen, which appeared in only a few documents, having the highest $tfidf_w$ scores.

This prompted the introduction of a second difference to our TF-IDF implementation. Since words that appear in an insignificant portion of a collections documents are likely to not be important to the collection as a whole, we filtered out words that appeared in less than 10 % of the documents in our collection.

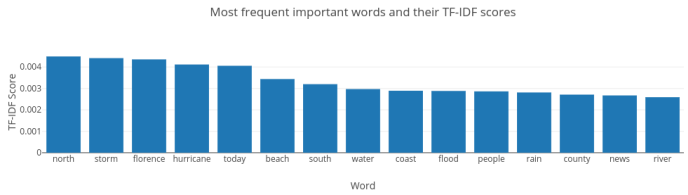

Figure 1: Most frequent important words and their TF-IDF scores

Figure 1 shows the TF-IDF scores of the 15 most important frequent words resulting from this method. The 200 most important frequent words are listed in Appendix C

4.2 A set of synsets that cover the words

For clarity, a synset in this situation refers to a set of synonyms related to our important words.

After forming our list of the most frequent and important words throughout the articles, we were then tasked with creating a list of synonyms to go along with this set. This topic was fairly open ended so we started with one of the most basic ideas. Our first thought was to go through the entire list of frequent and important words that we already have, and simply extend on this list so that each of these words is also associated with a list of similar words. We accomplished this by using the WordNet library from NLTK. Specifically, we used their synsets method to find the list of synonyms. This worked fine, however did not tell us any more about the documents than the information that we already had. Yes we now have a list of words with the same meaning as our important words, but who knows if these synonyms have anything to do with the articles themselves.

To combat this problem, we decided to first generate all of the synonyms as before, however we then filtered the list of synonyms to only include words that were in the list of frequent words itself. This improved on the previous method by making it so that only **relevant** synonyms to our important words would appear. We were much more happy with this result as we got rid of extra irrelevant words, however it seemed unfair to only count the number of times that the important words appeared in the articles, when their newly formed synsets should have other words that are relevant and mean the same thing.

Extending on this idea we decided to take this new list and perform a frequency analysis to group together words that appeared in each other's synsets, summing their frequencies. This method would make it so that the most frequent synsets that we see in the output are actually the most frequent, instead of just being the synsets associated with the most frequent important words. This method led us to very good results that well represent the ideas in the articles. After this last change, the only other modifications that we did to our code was to append to the original synonyms list by adding the hypernyms of each synonym. This was again done using the same NLTK library. By generating a more extensive initial list for each word, we have helped extend the result to more related words.

Table 1 shows the 10 most frequent synsets from our dataset, while Appendix E shows the 50 most frequent synsets obtained with this method, along with the word from which the synsets were constructed. Looking through this list, all of the words and their synonyms are highly related to our articles. One disadvantage of our strategy, however, is that proper nouns and specific words such as florence, carolinas, or even hurricane do not have many (or any) synonyms or hypernyms when run through the NLTK library. This makes them appear by themselves in the result. This was however expected and despite it the synsets are very accurate.

Frequency	Words in synset
16789	area, building, point, content, place, move, center
16650	point, condition, area, move, station, order, estimate, spend, place run
16423	damage, leave, direct, move, travel, continue, occur, pass, free, run, change, break
16221	supply, state, direct, produce, pass, release, move, provide, give, change, esti-mate, show, stretch, lead, offer
14321	thing, location, line, move, leave, change, part
13614	leave, location, flight, travel, advance, change, move, pass
13254	weather, travel, move, wind
13138	hit, travel, move, change, show, fly
13019	road, travel, move, work, drive, hit, power
12749	region, people, arrive, change, bring, land

Table 1: Most frequently occurring synsets

4.3 A set of words constrained by part of speech

Part of speech describes a word's role in a sentence. The parts of speech that were prioritized when we created our set of words were nouns (N) and verbs (V). Using the POS tags in the Penn

Treebank project, we grouped the sub-classes of nouns–noun (NN); nouns, plural (NNS); noun, proper (NNP); nouns, proper, plural (NNPS)– and the sub-classes of verbs–verb, base form (VB); verb, past tense (VBD); verb, gerund or present participle (VBG); verb, past participle (VBN); verb, non 3rd person singular present (VBP); verb, 3rd person singular present (VBZ)– into two separate sets. When parsing through our dataset, any word associated with a POS that is in either of these two sets would be thrown into a new set to be lemmatized later on.

```
nouns = Set([‘‘NN, ‘‘NNS, ‘‘NNP, ‘‘NNPS])
verbs = Set([‘‘VB, ‘‘VBD, ‘‘VBG, ‘‘VBN, ‘‘VBP, ‘‘VBZ])
```

Generating noun phrase chunks

Many words in a sentence also tend to be grouped together as a phrase. For instance, “North Carolina” can be split into simple tokens that may not have much meaning by themselves as “North” and “Carolina”. As a result, it is a good idea to chunk, or extract phrases from the text, on top of POS tagging. Thus, “North Carolina” would be chunked together as a single word. NLTK uses regular expressions to generate chunks by checking grammar using POS tags. We searched for noun phrase (NP) chunks by searching for determiners (DT) followed by adjectives (JJ) and then a noun from our noun set. NPs would be added to our set of nouns for generation of important nouns later.

A snippet showing how chunking works is shown below:

```
record = ‘‘‘A state of emergency was declared as the hurricanes approaches
            the east coast’’’
regex = (‘‘‘NP: {<DT>?<JJ>*<NN>}’’’)
chunking = nltk.RegexParser(regex)
post = nltk.pos_tag(nltk.word_tokenize(record))
t = chunking.parse(post)
for sub in t.subtrees():
    print(sub)
```

Looking at the example record above, “the east coast” would have been chunked together into one NP as “the” is a DT , “east” is a JJ that precedes “coast”, a NN. The output is shown to be:

```
(S
  (NP A/DT state/NN)
  of/IN
  (NP emergency/NN)
  was/VBD
  declared/VBN
  as/IN
  the/DT
  hurricanes/NNS
  approaches/VBZ
  (NP the/DT east/NN)
  (NP coast/NN))
(NP A/DT state/NN)
(NP emergency/NN)
```

```
(NP the/DT east/NN)
(NP coast/NN)
```

Here multiple NPs have been determined: "A state", "emergency", "the east coast". The results are not perfect, though being able to group together words that may not be extremely relevant by themselves is important for determining relevant nouns and verbs later on.

Generating a set of important nouns and verbs

We used NLTK to tokenize and part of speech (POS) tag our data set. Then we filtered out stopwords and chunked together NPs, creating a filtered, POS tagged, and NP chunked list. Taking this new list, we parsed through every item looking for words that had a corresponding POS in either the noun set or verb set. After producing two separate lists of all the nouns and verbs found in our dataset, we lemmatized each list and used tfidf to determine the most important nouns and verbs.

4.4 A set of discriminating features

Generating a feature list

To find a set of discriminating features, we adopted a modified bag-of-words approach. The main difference between our approach and the bag-of-words approach is that we encode only important words, bigrams, and synsets, as opposed to just the words present in the dataset.

In order to achieve this, we first need a set of feature labels, for which we will search in the web pages in our collection. We encode these feature labels as a list of strings (representing words and bigrams) and sub-lists of strings (representing synsets). We first extract important words, frequently occurring bigrams, and synsets from our collection. We then merged these three lists into a single feature label list, with the following rules:

1. If both words in a bigram are in the important word list, add the bigram to the feature label list
2. Remove words that are already a part of the feature label list as one of the words in a bigram from the important word list
3. Add the remaining important words to the feature label list
4. For each important word in the feature label list that is not part of a bigram, if it has a synset associated with it, replace the word with its synset.

Appendix D lists the feature label list obtained with this method.

We then create two feature sets: one that encodes the presence of each feature label in each record ($F_{presence}$), and one that encodes the number of occurrences of each feature label in each record (F_{count}).

Labeling a subset of records

We adopt a supervised approach to classifying the records in our dataset. As such, we first label a subset of our records as either relevant to our topic, or not.

To label a subset of our records, we employed some simple statistical heuristics. For each record, we calculated $S = \sum F_{presence}$, and examined the values of S. Figure 2 shows the distribution of S we obtained. Because the distribution is skewed right, we derived two thresholds from this distribution:

- Relevance threshold: The relevance threshold $T_r = \overline{S} + 1.5 \cdot \sigma^2(S)$. Any record where $S > T_r$ is labeled as relevant.
- Irrelevance threshold: The irrelevance threshold $T_i = 3$. Any record where $S < T_i$ is labeled as irrelevant.
- Any record where $T_i \leq S \leq T_r$ is left unlabeled.

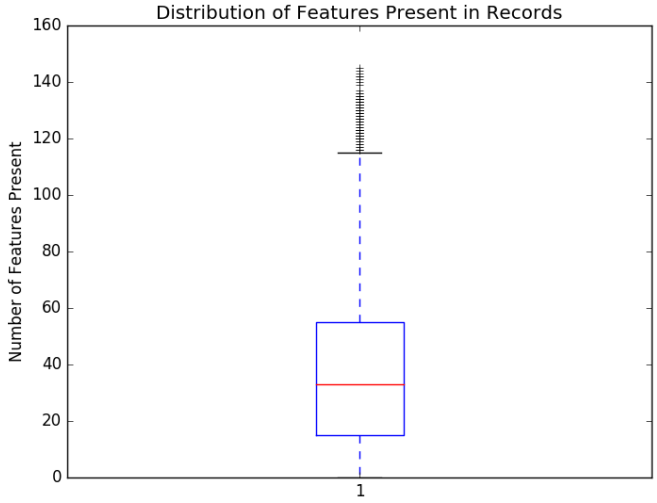

Figure 2: Distribution of $\sum F_{presence}$

These rules resulted in 777 records being labeled as relevant, and 472 records being labeled as irrelevant, out of the total of 8918 records longer than 100 characters.

Using a classifier to label the remaining records

To classify all the records as either relevant or irrelevant, we then built a multi-layer perceptron classifier with 3 hidden layers of size 100, 50 and 10 (for a total of 5 layers including input and output), using the PySpark machine learning (ML) module.

The automatically labeled training data was split into a training and test set, with a $7:3$ ratio. The classifier was then trained on the training set using the default learning rate of 1.0×10^{-6} and maximum epochs of 100. The resulting trained model achieved 100% accuracy on the test set,

which suggests that the model overfit on the training data. However, upon examining samples of the records classified as relevant and irrelevant, we decided that the results were good enough to use.

This model classified 7524 records as being relevant, and 1394 records as irrelevant.

Identifying relevant sentences

Further examination of the relevant records revealed that many of them had irrelevant text making up a small-to-significant chunk of their sentences. To get rid of this irrelevant text, we split each relevant record into its sentences, and repeat the above two steps to classify sentences as being either relevant or irrelevant. Figure 3 shows the distribution of features present in the sentences of the relevant records.

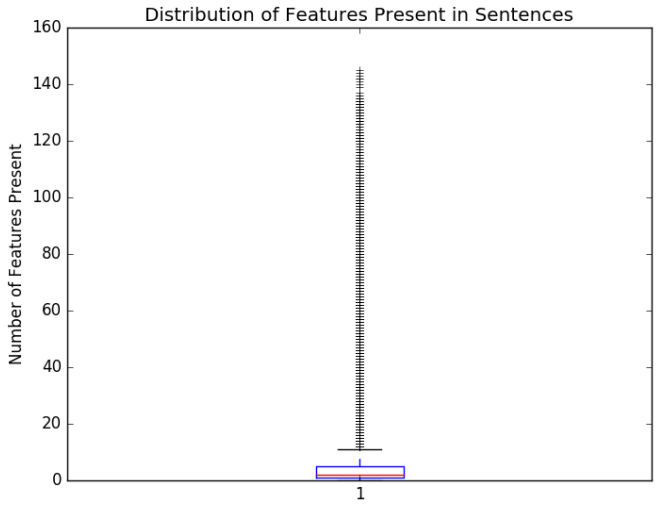

Figure 3: Distribution of $\sum F_{presence}$

Because of the high degree of skew in the distribution of features present in the sentences of the relevant records, we used four sets of thresholds for labeling this data before classification, the results of which are shown in Table 2.

T_r	T_i	labeled relevant	labeled irrelevant	classified relevant	classified irrelevant
$\overline{S} + 2.0 \cdot \sigma^2(S)$	2.0	7037	108276	70084	179128
$\overline{S} + 1.5 \cdot \sigma^2(S)$	1.0	13643	63765	95155	154057
$\overline{S} + 1.0 \cdot \sigma^2(S)$	1.5	16162	108276	102808	146404
$\overline{S} + 1.0 \cdot \sigma^2(S)$	1.0	16162	63765	148732	100480

Table 2: Results of sentence classification with different thresholds

The relevant sentences are saved to a file, separately, as well as recombined to reconstruct the original records, but without the irrelevant sentences. Reconstructed records consisting of less than 5 sentences were ignored, because they are less likely to contain useful information.

Selecting discriminating features

To select a list of discriminating features, we perform the chi-squared independence test on the $F_presence$ feature set, using the results of sentence classification as the category label. This involves computing a statistic to determine the level of dependence between the category label and each feature in the feature set, and selecting the top n features with the highest dependence statistic. We utilize PySpark's chi-squared feature selector model to do this, and obtain the following 50 discriminating features:

1. Hurricane Florence
2. North Carolina
3. storm surge
4. make landfall
5. Myrtle Beach
6. East Coast
7. million people
8. Wilmington N.C.
9. United States
10. National Weather
11. Monday Sept.
12. Rights Reserved
13. mandatory evacuation
14. heavy rain
15. major hurricane
16. Friday morning
17. Thursday night
18. Carolinas Virginia
19. state emergency
20. rise water
21. order evacuate
22. All reserve
23. early week
24. east coast
25. forecast track
26. Tuesday September
27. A link
28. The New
29. President Trump
30. comment
31. work, force, Service
32. show, lead
33. change, turn, affect, live, move
34. weekend
35. change, make, bring
36. Sunday
37. experience, change, reach, find
38. land, reach, hit, work, make, ground
39. change, impact
40. Emergency
41. late
42. information, News
43. local
44. affect, move, approach, hit
45. government
46. content, food
47. big
48. check, watch
49. fall, change
50. change, affect

Figure 4 summarizes the above steps for selecting discriminating features and cleaning the dataset.

4.5 A set of frequent and important named entities

To start extracting important named entities, we first needed to decide on how we wanted to determine if a particular word or phrase is a named entity or not. To do this we looked at and tested the libraries of both NLTK and spaCy and found both to be about equally as good at determining this. SpaCy provides more detailed tagging of these entities but takes longer to run

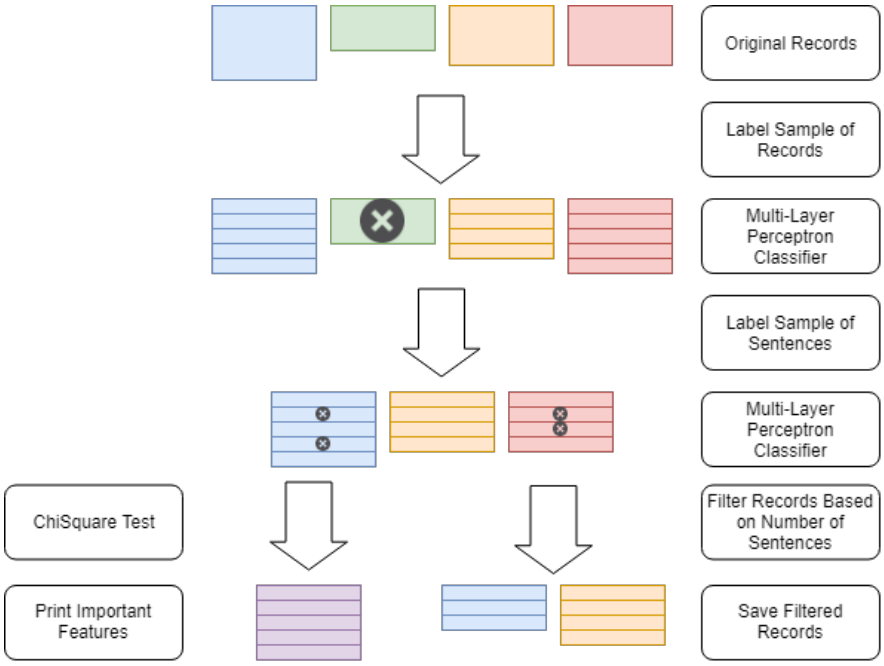

Figure 4: Pipeline for extracting discriminating features and cleaning the dataset

while NLTK was much more light weight, but only has a few different types of entities it can recognize (and the accuracy of this is not always the best). We decided to go with NLTK because of the quick run speed, and because for this topic it is not important to us what type of thing a named entity is (organization, geopolitical entity, etc), we just want to recognize it as one.

Under the hood, NLTK is using a chunking algorithm to check if words and phrases match a particular part of speech sequence, and the results turned out to be very good. After chunking all of the named entities, we then filtered the sentence lists to only include these named entities, and did a frequency count of them. The 20 most frequent named entities we obtained with this method from a portion of the dataset are shown in Appendix F.

Looking at the results in the appendix it is easy to see that all 20 words are extremely relevant to our event, and would be the named entities that one thinks of immediately when Hurricane Florence is brought up. We decided that since this strategy was simple yet extremely effective, there was no need to try to attempt to add logic to deal with determining the importance of these words as the frequency was working perfectly for the relevant article filtering that we will discuss later.

4.6 A set of important topics

LDA is a type of statistical modelling that can be used for discovering topics that occur in a given set of documents. This means that it can be used for classification of texts in documents by their topic. LDA works by building a topic per document model and words per topic model using Dirichlet distributions.

To implement topic identificiation using LDA, we use the Gensim Python library. First, we split each record into a list of words; this allows us to create a bag of words from each document. To do so, we first create a dictionary mapping words to integer IDs from the split records using the

corpora submodule of Gensim. We then call the doc2bow function from the resulting dictionary object to create a bag of words for each record in the dataset.

The last step of the process is to train the LDA model, using the bags of words generated previously as the training set. LDA assigns a relative weight to each topic based on the number of topics we select and the words occurring in that topic. Upon experimenting with the parameters of the model, we find that we obtain the best results when we try to identify 7 topics, with 5 words describing each topic. The descriptions of the topics identified are shown in Table 3.

Topic number	Words describing the topic				
1	information	news	emergency	disaster	power
2	high	share	pollen	kit	area
3	storm	hurricane	news	florence	storm
4	news	florence	de	hurricane	storm
5	florence	hurricane	north	storm	coast
6	hurricane	storm	florence	emergency	north
7	share	post	link	hurricane	florence

Table 3: Topics identified by LDA

As can be seen in Table 3, the descriptions for the topics identified by LDA are distinct, and generally make sense, with the exception of the 2nd and perhaps the 4th topics.

4.7 An extractive summary, as a set of important sentences

Unsupervised model using PyTeaser

An extractive summary was created by including the provided URLs from the dataset as preprocessing initially ignored URLs. PyTeaser is a Python library based off of Jolo Balbins (MojoJolo on GitHub) TextTeaser API written in Scala. PyTeaser looks at the headline of an article and extracts sentences that appear relevant by scoring sentence length, position, and keyword frequency. Using the URLs from the dataset, the summarizer function utilizing PyTeaser was ran on each web page to summarize. Summaries were then concatenated in batches of twenty and the summarize function was once run again with relevant topics such as "Hurricane Florence". We did this because when we concatenate the summarized articles together, there is no longer a single headline that describes all the content in the articles. This approach was repeated until we reached a final list of concatenated summaries that was used to generate the final extractive summary. The final generated extractive summary was then manually cleaned with the goal of removing sentence fragments and irrelevant information.

An issue we noticed after running the PyTeaser model is that since the model looks directly at the webpages that the their URLs point to, all preprocessing and filtering was negated. This resulted in the inclusion of extracted sentences that our summarize function deemed relevant to the webpage, but not for our overall extractive summary. Normally, this type of information would have been filtered out during preprocessing but the PyTeaser model did not take account of extraneous information on webpages. As a result we had to remove several sentences that were related to individual crises and reactions towards Hurricane Florence. An example of such sentences we removed were:

‘‘Lindsay and Abe seal the deal as Hurricane Florence approaches. Lindsay Ryan and

her fiance Abe Mosher live in California but headed to Charleston to bring their wedding to Lindsays 93-year-old grandparents.’’

‘‘Heres the latest position and forecast of Florence from the National Hurricane Center: APP USERS: Tap here to see the forecast track Remnants of Florence arrive here early next week...Metro area timing: view more here APP USERS: Tap here to view the hurricane tracker The letter must have had some effect because city officials did amend the original order somewhat. ’’

Another issue we noticed with the PyTeaser model is that the summarize function occasionally focused too heavily on the keywords “Hurricane Florence” and would extract any sentence that contained those keywords. While this is not a pressing issue, it leads to some redundant information. Out of the seventeen sentences in the final extractive summary, fifteen of them contained the keywords “Hurricane Florence” or “Hurricane”. Nonetheless, this may be viewed as a success since the model picked sentences that were relevant to the topic, Hurricane Florence. A more diverse set of keywords could have varied extracted sentences and limited redundant information. However, a trade-off of using more keywords could have lead to an increased number of irrelevant sentences. The inclusion of preprocessing and filtering with a larger set of keywords may improve extractive summary results.

The final summary generated using this model can be found in Appendix H.

Model using the Summa library

As an alternative to the PyTeaser model, we explored the Summa Python library, which is based on the TextRank algorithm [6], with some optimizations to the similarity function used in the algorithm [7]. The summarizer in the Summa algorithm does not require a URL or a headline in order to summarize an article, and is much faster than PyTeaser. However, the summaries generated by it are very similar to those generated by PyTeaser in terms of content, and do not produce better ROUGE scores, so we abandoned this approach.

Model using lexical chains

A model using lexical chains was also attempted following Dr. Hollingsworth’s thesis [8], research paper [9], and patent [10] on automatic text skimming. Dr. Hollingsworth released a web application using his method called *Skimcast*. Dr. Hollingsworth’s personal blog [11] wherein he discusses text skimming and the creation of Skimcast was also used as a reference for this model.

A lexical chain is a string of related words in text that can be useful in text summarization. Since lexical chains are independent of the grammatical structure of the text, it is easier to choose words that represent fragments of the concept. For instance, given the following sentence:

```
Wind speeds increase as Hurricane Florence grows in size and strength.
```

Lexical chaining may identify the following chain:

```
[wind, increase, hurricane]
```

From this chain, one can piece together that the sentence has to do with wind during a hurricane.

Lexical chains are interesting in that they aid in detecting shifts in topic in longer texts or multiple texts. Thus it becomes helpful in figuring out what sentences should be extracted in an extractive summary if subject matter changes.

The model attempted utilized lexical chains and adjectives to generate an extractive summary. A key part of Dr. Hollingsworth's method utilized adjectives in keyword detection and lexical chaining. Relevant adjectives were detected by both adjectives' occurrence patterns in the text and the adjective's grammatical properties. Skimcast's success seemed to be due to relevant adjective inclusion in lexical chains, rather than TF-IDF, cosine similarity, or sentence position.

We tested out Dr. Hollingsworth's method by randomly choosing a hundred records from our filtered dataset and concatenating them into one article for Skimcast to summarize. We noticed that Skimcast output seemed to read more cohesively and be similar to a story; it included more individual names and the effects of Hurricane Florence. Skimcast's generated extractive summary did have less redundant information as well.

Due to time constraints and difficulty involved in emulating Dr. Hollingsworth's algorithm, the model using lexical chains was never completed. A model that utilizes both lexical chains and more traditional models of sentence extraction seems interesting, and potentially better than either model individually.

4.8 A set of values for each slot in a template matching collection semantics

Getting Started

To start with this section we were given an outline of many questions that we could use to determine what information we should try to extract from the data set. This outline can be found on the course homepage under the Metadata Hurricane link. We used this as our starting point, however we modified some of the questions, adding and removing to fit our needs. After this was done, we next moved to figuring out how to extract the information that would answer these questions.

Answering the Simplest Questions

Our initial idea was to use exclusively regular expressions to find sentences in the articles that matched specific phrases that we know would answer the questions. As an example to extract the name of the hurricane, it is fairly simple to find every match to the regular expression [Hh]urricane ([A-Z][a-z]+) and then find the most common result from this query. This turned out to be a very effective solution for a portion of the questions in the data set, however occasionally we would either get too few matches even in the big data set, or the information could not be given in a format that is easily regex matched. Generally, these problems went hand in hand where we would not be able to generate an all encompassing regular expression, which would therefore result in it producing a minimal amount of output. When we did get a success with this method however, they were usually very good.

To solve this we first looked into using NLTK's chunking functionality as before, however, unlike the previous situation where we simply wanted to extract the named entities themselves, it was now useful to have more information tagging these entities. After meeting with members of other groups in the class, we decided as a whole that spaCy's NLP library would be worth checking out. After

further investigation we found that given a sentence, spaCy can label everything from an amount of money to a date. This extensive labeling is perfect for answering the hurricane specific questions that we could not answer with just regular expressions.

As a visual example of this strategy is action consider the following application. When trying to find the overall category of the storm, the regex '[Cc]ategory ([0-9]+)' was used. This regex was used to find the overall category of the storm, as well as the category at landfall by first filtering just to sentences about landfall. It can been seen that this worked in how the results of these regexes were different and accurate. The histogram representation of these regex matches can be found in Figures 5 and 6.

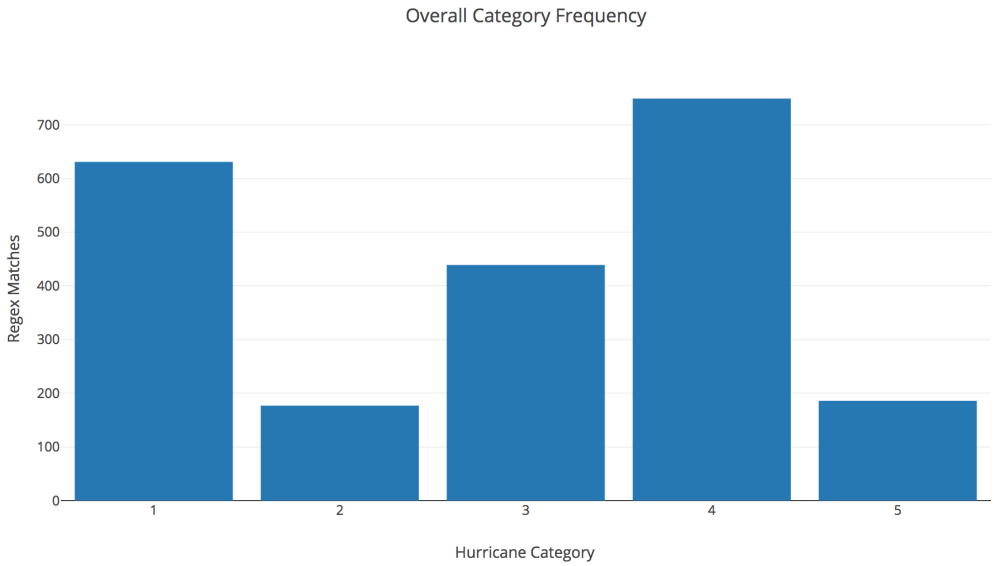

Figure 5: Regex matches for overall hurricane category

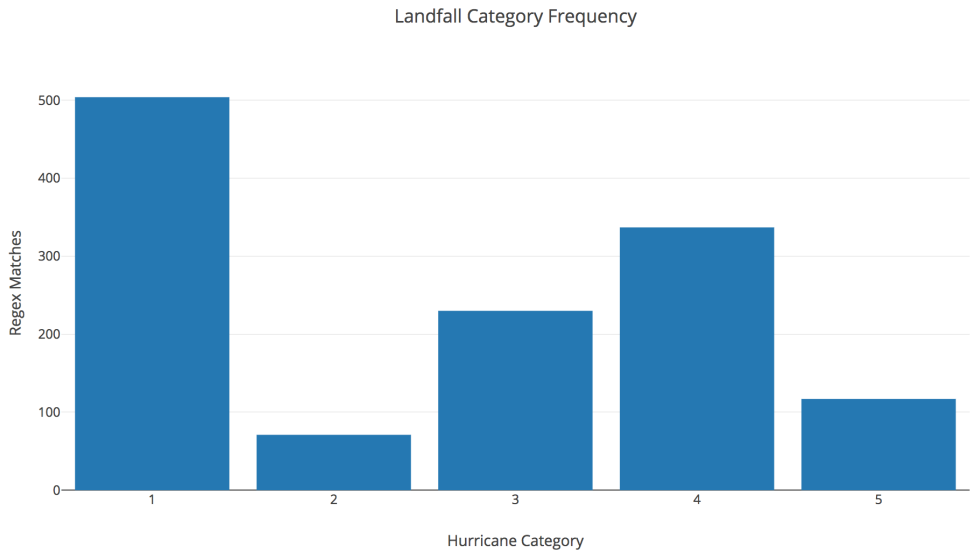

Figure 6: Regex matches for landfall hurricane category

Filling in the Remaining Holes

Now that we decided on a way to identify different types of information, we next needed to solve the problem of identifying the tagged entities that would answer our remaining questions. We accomplished this task in several different ways. Most often this was done through filtering through all of the sentences in the data set to get only those that are relevant to the question that we are trying to answer. An example of this is if we are trying to find out the average and maximum rainfall, simply filtering to only sentences that include variants of the word rain is surprisingly very effective. After looking though the remaining sentences after the filtering, we noticed that almost every single one of them had some form of information about the amount of rainfall. Since in this example we now only had sentences that were directly related to the question, we next had to decide how to get this information from the sentences.

For this task, we looked to spaCy's entity tagging. Since we knew now that these sentences all contained information about rainfall, if we now look at the phrases tagged with QUANTITY we can find that most of them are measurements of rain. When a sentence talked about multiple measurements, however, it was also possible that we could have things such as wind speeds mixed into this QUANTITY list. This can easily be solved, however, by filtering the quantities to only specific units that are appropriate for rain measurements. For this we would filter to just those quantities that were measured in cm or in. These were then passed through a helper method which standardized them by converting all units to inches, and then numpy was used to find statistics about the measurements. These statistics can be used to find the average of all amounts of rainfall measure in all of the articles that were given in our data set. To find the max of the values, however, is a bit more difficult. It would be naive to just take the max of this set as that would make it so any one article with an unrealistic value would ruin our results. It also is not practical to take the most common large number, or the average of the higher numbers as this could also be skewed by a small number of articles. What we decided to do in the end was to calculate the standard deviation of our data set and then choose the maximum to be the mean + one standard deviation. This gives us an above average value that cannot easily be skewed by a small number of measurements. As we will discuss later, this gave us very accurate results.

Although the previous paragraph talked exclusively about how we obtained our rainfall measurements, most of the strategies that we discussed can be applied to a large number of values. Wind speed, power outages, injuries, deaths, and storm speeds can all be measured in a very similar way. This is also almost identical to the strategy that we used to extract dates and events. The only difference is that the statistical analysis was done slightly differently. As a quick example, if we filter all of our sentences to those related to the storm making landfall, we can then extract and average the dates from these sentences to figure out when the storm hit land. We also can use these same sentences to find the most common geopolitical entities mentioned, which will be the landmarks where the storm was hitting. Applying these strategies to different topics was sped up very greatly by creating generic extraction methods that could be applied to any topic, however much of the debugging process took a long time, as the program had to tag every sentence in the data set before we could perform the filtering.

Wind speed and rainfall amounts are two of these template values that are most representative of how the data was collected. These measurements filled in values in our template summary though numerical analysis of a large set of potential values that we extracted. One visual analysis of these values can be found in 8 and 7 where we can see a box and whisker plot of the extracted values. When looking at the first and third quartile and the mean of these plots we can see values very

close to those that were put into the final template summary found in the appendix.

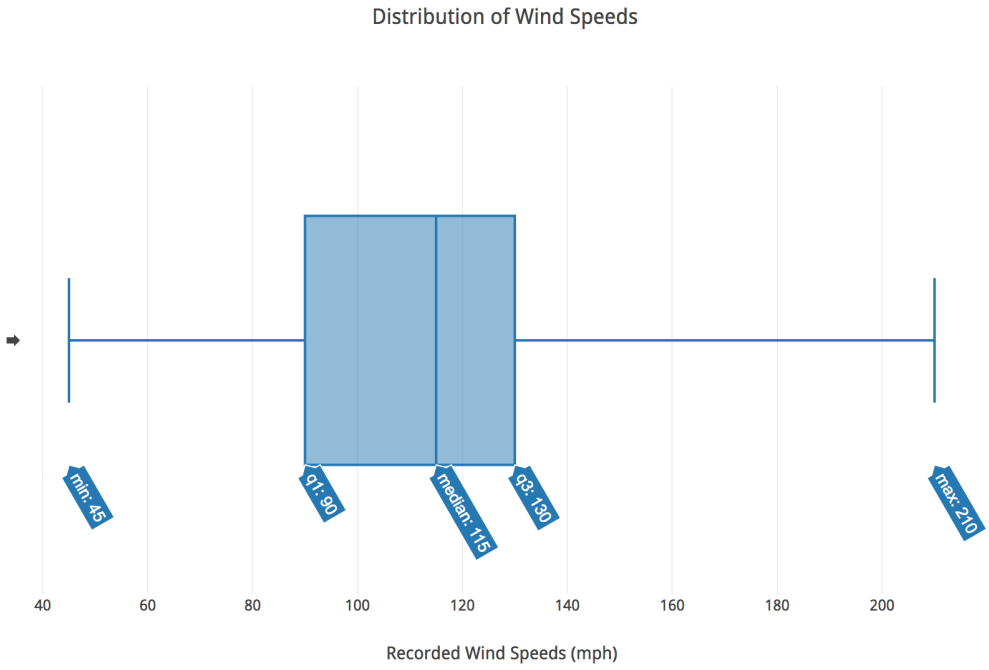

Figure 7: Distribution of the extracted values for wind speed

Reflective Thoughts

When comparing the values that we received from our summary to the official measurement, it is fair to say that we came extremely close to the actual values. As an example, for our measurements of maximum wind speeds and maximum rainfall, our results gave us 135 mph and 32 inches, respectively, when the actual maximum recorded values were 137 mph and 31 inches, according to Google. This is just a sample of how accurate our data was. Most of the figures that we came up with are just as close to the real values. In general when dealing with slot and value summaries, most of the variance in them comes from how close you can get to the real answers to the initial questions. The readable summary explaining the resulting values can stay the same each time so most of the impact comes from the values themselves. Because of this fact, we think that our summary is a very good slot and value solution.

What we think could be improved on in our template summary generation is the amount of information that we gathered, and, at times, the generality of our slot value answers. Although our summary has a substantial amount of information in it, to really compare to a human formulated summary you need information about every single part of the event. Our particular solution struggled with some numbers such as speeds of the storm at certain dates. The main difficulty of figuring information like this out is that it is difficult to distinguish from the date the article was written, an irrelevant date about a different piece of information, and the actual date that the information is about. This is due partially to the mechanical difficulty of associating the two pieces of information, but also due to the fact that even in the large data set there may not be that many articles with the specific information that we are seeking. The second weakness that was mentioned we believe to not be as major, however, some parts of the slot value system may not give as good results if thrown

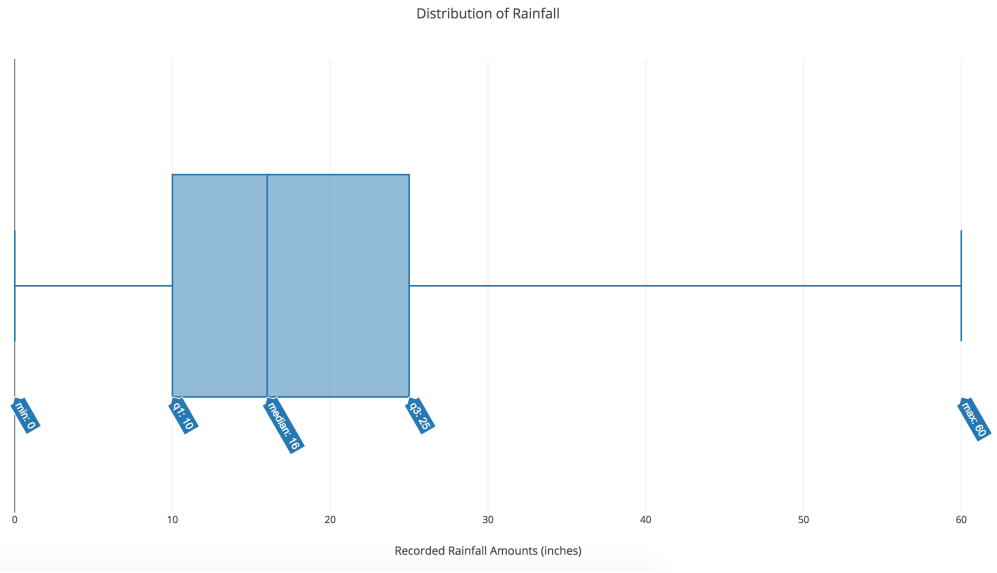

Figure 8: Distribution of the extracted values for rainfall

into a different type of storm. As an example, a shorter lasting storm would not have as many dates to pick out, and differing measurements which may cause skewed results when run through our program. Both of these problems are ones that are very difficult to avoid when attempting to fill out slots with values. We believe that, despite them, our summary is quite accurate.

4.9 A readable template summary

Now that the hard work of extracting specific values from the articles was finished, we needed a summary explaining these slots and values. This summary was something that we worked on hand in hand with the previous unit as the two were very closely related. The hurricane metadata page that we used as a starting point for the previous unit also had a section showing a summary that explained the slot values. We used a similar process as for the previous unit which was to use this section as an example of a summary, and then make our own that better fits the values that we extracted. This was done similarly to how we generated the golden standard summary wherein we gathered information first and then just logically put it together to form the summary based on the summary writing presentation that was given in class. The main ideas that we used were to start with a broad overview of the event, then give more details later on. We also made sure to only use the active voice and to not overgeneralize and add information that our program did not extract. The final summary generated by filling in this readable summary with the values from the previous unit can be found in Appendix J.

4.10 A readable abstractive summary

The sequence-to-sequence model

Due to the time needed to train recurrent neural networks, we use pre-trained models to generate the abstractive summaries. Our first attempt involved using a pre-trained version of the sequence-to-sequence neural network model, available on GitHub at https://github.com/thunlp/TensorFlow-Summarization. This model was trained on the GigaWord dataset, and comprised of a GRU encoder and a GRU decoder. However, the resulting summaries contained many unknown tokens, were

very short in length because the model was trained to produce headlines, and were not always semantically correct. As a result of that, we abandoned the sequence-to-sequence approach.

The Pointer-Generator model

The next approach we consider is a pre-trained Pointer-Generator model [2], available on GitHub at https://github.com/abisee/pointer-generator. This model was trained on the CNN / Daily Mail datasets, and is based on TensorFlow's sequence-to-sequence model.

Unlike the sequence-to-sequence model, the Pointer-Generator produces summaries that are between 2 and 3 sentences in length for each text fed into it, does not result in unknown tokens, and produces results that are mostly semantically correct. For these reasons, we adopt the Pointer-Generator approach going forward.

Clustering relevant records

Summarizing every single relevant record in our dataset would produce a summary that is both too long and incredibly redundant, since many records would have similar information. To prevent that, we adopt a clustering approach, similar to what is described in [1].

The PySpark machine learning module contains implementations of K-Means, Bisecting K-Means, and LDA clustering. In order to identify the best clustering mechanism, we ran a multitude of experiments, using all three methods on the dataset of relevant records obtained in Section 4.4. The Bisecting K-Means experiments all got stuck while training the Bisecting K-Means model. Since we were unable to determine why that happened, we excluded this clustering method from our experiments. Tables 4 and 5 summarize the cluster sizes obtained from these experiments.

Number of clusters	Number of features	Feature type	Cluster size summary			
			Minimum	IQR	Median	Maximum
5	100	counts	2	274	43	7466
5	1000	TF-IDF	3	211	45	7534
10	1000	counts	2	569	71	4465
10	5000	TF-IDF	1	56	26	6725
15	1000	counts	1	634	28	2702
15	10000	counts	1	634	28	2702
20	1000	TF-IDF	1	41	10	4741
20	5000	counts	1	88	17	3079

Table 4: Summary of cluster sizes from clustering experiments with K-Means

We then examine the sizes of the clusters generated with the K-Means and LDA methods with different parameters. Our intuition is that the experiments yielding clusters that are of about even size, with a small inter-quartile range, would produce the best clustering results. Our results indicate that the LDA clustering method, with 10-15 clusters, using a vocabulary size between 500 and 1000, and a feature set consisting of TF-IDF word scores, produces the most balanced clusters.

Number of clusters	Number of features	Feature type	Cluster size summary			
			Minimum	IQR	Median	Maximum
5	100	TF-IDF	149	866	1415	3086
5	1000	counts	11	1266	1586	2798
10	1000	TF-IDF	138	744	618	2116
10	5000	TF-IDF	0	1040	168	3102
15	1000	TF-IDF	105	508	358	1831
15	10000	counts	0	673	229	1945
20	1000	counts	0	377	8	2100
20	5000	TF-IDF	12	454	240	1389

Table 5: Summary of cluster sizes from clustering experiments with LDA

Abstractive summary result

To produce the abstractive summary, we ran LDA clustering on the records reconstructed from the relevant sentences obtained from the third classification result in Table 2, with a vocabulary size of 1000, TF-IDF word scores as the classification features, varying the number of clusters between 5, 10, and 15. The distribution of records for each of these clustering results is shown in Figure 9. For each clustering result, we then select the article closest to the center of each cluster (provided the cluster has at least one element), run the Pointer-Generator model on this set of articles, and concatenate the results.

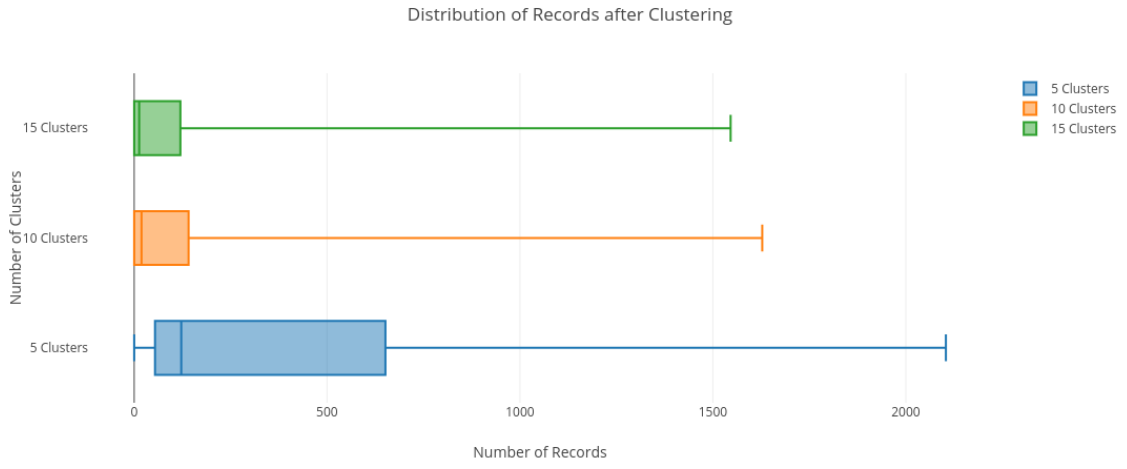

Figure 9: The distribution of records after clustering

We manually remove irrelevant sentences like image captions, and fix the spacing and capitalization errors in the resulting summaries. We also remove any summarized articles that are not relevant to Hurricane Florence, such as the article about Hurricane Isaac. The resulting summary produced from 10 clusters obtained the best ROUGE scores, and is shown in Appendix I.

5 Generating a golden standard summary for Team 14

Our task was to develop the golden standard summary for Team 14 whose topic was the Facebook Data breach. We decided that the best overall strategy to do this would be to first gather a set of articles about the event that we thought to be comprehensive. After this we would use this set of articles to fill out relevant information about the event so that it could be organized well. Finally, using this information we would develop our rough draft of the golden standard summary.

We began making the summary by using the Solr database to gather a set of articles that Team 14 was given that we thought to be useful. To do so we essentially read through about 50 articles and chose about half of them that were the most informative and least biased. This set of about 25 articles we found to cover the entire event in a very detailed manner and so we then moved on to information gathering.

The next step was to decide how we wanted to organize the information in the articles. A set of guidelines from the Test Analysis Conference of 2011 was linked on our class website that had a list of topics along with questions about the topics that could cover them well. Unfortunately, the Facebook Data Breach did not fit well into any of the topics. We solved this by choosing the topic that was most closely related which we found to be Attacks (Criminal/Terrorists). We of course had to modify some of the questions, however after finishing and filling out the information we came to an outline of the event that can be seen in Appendix A.

This information was then used by our group to write a summary in an upside down pyramid format where we first summarize the entire event and then give specific details in the later paragraphs. After the presentation by Professor Michael Horning, we went back and changed the summary to use the active voice consistently. This rough draft was reviewed and edited by all of the members of the group until we came to a version that we were satisfied with.

Finally, after receiving suggestions from the instructor, we included some information from online articles that expanded on and/or corrected the information we obtained from the Solr database of Team 14.

6 Evaluation

To evaluate our summaries, we used the Golden Standard Summary produced for our team by Team 9, which can be found in Appendix J, in conjunction with a slightly modified version of the evaluation script provided to us by the Graduate Teaching Assistant.

The provided evaluation script performs one of the following three assessments based on the number of parameters passed to it:

1. Rouge-para: Splits the produced and Golden Standard summaries into paragraphs, and calculates the average ROUGE-1, ROUGE-2, ROUGE-L, and ROUGE-SU4 scores between the paragraphs.

2. Rouge-sent: Splits the produced and Golden Standard summaries into sentences, and calculates the maximum ROUGE-1 and ROUGE-2 scores between the sentences.

3. Cov sent: Uses spaCy to extract the named entities from the produced and Golden Standard summaries, and finds the percentage of the named entities present in the Golden Standard

Summary that are also present in the produced summary.

The script was modified to perform all three evaluations, instead of one at a time. In our evaluation of the summaries produced, we made use of both the evaluation produced by the evaluation script, as well as human evaluation as performed by the members of our team.

6.1 Evaluating the template summary

Our human evaluation of the template summary was that it was very readable, albeit short, and provided very accurate answers to the template blanks. It flowed in a logical manner, due to the structure imposed by the template.

The ROUGE-based evaluation of the template summary, shown in Table 6, indicates that the template summary uses a reasonable number of the same words as the Golden Standard Summary, but misses a large chunk of the named entities present in the Golden Standard, and does not use many similar bigrams.

Rouge-para				Rouge-sent		Cov sent
ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-SU4	ROUGE-1	ROUGE-2	Coverage (%)
0.23333	0.03448	0.2	0.08537	0.6667	0.42857	3.05

Table 6: Rouge scores and named entity coverage of the template summary

6.2 Evaluating the extractive summary

Our human evaluation of the extractive summary was that it was much longer than the template summary, but much less readable. Some of the sentences appeared to contradict each other, and there was very little sense of flow in it, as the sentences jumped between time periods and topics. Furthermore, since sentences were extracted with no post-processing, the changes in topic were almost always abrupt. It also seemed to capture less of the details than the template summary did, but it did capture much more of the background of the Hurricane event, as well as more about the reactions to the Hurricane.

The ROUGE-based evaluation of the extractive summary, shown in Table 7, indicates that the extractive summary captured more of the words and named entities from the Golden Standard Summary than the template summary, but did not capture any of the bigrams.

Rouge-para				Rouge-sent		Cov sent
ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-SU4	ROUGE-1	ROUGE-2	Coverage (%)
0.26667	0.0	0.26667	0.06707	0.50000	0.40000	6.11

Table 7: Rouge scores and named entity coverage of the extractive summary

6.3 Evaluating the abstractive summary

Our human evaluation of the abstractive summary was that its readability and length fell somewhere between the extractive and template summaries. The paragraphs of the summary were more

cohesive than those of the extractive summary. It did seem to miss more information than either of the previous summaries, and the paragraphs were not in chronological order.

The ROUGE-based evaluation of the abstractive summary, shown in Table 8, indicates that the abstractive summary was comparable to the extractive summary in terms of capturing individual words from the Golden Standard Summary. It also captured more bigrams than either of the other two summaries, but still missed the overwhelming majority of the entities named in the Golden Standard Summary.

Rouge-para				Rouge-sent		Cov sent
ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-SU4	ROUGE-1	ROUGE-2	Coverage (%)
0.26667	0.06897	0.26667	0.08537	0.50000	0.25000	3.05

Table 8: Rouge scores and named entity coverage of the abstractive summary

7 User Manual

7.1 Requirements

In order to use this project, you need the following installed:

- Python 2.7 and Python3.4+
- git
- Hadoop
- Spark v2.0+
- Java 8

7.2 Getting Started

Use git to download the repository containing the codebase for this project from https://github.com/ffrankies/BigDataTextSummarization. After downloading the repository, use git to initialize the pointer-generator and pointer-generator/data_processor submodules.

Make sure that your data is in the form of a list of JSON objects, with the following keys, at minimum: URL_s, containing the URL from which the article was obtained, and Sentences_t, containing the cleaned text from the article.

Upload the data to the Hadoop cluster. This allows Spark to speed up processing on the dataset, since Hadoop will parallelize the data across multiple nodes.

Make sure that the JAVA_HOME environment variable is set to the directory containing the Java 8 runtime environment.

Install Anaconda, and create a new environment named nltk_env. Activate this new environment, and install the Python packages listed in requirements.txt. Afterwards, open a Python interpreter, import NLTK, and run the following command: *nltk.download(all)*.

Zip the following items and store them in the root directory of the project:

- The contents of the nltk_env environment, into nltk_env.zip.
- All the Python files in the project, into pyfiles.zip
- The tokenizers directory from the NLTK data directory, into tokenizers.zip
- The taggers directory from the NLTK data directory, into taggers.zip
- The corpora directory from the NLTK data directory, into corpora.zip

Lastly, create a configuration file named spark-defaults.conf, with the following contents:

spark.yarn.appMasterEnv.PYSPARK_PYTHON=./NLTK/nltk_env/bin/python
spark.yarn.appMasterEnv.NLTK_DATA=./
spark.executorEnv.NLTK_DATA=./
spark.yarn.dist.archives=nltk_env.zip#NLTK,tokenizers.zip#tokenizers,
taggers.zip#taggers,corpora.zip#corpora[3]
spark.eventLog.enabled=true
spark.eventLog.dir= /spark_logs
spark.ui.enabled=false

7.3 Tutorial

Running the project

After performing the steps above, the scripts in our repository are ready to be run. Each script can be run with the following command: *spark2-submit –master=local[4] –py-files=pyfiles.zip –properties-file=./spark-defaults.conf [script-name] [cmd-line arguments] [output redirection]*

The only scripts that are the exception to this rule are the Extractive Summarizer and Pointer-Generator Abstractive Summarizer, which are run with with python2 and python3, respectively.

Spark produces a lot of output that isn't necessarily useful, so we recommend discarding the error output from the script (*... 2 > /dev/null*) and redirecting the output of the scripts to a file (*... > outputfile.out*).

The scripts referred to here are described in Section 8.5. Before running the summary scripts, the following steps need to be followed:

1. Obtain a feature set from your dataset using the Feature Set Extractor.
2. Label a training set of records using the Automatic Labeling Script.
3. Obtain the relevant records from the automatically labeled dataset using the Classifier Script.
4. If needed, cluster the relevant records using the Clustering Script.
5. Download the resulting dataset from the Hadoop cluster using the *hadoop fs -get [filename]* command.

[3]This line was split so it would fit on one page: the spark-defaults.conf file should have this on the previous line

For a description of the command-line options available to each script, run the script with the *-h* option.

The template and extractive summaries can be run on the results obtained at this point by running the appropriate script. The abstractive summary requires the following additional steps before running the Pointer-Generator Abstractive Summarizer:

1. Tokenize the records to be summarized using the Article Tokenizer for the Pointer-Generator.
2. Preprocess the tokenized records using the Abstractive Summarizer for the Pointer-Generator.
3. Download and extract the pre-trained model, available at the following URL: https://drive.google.com/file/d/0B7pQmm-OfDv7ZUhHZm9ZWEZidDg/view.

To obtain good results with the abstractive summary, the vocabulary file used in creating the summary should be the one produced by the pre-trained model. This vocabulary file (as well as the pre-processed CNN dataset) is available at the following URL: https://drive.google.com/file/d/0BzQ6rtO2VN95a0c3TlZCWkl3aU0/view.

For more information on running the pointer-generator model, consult the README at the following URL: https://github.com/abisee/pointer-generator.

8 Developers Manual

8.1 Project Architecture

The ten steps into which the project was broken down facilitated a simple, cumulative project architecture. Every step of the project corresponds to a Python file (except in the cases where a stage is complex enough to warrant splitting it up into multiple files), and each successive stage depends on the previous one. This ensured that we minimized duplicating code that we had already written, with the drawback of propagating flaws in earlier steps throughout our project.

The Python scripts for all ten steps rely on Spark, JAVA 8, PySpark, and Hadoop to run properly. This is done in order to parallelize the processing of our large web page collection, which consists of 10,948 web pages - a process that would have been too slow to run on a single node.

8.2 Data Processing

The bulk of the data processing was done using the provided Scala script, which processed the WARC web page collection, stripped the HTML tags, and output a JSON file containing the text in the body of each web page.

We found that this processing was sufficient, as we did not find any HTML or JavaScript artifacts in our processed articles.

On top of this, we wrote several pre-processing methods, which can be called as-needed, to perform the following functions:

- Tokenize the words in each article
- Tag the tokenized words with their parts of speech using the pos_tag method built into NLTK
- Filter out punctuation, stopwords, and non-dictionary words from the tagged and tokenized words
- Lemmatize the tagged words

8.3 Installation

- Download and install the Java 8 JDK, which is needed for Scala
- Download and install Scala v2.11
- Download and install Apache Spark v2+
- Install the Python requirements listed in requirements.txt

8.4 Future Work

Classifying relevant records and sentences

The results of the abstractive summary indicate that the classifier described in Section 4.4 does not do a satisfactory job of distinguishing between relevant and irrelevant records and sentences. This may be because the automatic labeling technique only labels the extreme cases, where a record/sentence is either completely irrelevant, or contains a large number of important features.

Some possible solutions include manual labeling of records and sentences, or using more sophisticated feature sets, such as word2vec. Exploring other classifiers, like random forests and decision trees, is another option to consider.

Clustering records and sentences

Our code for the bisecting K-Means clustering algorithm never completes execution, so one obvious area for future work is to fix that, and compare its results to the original K-Means and LDA clustering algorithms.

Another area for improvement is our metric for selecting the clustering algorithm and the parameters with which to run said algorithm. PySpark's implementations of LDA and K-Means produce different evaluation metrics (log perplexity and within set sum of squared errors, respectively). Due to this, we need to manually calculate some common metric that we can use to directly compare the results of the two algorithms. This would help in selecting the best algorithm and the parameters with which to run it, since our solution - to pick the algorithm and parameters that produce the most balanced clusters in terms of their size - does not guarantee that the resulting clusters are compact.

Abstractive summarization

One idea to improve the abstractive summaries we generate is to select the cluster centers instead of random articles from within the clusters. PySpark's LDA model does not have a notion of cluster centers, and its K-Means model allows users to extract the feature combinations that are at the

cluster centers, but does not have an easy way to access the record/item that is closest to the cluster center. Thus, this would involve calculating distances between cluster centers and the records in each cluster for K-Means, and, additionally, calculating the cluster centers for LDA. Another idea is to select multiple articles from within clusters, to hopefully retain more information from each cluster topic.

Training the Pointer-Generator network with different parameters is another option, since the performance of neural networks varies with the parameters chosen.

A third option here is to iteratively perform extractive summarization on each cluster, and then run abstractive summarization on the results. This could help retain more of the information present in each cluster, since all the records will be used in the summarization process.

8.5 Inventory

In the interest of readability, we define the following variables:

- **UNLABELED: /home/public/cs4984_cs5984_f18/unlabeled/data**: The directory in which the unlabeled data is stored.
- **UNLABELED_SCRIPTS: /home/public/cs4984_cs5984_f18/unlabeled/script**: The directory in which the unlabeled preprocessing scripts are stored.
- **HADOOP: /user/cs4984cs5984f18_team12**: The Apache Hadoop file system directory for team 12.
- **HOME: /user/cs4984cs5984f18_team12**: The home directory for team 12.
- **REPOSITORY: https://github.com/ffrankies/BigDataTextSummarization**: The repository housing the code for this project.

Provided Dataset Files

The following dataset files were provided to us by the professor and/or the teaching assistant for the class.

- **Big WARC Dataset Index (UNLABELED/12_Hurricane_Florence_big/12_Hurricane_Florence_big.cdx and HADOOP/12_Hurricane_Florence_big/Hurricane_Florence_big.cdx)**: The index file to the big Hurricane Florence unlabeled dataset.
- **Big WARC Dataset (UNLABELED/12_Hurricane_Florence_big/12_Hurricane_Florence_big.warc.gz and HADOOP/12_Hurricane_Florence_big/Hurricane_Florence_big.warc.gz)**: The original WARC file containing the records of the big Hurricane Florence unlabeled dataset.
- **Small WARC Dataset Index (UNLABELED/12_Hurricane_Florence_small/12_Hurricane_Florence_small.cdx and HADOOP/Hurricane_Florence_small/Hurricane_Florence_small.cdx)**: The original index file to the small Hurricane Florence unlabeled dataset.

- **Small WARC Dataset (UNLABELED/12_Hurricane_Florence_small/12_Hurricane_Florence_small.warc.gz and HADOOP/Hurricane_Florence_small/Hurricane_Florence_small.warc.gz)**: The original WARC file containing the records of the small Hurricane Florence unlabeled dataset.

Provided Scripts

The following starter files and scripts were provided to us by the professor and/or the teaching assistant for the class.

- **WARC to JSON Converter (UNLABELED/script/ArchiveSpark_sentence_extraction.scala)**: The starter script for processing the provided WARC-format datasets into JSON-formatted datasets.
- **Evaluation Script (REPOSITORY/eval.py)**: The script for using ROUGE to evaluate the produced summaries.

Modified Preprocessing Scripts

The following preprocessing scripts are slightly modified versions of the provided preprocessing scripts.

- **WARC to JSON Converter for the Big Dataset (HOME/extract_big_dataset.scala)**: A modified version of the WARC to JSON Converter script, used for transforming the Big WARC Dataset into the Big JSON Dataset.
- **WARC to JSON Converter for the Small Dataset (HOME/extract_small_dataset.scala)**: A modified version of the WARC to JSON Converter script, used for transforming the Small WARC Dataset into the Small JSON Dataset.

Processed Data Files

The following processed data files were created by the members of Team 12 during the course of this project.

- **Big JSON Dataset (HADOOP/Hurricane_Florence_big/sentences/part-00000-f466ce4e-9518-4ab2-b15f-987f3643a84d-c000.json)**: The JSON-formatted version of the Big WARC Dataset.
- **Small JSON Dataset (HADOOP/Hurricane_Florence_small/sentences/part-00000-579e0208-41cd-4b73-abc3-fa7287550223-c000.json)**:The JSON-formatted version of the Small WARC Dataset.

Python Scripts

The following Python scripts were developed by the members of Team 12 to address the project requirements.

- **Constants File (REPOSITORY/constants.py)**: Holds common variables such as dictionary indexes, that are used by multiple files within the project.

- **Word Counter (REPOSITORY/wordcount.py)**: Counts words and collocation bigrams present in the dataset. Also contains code for loading and preprocessing dataset records, and for initiating the spark context.
- **Important Word Finder (REPOSITORY/tfidf.py)**: Uses the TF-IDF method to extract important words from the dataset, and filters collocation bigrams so the list returned only contains bigrams made out of important words.
- **Synset Maker (REPOSITORY/synsets.py)**: Uses the NLTK WordNet module to create synsets from a given list of words.
- **Part of Speech Tagger (REPOSITORY/post.py)**: Creates a list of important nouns and words in the given dataset using NLTKs part of speech tagger, in combination with the Imporant Word Finder script.
- **Named Entity Extractor (REPOSITORY/namedEntity.py)**: Uses NLTKs named entity chunking method to extract named entities from a given dataset.
- **Topic Analyzer (REPOSITORY/lda.py)**: Performs topic analysis on a given dataset using the LDA implementation from the Gensim library.
- **Feature Set Extractor (REPOSITORY/featureset.py)**: Uses the output of the Word Counter, Important Word Finder and Synset Maker to produce a feature set for filtering unrelated records out of the dataset.
- **Automatic Labeling Script (REPOSITORY/autolabel.py)**: Uses two separate thresholds, and the output of the Feature Set Extractor, to automatically label a training set of records in the dataset.
- **Classifier Script (REPOSITORY/mlp.py)**: Uses a multi-layer perceptron, in combination with the output of the Automatic Labeling Script, to classify records as either relevant or irrelevant to the Hurricane Florence topic.
- **Extractive Summarizer (REPOSITORY/extractive.py)**: Uses the pyteaser library to create extractive summaries of the dataset.
- **Extraction Helper Functions (REPOSITORY/extractionHelpers.py)**: Contains helper functions for information extraction from the dataset.
- **Template Summarizer (REPOSITORY/infoExtraction.py)**: Contains the code for generating extracting information from the dataset and inserting it into the summary template slots.
- **Clustering Script (REPOSITORY/cluster.py)**: Performs clustering on a given dataset using the K-Means or LDA algorithm provided in PySparks machine learning module.
- **Article Tokenizer for the Pointer-Generator (REPOSITORY/pointer-generator/data_processor/json_to_hash.py)**: Tokenizes the articles from a given JSON dataset.
- **Article Preprocessor for the Pointer-Generator (REPOSITORY/pointer-generator/data_preprocessor/make_datafiles.py)**: Processes the tokenized articles into the proper format for the Pointer-Generator abstractive summarizer.

- **Pointer-Generator Abstractive Summarizer (REPOSITORY/pointer-generator/run_summarization.py)**: Performs abstractive summarization on the preprocessed dataset.

9 Lessons Learned

The absence of productivity tools like git on the Hadoop cluster made it harder for us to properly utilize the Hadoop cluster. We ended up manually copying files from our local git workspaces to the Hadoop cluster using scp whenever we needed to update the code on the cluster, which was both inefficient and led to conflicts.

Many of the tools like PySpark, Hadoop, and Anaconda were new to all the members in our group. While they were certainly useful, we spent more time either misusing them or avoiding them altogether - leading to scripts that took too long to run, or failed when they were run on the cluster. This also led to too much time being spent debugging various errors with the tools used.

In an attempt to produce results faster, we did not pay much attention to best practice in software development. This ended up slowing us down, since we often misunderstood how code written by other team members worked.

Finally, we spent too little time looking at the big picture of the project, and too much time working on the details of the first five steps. As such, we felt rushed when producing the actual summaries, since we were behind schedule. Had we paid more attention to the big picture and started out with the abstractive, extractive, and template summaries in mind, we would likely have both been less stressed, and produced better summaries.

In the future, when working on similar projects, we would likely try to spend some time at the beginning familiarizing ourselves with the new technology, until all the team members are reasonably comfortable using them. Additionally, we would make better use of GitHub's code reviews and third-party integrations to help us write more readable code. Finally, we would try to spend more time understanding the final deliverable of the project, and focus more on getting that done instead of getting stuck in the details of the implementation plan.

10 Conclusions

We found that the template summarization method produced the best summary (Appendix G). It was the only one that was coherent throughout, and contained the most important information about the hurricane. Additionally, many of the values that we were able to extract from the articles were very close to the actual values. However, the process for selecting information for the template summary is slow, and has to be performed manually. It might have been much faster to simply search through the dataset and obtain the information, instead of writing regex expressions to do the same. Another disadvantage of this method is that only information that we specifically look for will be included, which may cause our summary to be too dependent on our specific dataset.

Although the template summary is the most readable, that is only because the template that the values were put into was made by a human. It can be said however, that if a program can extract a larger amount of values about an event in a more accurate and fast way, that throwing these

values into a template will always create a very readable and accurate summary meaning that this template summary approach does have a large amount of potential if it were improved.

The abstractive and extractive summaries produced sub-par results. Neither summary was particularly cohesive, and both contained irrelevant information, and not enough figures and other details about the hurricane. However, they did successfully capture more background information about the hurricane, as well as the reactions to it.

Our intuition is that better data cleaning, and better selection of articles for summarization, would help to produce much better abstractive and extractive summaries. Furthermore, in the case of the extractive summary, having some sort of topic or headline to provide to the summarizer could help produce better results.

A concern with our summarization methods is the low amount of named entity coverage produced by our summaries. There could be multiple reasons for this, including the Golden Standard Summary not being specific enough, and the entity coverage analyzer not equating similar but different numerical values. For example, 100 mph and 101 mph would be treated as different named entities, even though they are close enough to make little difference. Since named entities encode important information, however, there is a need to improve our summarization methods in order to make the summaries produced capture more important named entities.

Overall, most of the techniques that we employed were correct, but simply require more time or testing to generate a perfect summary of the event. Given the constraints of a semester to work on the project, there was not much more that we could have done to improve the project. This being said, following the methods that we laid out with a larger time frame for testing will almost certainly lead to a readable and complete summary of Hurricane Florence, or any other hurricane for which a large set of articles is obtained.

11 Acknowledgements

We would like to acknowledge the following groups and individuals for their contributions to this project:

- Liuqing Li (liuqing@vt.edu), who was the GTA for the Big Data Text Summarization class, for helping us get PySpark running on the Hadoop cluster.
- Chreston Miller from Team 7 (chmille3@vt.edu), for providing a script for preprocessing data for the pointer-generator summarization model.
- Michael Horning (mhorning@vt.edu) is a professor at Virginia Tech, for giving an in-class presentation on news summarization, and for a helpful discussion on additional data we could include in our summary.
- James Xie (https://github.com/leopard1) and Lei Xu (https://github.com/leix28), for making their sequence-to-sequence neural network code, as well as a pre-trained model, available on GitHub.
- Abigail See (https://github.com/abisee/pointer-generator), for making the Pointer-Generator code, as well as a pre-trained model, available on GitHub.

- Xiao Hu (https://github.com/xiaoxu193/PyTeaser), for adapting TextTeaser into a Python library and publishing it on GitHub for use.
- Jolo Balbin (https://github.com/MojoJolo/textteaser), for writing and releasing the original TextTeaser API.
- Chava Raja Venkata Satya Phanindra, Dhar Siddharth, Gaur Yamini, Rambhakta Pranavi and Shetty Sourabh from Team 9, for their work on the Golden Standard Summary for the Hurricane Florence dataset.
- The Summa NLP organization, for making the Summa library available on GitHub.

The dataset for this project was obtained with the support of NSF DUE-1141209 and IIS-1319578.

Bibliography

[1] Y. J. Kumar, O. S. Goh, H. Basiron, N. H. Choon, and P. C. Suppiah, "A Review on Automatic Text Summarization Approaches," *Journal of Computer Science*, vol. 12, no. 4, pp. 178–190, 2016. DOI: 10.3844/jcssp.2016.178.190. [Online]. Available: http://thescipub.com/PDF/jcssp.2016.178.190.pdf.

[2] A. See, P. J. Liu, and C. D. Manning, "Get To The Point: Summarization with Pointer-Generator Networks," in *Association for Computational Linguistics*, vol. 1, Vancouver, Canada, 2017, pp. 1073–1083. DOI: 10.18653/v1/P17-1099. arXiv: 1704.04368. [Online]. Available: https://arxiv.org/abs/1704.04368.

[3] D. K. Gaikwad and C. N. Mahender, "A Review Paper on Text Summarization," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 5, no. 3, pp. 154–160, 2016. DOI: 10.17148/IJARCCE.2016.5340. [Online]. Available: https://www.ijarcce.com/upload/2016/march-16/IJARCCE40.pdf.

[4] M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, K. Kochut, and E.-B. D. Trippe, "Text Summarization Techniques: A Brief Survey," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 10, pp. 397–405, 2017. DOI: 10.14569/IJACSA.2017.081052. arXiv: arXiv:1707.02268v3. [Online]. Available: https://arxiv.org/abs/1707.02268.

[5] J. Ramos, "Using TF-IDF to Determine Word Relevance in Document Queries," in *Proceedings of The First Instructional Conferences on Machine Learning*, 2003. DOI: 10.1.1.121.1424. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.1424{\&}rep=rep1{\&}type=pdf.

[6] R. Mihalcea and P. Tarau, "TextRank: Bringing Order into Text," in *Proceedings of EMNLP*, Stroudsburg, Pennsylvania: Association for Computational Linguistics, 2004, pp. 404–411. [Online]. Available: http://aclweb.org/anthology/W04-3252.

[7] F. Barrios, F. López, L. Argerich, and R. Wachenchauzer, "Variations of the Similarity Function of TextRank for Automated Summarization," in *Proceedings of Argentine Symposium on Artificial Intelligence*, Sociedad Argentina de Informática e Investigación Operativa, 2015, pp. 65–72. [Online]. Available: http://sedici.unlp.edu.ar/handle/10915/52082.

[8] W. A. Hollingsworth, "Using lexical chains to characterise scientific text," PhD thesis, University of Cambridge, 2008. [Online]. Available: https://ethos.bl.uk/OrderDetails.do;jsessionid=0D6EA59350E32781968819AF3DDB5D5F?uin=uk.bl.ethos.604174.

[9] M. Marathe and G. Hirst, "Lexical Chains Using Distributional Measures of Concept Distance," in *Computational Linguistics and Intelligent Text Processing*, Iaşi, Romania, 2010, pp. 291–302. DOI: 10.1007/978-3-642-12116-6_24. [Online]. Available: http://link.springer.com/10.1007/978-3-642-12116-6{_}24.

[10] W. Hollingsworth, *US8676567B2 - Automatic text skimming using lexical chains - Google Patents*, 2007. [Online]. Available: https://patents.google.com/patent/US8676567.

[11] ——, *Accessible Learning - With Artificial Intelligence*. [Online]. Available: http://accessible.blog/ (visited on 11/28/2018).

[12] S. Meredith, *Facebook-Cambridge Analytica: A timeline of the data hijacking scandal*, 2018. [Online]. Available: https://www.cnbc.com/2018/04/10/facebook-cambridge-analytica-a-timeline-of-the-data-hijacking-scandal.html (visited on 11/26/2018).

[13] A. Hern and D. Pegg, *Facebook fined for data breaches in Cambridge Analytica scandal — Technology — The Guardian*, 2018. [Online]. Available: https://www.theguardian.com/technology/2018/jul/11/facebook-fined-for-data-breaches-in-cambridge-analytica-scandal (visited on 11/26/2018).

[14] J. Bennet, K. Kingsbury, M. Cottle, M. Gay, C. Giacomo, J. Interlandi, S. Jeong, L. Kelley, S. Schmemann, B. Staples, J. Wegman, J. Broder, and N. Fox, *Opinion — Did Facebook Learn Anything From the Cambridge Analytica Debacle? - The New York Times*, 2018. [Online]. Available: https://www.nytimes.com/2018/10/06/opinion/sunday/facebook-privacy-breach-zuckerberg.html (visited on 11/26/2018).

[15] M. Saro, *Why the recent Facebook/Cambridge Analytica data 'breach' matters for students*, 2018. [Online]. Available: https://www.brookings.edu/blog/brown-center-chalkboard/2018/06/06/why-the-recent-facebook-cambridge-analytica-data-breach-matters-for-students/ (visited on 11/26/2018).

Appendix A

Golden Standard Summary Outline

The following is the outline used to generate the Golden Standard Summary for Team 14 about the Facebook data breach.

1 Attacks (Cyber)

1.1 WHAT: what happened

- In 2014 a Russian/American researcher, Aleksandr Kogan, was authorized by Facebook to view information about people who used his personality quiz app "thisisyourdigitallife".
- The data was to be used for research purposes and should have only given him information on the 270,000 people who agreed to download and use his app.
- He abused the fact Facebook's terms and services also allowed him to access the data of all of the friends of the people using his app. This changed the number of people that he was harvesting data on from 270,000 to 50 million.
- This data was then shared with Cambridge Analytica (apparently without Facebook's knowledge). CA then used this to categorized and profile the 50 million users so that they could better target them with adds.
- These targeted ads were then used to aid the Trump campaign in the 2016 election.

1.2 WHEN: date, time, other temporal placement markers

- In March of 2018 it was revealed to the public that Cambridge Analytica had been misusing Facebook user data that they harvested in 2014 and 2015.
- It was also revealed that despite having been asked to destroy the data be Facebook that the company had not done so.
- 2018 a whistleblower, Christopher Wylie can to the public.
- Throughout the 2016 presidential election, CA was paid by the Trump administration to help target specific voters with personalized ads.
- 2018 Facebook suspended all CA accounts.

1.3 WHERE: physical location

- Cambridge Analytica is a data analytics company based in the United Kingdom.
- The majority of the victims of the attack were American citizens participating in the 2016 election.

1.4 PERPETRATORS: individuals or groups responsible for the attack

- Facebook gave out data.
- Cambridge Analytica used data harvested against user will.
- Trump administration paid at least $6-15 million to CA to help target voters to assist in their election.
- SCL group, the parent company of Cambridge Analytica.

1.5 IMPORTANCE: importance of the attack

- More than 50 million people had their personal data harvested without their knowledge.
- In general, when advertising though a site like Facebook, the options for who the ads are shown to are quite limited to things such as "Users who liked a specific page." The ad targeting that CA did however was on such a massive scale since they were looking at so many users, and way more specific since they looked at personal details on each user to try to figure out what type of person they are.
- Usually cyber attacks like this in the past have been ignored, however people have been outraged by this data breach.
- Instead off focusing the outrage on the researcher who may have broken his contract with Facebook, people are upset that Facebook is selling large amounts of their data to third party companies just because it says they are allowed to in the user agreement.
- The users who downloaded and used the personality app agreed to the terms and services, but their Facebook friends who also had their data harvested never had even heard of the app and were still victims.
- Because of the public outrage from this incident, the media has started the conversation about how important terms and services are and who the data we store about ourselves online belongs to.

1.6 DAMAGES: damages caused by the attack

- Person privacy of millions of people who used Facebook was breached and used against them by CA.
- Voter manipulation was able to occur at a massive scale because of the amount of data that CA obtained.

1.7 AFTEREFFECTS: changes that came after this event

- Facebook company stock fell dramatically throughout the scandal and the court case that followed it. About 15% drop.
- Many Facebook user accounts deleted.
- Lawsuits filed and investigations begun.
- Facebook removed the friends of users loophole in 2015. Facebook reduced the amount of data available through their graph API also.
- The Facebook CEO Mark Zuckerburg was called to appear in front of the senate judiciary committee to reveal information about the breach.

Appendix B

Golden Standard Summary

The following is the Golden Standard Summary generated from the outline in Appendix A.

In 2014 Facebook authorized a Russian/American researcher named Aleksandr Kogan to view information about people who used his personality quiz app "thisisyourdigitallife". This data was to be used for research and should have only given him information on the 270,000 people who agreed to download and use his app. Kogan abused Facebook's terms of service by accessing the data of all the friends of the people using his app. This underhanded technique changed the number of people that he was harvesting data on from 270,000 to 87 million [12]. He then shared this data with Cambridge Analytica, a data analytics company based in the United Kingdom. Cambridge Analytica used this data to help categorize and profile more than 50 million people so that they could be better targeted with advertisements. Upon finding out about the massive data harvesting, Facebook removed "thisisyourdigitallife" from their platform, issued a mandate to Cambridge Analytica to destroy all the data they gathered, and fixed the loophole that subjected users to having their data exposed by their Facebook friends. However, Cambridge Analytica did not comply with Facebook's mandate, and since Facebook never checked that the data had been deleted, Cambridge Analytica was able to use this data in the 2016 election to aid the Trump campaign.

Throughout the 2016 presidential election, the Trump campaign paid Cambridge Analytica to help target voters with personalized ads. Giving users personalized advertisements may seem innocent at first, however in reality this was voter manipulation on a national scale using stolen data. In general, when advertising through a site like Facebook, the options for selecting who the ads are shown to are limited to things such as "Users who liked a specific page." The ad targeting carried out by Cambridge Analytica was on a more massive scale since they were looking at so many users, and was more specific since they looked at personal details on each user to try to figure out what type of person they are. This was all kept hidden from both Facebook and its users, and it was not until much later that the details of the situation came out.

In March of 2018, it was revealed to the public that Cambridge Analytica had been misusing the user data that they harvested from Facebook between 2014 and 2015. Christopher Wylie, a whistleblower from Cambridge Analytica helped expose the specifics of the situation, such as the scale of the operation and timeline of the events. In the past, most cyber attacks like this had been largely ignored, but something about this event specifically made people outraged. People were upset that Facebook was selling large amounts of their data to third party companies simply because the user agreement says they are allowed to do so. The users who downloaded and used the personality app agreed to the terms and services, but their Facebook friends, who also had their data harvested, were

affected despite never having heard of "thisisyourdigitallife". These events sparked a nationwide discussion on the importance of terms and services agreements, as well as who the personal data that users put online belongs to.

Facebook stock fell dramatically as a result of the scandal and the court case that followed it, with about a 15% decrease in stock price. Facebook was also fined £500,000 by the British Information Commissioner's Office (ICO) for two breaches of the Data Protection Act [13]. Along with this decrease in market value, many users also began deleting their accounts, to prevent themselves from becoming victims of similar privacy breaches in the future. Meanwhile, Cambridge Analytica's parent company, Strategic Communication Laboratories (SCL) Elections, was criminally persecuted for misusing the data they obtained from Facebook and not complying with data privacy laws. Two months after the breach was reported by the Observer, SCL Elections filed for bankruptcy [13].

In an attempt to secure their user's data and improve their public image, Facebook has more recently begun limiting the amount of data that is available through Facebook's Graph API, which is essentially a tool to allow programmers to get data about their ads, and the users their ads are being shown to. In addition to this, Facebook also revoked all of the accounts of Cambridge Analytica's parent company, SCL Elections, to ensure that they no longer have access to any user data. Overall, as a result of this scandal Facebook revamped their data privacy policies.

Despite these changes, there are concerns that Facebook did not take the breach seriously enough. In September of 2018, Facebook announced a different data breach that affected 50 million of its users. Due to three bugs in Facebook's code, hackers were able to obtain the phone numbers of its users. Ironically, the reason Facebook had access to these numbers in the first place was to provide two-factor authentication, a measure intended to increase the security on their users' accounts. This potentially allowed the hackers to access any of the services that these users are log into with their Facebook profile, including Tinder and Instagram [14].

Finally, the Facebook data breach started an important discussion about data privacy, and the increasing potential for its abuse by other companies and organizations. One example of this is the data collected on over 50 million public school students in the United States, which is also shared with researchers and third-party developers. This data is often stored on multiple storage systems, which increases the potential for a similar data breach to occur [15]. Concerns have also been raised about the kind of data political campaigns harvest, and how that information is harvested. In the United Kingdom, the ICO uncovered that the British Labour party was using data provided by Lifecycle Marketing (Mother & Baby) Limited, a company whose main goal is supposed to be providing information to pregnant women and new mothers. The ICO claims that Lifecycle Marketing collected data with no transparency as to how it will be used, and without consent. This data was then used by political parties to profile and target the country's population [13].

Appendix C

TF-IDF Results

The following are the 200 most important words in our collection, with their $tfidf_w$ scores, as identified by the modified TF-IDF method described in Section 4.1.

1. north: 0.00449
2. storm: 0.00441
3. florence: 0.00435
4. hurricane: 0.00411
5. today: 0.00405
6. beach: 0.00343
7. south: 0.00319
8. water: 0.00296
9. coast: 0.00289
10. flood: 0.00288
11. people: 0.00286
12. rain: 0.00281
13. county: 0.00271
14. news: 0.00267
15. river: 0.00259
16. tropical: 0.00255
17. center: 0.00254
18. home: 0.00250
19. high: 0.00241
20. surge: 0.00237
21. area: 0.00235
22. emergency: 0.00229
23. landfall: 0.00225
24. power: 0.00223
25. state: 0.00219
26. category: 0.00218
27. forecast: 0.00216
28. flooding: 0.00215
29. city: 0.00214
30. weather: 0.00208
31. day: 0.00208
32. national: 0.00204
33. make: 0.00202
34. move: 0.00202
35. time: 0.00200
36. east: 0.00199
37. myrtle: 0.00193
38. leave: 0.00186
39. evacuation: 0.00186
40. wind: 0.00183
41. data: 0.00182
42. coastal: 0.00182
43. share: 0.00179
44. trump: 0.00175
45. morning: 0.00174
46. week: 0.00173
47. rainfall: 0.00168
48. million: 0.00167
49. inland: 0.00167
50. heavy: 0.00161
51. post: 0.00159
52. close: 0.00159
53. rise: 0.00159
54. atlantic: 0.00158
55. information: 0.00157
56. house: 0.00156
57. hit: 0.00154
58. work: 0.00153
59. damage: 0.00152
60. continue: 0.00151
61. order: 0.00151

62. west: 0.00151
63. cape: 0.00149
64. track: 0.00149
65. eastern: 0.00146
66. bring: 0.00145
67. read: 0.00142
68. local: 0.00142
69. night: 0.00141
70. find: 0.00140
71. late: 0.00140
72. service: 0.00139
73. stay: 0.00139
74. ocean: 0.00139
75. show: 0.00138
76. air: 0.00138
77. major: 0.00137
78. site: 0.00137
79. watch: 0.00136
80. prepare: 0.00135
81. united: 0.00134
82. island: 0.00132
83. fear: 0.00131
84. region: 0.00131
85. live: 0.00131
86. provide: 0.00130
87. dangerous: 0.00129
88. early: 0.00129
89. year: 0.00128
90. disaster: 0.00128
91. force: 0.00127
92. evacuate: 0.00125
93. open: 0.00125
94. link: 0.00123
95. ahead: 0.00123
96. effect: 0.00123
97. path: 0.00122
98. family: 0.00122
99. remain: 0.00121
100. fall: 0.00120
101. low: 0.00119
102. catastrophic: 0.00118
103. sea: 0.00118
104. head: 0.00117
105. base: 0.00117
106. reserve: 0.00115
107. land: 0.00115
108. top: 0.00115
109. check: 0.00114
110. public: 0.00114
111. change: 0.00114
112. back: 0.00113
113. southeast: 0.00111
114. content: 0.00111
115. free: 0.00110
116. due: 0.00110
117. president: 0.00110
118. station: 0.00109
119. part: 0.00109
120. drive: 0.00108
121. line: 0.00108
122. potential: 0.00107
123. food: 0.00107
124. point: 0.00107
125. twitter: 0.00107
126. team: 0.00107
127. policy: 0.00106
128. world: 0.00106
129. big: 0.00105
130. strong: 0.00105
131. large: 0.00104
132. management: 0.00103
133. place: 0.00103
134. long: 0.00102
135. federal: 0.00102
136. sign: 0.00102
137. mandatory: 0.00101
138. travel: 0.00101
139. turn: 0.00101
140. number: 0.00100
141. medium: 0.00100
142. comment: 0.00099
143. southern: 0.00099
144. affect: 0.00099
145. reach: 0.00098
146. video: 0.00098
147. business: 0.00098
148. call: 0.00097
149. begin: 0.00097
150. company: 0.00097
151. afternoon: 0.00096

152. privacy: 0.00096
153. bad: 0.00095
154. cover: 0.00095
155. start: 0.00095
156. system: 0.00095
157. weekend: 0.00093
158. impact: 0.00092
159. plan: 0.00092
160. eye: 0.00092
161. report: 0.00091
162. ago: 0.00091
163. story: 0.00091
164. tell: 0.00091
165. government: 0.00090
166. click: 0.00089
167. contact: 0.00088
168. event: 0.00088
169. safe: 0.00088
170. run: 0.00087
171. response: 0.00087
172. support: 0.00087
173. approach: 0.00086
174. end: 0.00086
175. slow: 0.00085
176. follow: 0.00084
177. maximum: 0.00083
178. account: 0.00083
179. safety: 0.00083
180. address: 0.00082
181. receive: 0.00081
182. life: 0.00081
183. ready: 0.00081
184. governor: 0.00081
185. set: 0.00080
186. full: 0.00079
187. threat: 0.00078
188. lose: 0.00078
189. include: 0.00078
190. lead: 0.00077
191. declare: 0.00077
192. experience: 0.00076
193. ground: 0.00075
194. learn: 0.00072

Appendix D

Feature Label List

The following is the list of feature labels generated in Section 4.4. Each individual word is an important word identified using TF-IDF. Each pair of words not separated by a comma is an important bigram. Each list of words separated by commas is a synset of important words. Due to a bug in our code, the individual words were treated as synsets. However, that does not affect our later results, since we simply check if any word in the synset is present when building the feature lists for the records and sentences.

1. Hurricane Florence
2. North Carolina
3. storm surge
4. make landfall
5. Myrtle Beach
6. East Coast
7. million people
8. Wilmington N.C.
9. United States
10. National Weather
11. Monday Sept.
12. Rights Reserved
13. mandatory evacuation
14. heavy rain
15. major hurricane
16. Friday morning
17. Thursday night
18. Carolinas Virginia
19. state emergency
20. rise water
21. order evacuate
22. All reserve
23. early week
24. east coast
25. forecast track
26. Tuesday September
27. A link
28. The New
29. President Trump
30. comment
31. work, force, Service
32. show, lead
33. change, turn, affect, live, move
34. weekend
35. change, make, bring
36. Sunday
37. experience, change, reach, find
38. land, reach, hit, work, make, ground
39. change, impact
40. Emergency
41. late
42. information, News
43. local
44. affect, move, approach, hit
45. government
46. content, food
47. big
48. check, watch
49. fall, change

50. change, affect
51. River
52. time, day
53. Washington
54. make, move, change, tell, leave
55. bad
56. continue
57. move, change, set
58. coastal
59. For
60. catastrophic
61. back
62. inland
63. government, State
64. year
65. base, home
66. Email
67. turn, leave, lead
68. content
69. Facebook
70. land, Cape
71. approach
72. change, full
73. damage, leave, move, continue, free, run, change
74. state, provide, power
75. move, reach
76. U.S.
77. Florida
78. people, free
79. part, number, base
80. information, news
81. move, call, post
82. strong
83. change
84. change, make, prepare
85. ahead
86. US
87. part, move, change, turn
88. area, move, order, place
89. day, afternoon
90. threat
91. south
92. Category
93. family
94. No
95. Please
96. number
97. weather
98. Storm
99. line, path
100. move, force
101. area, start, move, open
102. disaster
103. city
104. news, story
105. region, area
106. west
107. region, turn, reach, hit, top
108. due
109. long
110. tropical
111. turn, line, leave, make, begin, move, start
112. experience, live
113. order, declare, call
114. tell
115. day, today
116. rainfall
117. top, line, head
118. Georgia
119. north
120. area, content, place, move, Center
121. Wednesday
122. American
123. line, move, leave, change, part
124. Charleston
125. cover, line
126. news, declare, account
127. include
128. dangerous
129. work, move, make, affect
130. region, County
131. change, receive
132. remain
133. cover, flooding
134. site
135. high

136. content, experience, change
137. move, approach, close
138. change, ready
139. ocean
140. change, move, check
141. information
142. provide
143. eastern
144. change, damage
145. cover, flood
146. potential
147. Atlantic
148. state, declare
149. slow
150. begin
151. safe
152. I
153. effect
154. stay, check
155. information, data
156. South
157. ago
158. region, people, change, bring, land
159. area, content, place, move, center
160. include, cover
161. Saturday
162. As
163. experience, time
164. region
165. weather, move, wind]

Appendix E

Synset Results

The following is the top 50 list of relevant synonyms for important words with the frequency count for the set in parentheses.

- center (16789): area, building, point, content, place, move, center
- place (16650): point, condition, area, move, station, order, estimate, spend, place
- run (16423): damage, leave, direct, move, travel, continue, occur, pass, free, run, change, break
- give (16221): supply, state, direct, produce, pass, release, move, provide, give, change, estimate, show, stretch, lead, offer
- part (14321): thing, location, line, move, leave, change, part
- pass (13614): leave, location, flight, travel, advance, change, move, pass
- wind (13254): weather, travel, move, wind
- fly (13138): hit, travel, move, change, show, fly
- drive (13019): road, travel, move, work, drive, hit, power
- land (12749): region, people, arrive, change, bring, land
- hurricane (12733): hurricane
- open (12659): area, start, move, open
- turn (12650): development, part, move, change, travel, turn
- start (12045): point, turn, line, leave, begin, move, start
- leave (11934): move, produce, change, pass, give, lose, leave
- florence (11162): florence
- move (10820): decision, change, turn, affect, live, move
- hit (10593): contact, affect, kill, move, arrive, approach, hit

- end (9880): point, content, part, state, change, end
- storm (9875): storm
- set (9154): move, change, put, estimate, set
- drop (9147): move, lose, change, fall, drop
- wave (9117): rise, weather, move, wave
- air (9082): gas, region, wind, medium, travel, dry, air
- check (9038): stop, change, move, check
- put (9031): move, change, spend, estimate, put
- world (9025): experience, people, part, group, world
- advance (8788): increase, travel, support, move, set, advance
- carolina (8787): carolina
- direct (8721): order, manage, move, plan, direct
- safety (8709): condition, area, hit, safety
- rush (8626): run, travel, urge, move, effect, rush
- work (8517): energy, work, pass, move, affect, manage
- close (8430): end, move, approach, fill, close
- ship (8342): move, board, travel, put, ship
- strike (8281): read, affect, find, move, strike
- point (8045): location, state, part, end, direction, contact, point
- area (7881): region, area
- top (7509): region, turn, degree, pass, supply, reach, hit, top
- evacuate (7324): move, evacuate
- channel (7314): bring, move, channel
- post (7236): move, record, call, post
- free (7157): people, issue, pass, free
- reach (7104): arrive, move, reach
- north (7043): location, direction, north
- force (6742): organization, move, force
- people (6660): group, family, people

- death (6592): change, state, end, death
- station (6557): facility, move, station
- danger (6533): condition, area, danger
- stretch (6320): increase, move, stretch

Appendix F

Named Entity Extraction

The following is a list of the top 20 most frequent and important named entities that we extracted from a portion of the large data set.

1. Hurricane Florence : 4853
2. Florence : 4685
3. North Carolina : 3979
4. South Carolina : 2506
5. Wilmington : 1532
6. Virginia : 1287
7. North : 1273
8. Hurricane : 1216
9. Carolinas : 1171
10. U.S. : 930
11. USA : 853
12. N.C. : 753
13. New Bern : 704
14. National Hurricane Center : 669
15. East Coast : 644
16. Carolina : 504
17. Cape Fear : 444
18. Bermuda : 428
19. Category : 427
20. United States : 426

Appendix G

Template Summary

The following is the final template summary generated by our team.

Hurricane Florence was a huge storm occurring during the month of September 2018 which peaked as a Category 4 hurricane. The storm was first detected around Sep 9, 2018 when it was known as Tropical Storm Florence and the storm grew in size and ferocity until it become known as a hurricane. As the storm progressed, wind speeds were seen to be ranging from as low as 79 miles per hour to an astounding 139 miles per hour at the peak of the storm. Most areas that were affected by the storm had winds of 109 miles per hour on average when the storm hit them. For rainfall, some areas experienced as much as 32 inches of rain, however other areas on the fringes of the storm only got 5 inches. On average however, most areas affected by the storm had 18 inches of rainfall. During the storm's lifetime, it averaged a diameter of 400 miles across and at its peak reached over 470 miles.

In preparation for the storm's approach, evacuation was determined necessary in South Carolina, Virginia, and North Carolina. About 1.5 million people were evacuated in total over this area and have moved out of the storms primary path. Hurricane Florence made landfall as a Category 1 hurricane on September 12, 2018. The storm landed in both North Carolina and South Carolina. Throughout it's vicious path, Hurricane Florence killed 55 people and caused billions of dollars in damages.

Appendix H

Extractive Summary

The following is the final extractive summary generated by our team using PyTeaser.

Hurricane Florence becomes first major storm of 2018 Atlantic season. The sixth named storm of the 2018 Atlantic hurricane season gathered strength Wednesday to become the first major hurricane of the year. Florence is the third hurricane of the 2018 Atlantic hurricane season, after Hurricane Chris (Category 1) and Hurricane Beryl (Category 2), which formed in early July but didn't make landfall. But riding out Hurricane Florence in areas that are being evacuated is not a good idea, especially in light of all the devastation brought about by Hurricane Katrina in 2005 and Hurricane Harvey and Hurricane Maria in 2017 (Puerto Rico's government recently raised the islands Maria-related death toll to 2,975 people). The first major hurricane of the 2018 Atlantic season, Florence on Wednesday afternoon had maximum sustained winds of 125 mph, according to the National Hurricane Center.

Washington Post declares Trump is complicit for dangerous storm via FOX NEWS. Should we be worried that Hurricane Florence, a Category 4 storm, will hit New York? A million told to flee as Hurricane Florence stalks US East Coast. More than a million people were under evacuation orders in the eastern United States Tuesday, where powerful Hurricane Florence threatened catastrophic damage to a region popular with vacationers and home to crucial government institutions.

Hurricane Florence is now a strong Category 2 hurricane as of 5 a.m. Trump: "We are totally prepared for Hurricane Florence". President Trump just had a meeting with the Department of Homeland Security and Federal Emergency Management Agency officials as Hurricane Florence continues to barrel toward the East Coast. Hurricane Florence could inflict the hardest hurricane punch North Carolina has seen in more than 60 years, with rain and wind of more than 130 mph (209 kph). The National Hurricane Center says Florence is producing tropical storm-force wind gusts in Florence, South Carolina, about 60 miles from the coast.

By Monday morning, the hurricane center classified Florence as a major Category 3 hurricane, with sustained winds of at least 111 mph. Hurricane Florence was a Category 4 hurricane by Monday afternoon and is on track to make landfall in the Carolinas. Blood donors welcome at Englewood hospital in preparation for Hurricane Florence Forty-five members of the Colorado Task Force 1 Urban Search and Rescue Team prepared to meet Hurricane Florence head-on.

Appendix I

Abstractive Summary

The following is the final abstractive summary generated by our team using the Pointer-Generator network.

The National Hurricane Center predicts the storm will make landfall some latest updates. The storm has increased in strength and speed from earlier updates on Monday. Florence will likely hit land as a category 4 hurricane late Thursday evening before weakening again.

More than 1.5 million people have already been ordered to evacuate coastal areas. Models have come into agreement that a Northward turn before reaching the united states is unlikely and that a building high-pressure zone North of the storm will cause it to slow or stall once it reaches the coast or shortly thereafter. Where exactly the zone of heaviest rain will be is a big uncertainty.

His wife Angie Wood said their home was also flooded by the nearby little river after Matthew, but not nearly to the same levels. Flood waters from the cresting rivers inundated the area after the passing of Hurricane Florence. Florence could become an extremely dangerous hurricane sometime Monday. The storm was moving west at 13 mph (20 kmh) and expected to accelerate over the next 36 hours.

Forecasters say the combination of a life-threatening storm surge and the tide will cause normally dry areas near the coast to be flooded by rising waters. North Carolina corrections officials said more than 3,000 people were relocated from adult prisons and juvenile centers in the path of Florence, and more than 300 county prisoners were transferred to state facilities.

The storm was centred 625 miles (1,005 km) southeast of the lesser Antilles and expected to pass south of Puerto Rico, Hispaniola and Cuba, while Hurricane Helene was moving northward away from land. The coastal surge from Florence could leave the eastern tip of North Carolina under more than 9 feet (2.75 metres) of water in spots.

Appendix J

Golden Standard Summary Generated by Team 9

The following is the Golden Standard Summary generated for us by Team 9. The is the reference used when evaluating the summaries in Appendices G, H and I.

The National Hurricane Center identified a potential tropical storm in the eastern Atlantic Ocean with a wind speed of around 30 mph on August 30, 2018. It originated near Cape Verde, off the coast of West Africa. This became a tropical storm named Florence on September 1. It developed into a Category 2 Hurricane on September 4. On September 5, it was promoted to Category 3 hurricane with maximum wind speed around 130 mph. By September 7, it had degraded to a tropical storm. However, it was fueled by 29 degree C sea surface temperatures, so on September 10, it strengthened into a Category 4 hurricane with sustained wind speeds of 140 mph and a central pressure of 939 mbar. On September 12, Florence's peak wind speeds gradually fell to 110 mph, downgrading the hurricane to Category 2. Hurricane Florence finally made landfall on September 14 as a Category 1 hurricane with sustained winds of 80 mph near Wrightsville Beach, North Carolina at 7:15 am local time. The storm was moving at just 6 mph when it landed and had maximum sustained winds near 90 mph. By the end of the day, Florence was downgraded to a tropical storm and was moving slowly west into South Carolina at just 3 mph. The hurricane cut a path through the Eastern United States, particularly North Carolina and South Carolina. Florence weakened as it moved inland, degenerating into a post-tropical cyclone over West Virginia with maximum sustained winds of 25-30 mph on September 17, and dissipating on September 19 over the North Atlantic.

Hurricane Florence was the wettest tropical storm cyclone on record in the Carolinas, causing severe damage due to storm surge and freshwater flooding. Florence caused a maximum of 36 inches of rainfall in Elizabethtown, North Carolina, 34 inches in Swansboro, and 24 inches of rainfall in both Wilmington, NC and Loris, South Carolina. An average of over 20 inches of rainfall was recorded covering 14,000 square miles from Fayetteville, North Carolina, to Florence, South Carolina. Major rivers like Neuse, Eno, Cape Fear, and Lumber spilled over their banks, with inland flooding in Fayetteville, Smithfield, Lumberton, Durham, and Chapel Hill.

The death toll from Hurricane Florence was at least 55. Of them, 30 were a direct consequence of the hurricane. Florence caused immense damage to public and private property. CoreLogic estimated over 600,000 residential homes were damaged by flooding and/or winds. The total cost of the damage was estimated to be between 38 billion and 50 billion dollars.

From 7-12 September, the mayor of Washington, D.C., and the governors of the Carolinas, Virginia, Georgia, and Maryland declared a state of emergency. President Donald Trump declared an emergency in North Carolina, allowing it to access federal funds. Six counties of North Carolina, namely Brunswick, Currituck, Dare, Hyde, New Hanover, and Onslow, were placed under mandatory evacuation on Sept. 10, while Durham, Johnston, Orange, Harnett, and Chatham counties were under high alert and possible evacuation. Franklin Township and the southern part of Sampson County were placed under mandatory evacuation on Thursday, Sept. 13, providing them with much less time for evacuation. Eight counties of South Carolina along the state's 187-mile coastline were placed under mandatory evacuation on Sept. 11. All the roads on I-26 and Route 501 were directed away from the coast. State government offices, including schools and medical facilities, were closed in 26 counties of South Carolina. In Virginia, mandatory evacuations were issued at around 8 a.m. on Sept. 11, for about 245,000 residents in a portion of Hampton Roads and the Eastern Shore area. Evacuations from coastal areas were proceeding normally on Interstate 40, U.S. Highway 70, U.S. Highway 74 and other routes. Around 10 million people resided in the storm's path; around 250,000 people from Virginia, 750,000 people from North Carolina, and 500,000 people from South Carolina were issued evacuation orders. Raleigh officials used reverse-911 to notify its residents in low-lying areas about Florence and activated a non-emergency phone line for residents to get information about effects of Florence and issues that needed to be addressed. All the counties launched a text service to alert their residents and provide them updates about the hurricane. Across the three states, about 3,000 National Guard members and fourteen squads of State Highway Patrol troopers had been activated to assist with hurricane recovery. 126 shelterswere opened across North Carolina and several more across South Carolina and Virginia. Approximately, 10,000, 40,000 and 4000 people were residing inside shelters in North Carolina, South Carolina, and Virginia, respectively.

Duke Energy, one of the major electrical utilities companies, predicted that 3 million people across the Carolinas would be affected by the power outage. More than 600,000 people were cut off from the power supply over the weekend, and 223,000 people were without power for the entire week. Duke had more than 20,000 workers working to restore power across North Carolina and South Carolina. It took more than two weeks, till Sept. 26, to fully restore power. Track vehicles in flooded areas replaced electric poles. Overall, power restoration was aided by over 40,000 workers.

Most of the air traffic operations in the areas affected by Florence were suspended. Many roads like I-40, I-95, US-17, US-70, etc. were closed due to flooding. President Donald Trump signed an emergency declaration for North Carolina, making federal emergency aid available to the state. Insurance payments and federal disaster aid helped in restoration. The Virginia National Guard announced plans to bring up to 1,500 soldiers, airmen, Virginia Defense Force members, and up to 6,000 personnel to aid in response operations of Hurricane Florence. The Diabetes Disaster Response Coalition created several resources to aid people with diabetes in areas affected by Florence.