Generating Text Summaries for the Facebook Data Breach with Prototyping on the 2017 Solar Eclipse

April Fitzpatrick
Akshay Goel
Leah Hamilton
Ramya Nandigam
Esther Robb

CS 4984/CS 5984 - Big Data Text Summarization
Edward A. Fox
Virginia Tech, Blacksburg, VA 24061

December 13, 2018
# Table of Contents

Table of Figures ........................................................................................................ 3
Table of Tables ............................................................................................................ 4
Executive Summary ..................................................................................................... 5
Introduction ................................................................................................................ 7
Literature Review ......................................................................................................... 8
Approach ..................................................................................................................... 10

## Cleaning the Data
- jusText Cleaning and Initial Modifications ............................................................... 11
- jusText Evaluation Results ...................................................................................... 13
- Parameter based cleaning + Similar Document Removal ........................................ 14

## Module 1: Most Frequent Words ........................................................................... 17

## Module 2: WordNet Synsets ................................................................................ 21

## Module 3: Parts of Speech Tagging ...................................................................... 26

## Module 4: Words/Word Stems of Discriminating Features .............................. 28

## Module 5: Frequent and Important Named Entities ............................................ 32

## Module 6: Important Topics.................................................................................. 36
- 6.1 LSA/LDA ........................................................................................................... 36
- 6.2 Clustering .......................................................................................................... 38
- 6.3 Plotting Clusters ............................................................................................... 42
- 6.4 Final Topic Results ........................................................................................... 44

## Module 7: Extractive Summary of Important Sentences .................................... 47
- 7.1 Overall Summary Approach ............................................................................. 47
- 7.2 Clustered Summaries Approach ....................................................................... 49

## Module 8: Values for Each Slot Matching Collection Semantics .................... 52

## Module 9: Readable Summary Explaining Slots and Values .............................. 57

## Module 10: Readable Abstractive Summary ....................................................... 58
- 10.1 Pointer-Generator Network (PGN) ................................................................. 58
- 10.2 FastAbsRL ...................................................................................................... 65
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation</td>
<td>69</td>
</tr>
<tr>
<td>ROUGE Evaluation</td>
<td>69</td>
</tr>
<tr>
<td>Solr Indexing</td>
<td>71</td>
</tr>
<tr>
<td>Gold Standard Summaries</td>
<td>72</td>
</tr>
<tr>
<td>User's Manual</td>
<td>75</td>
</tr>
<tr>
<td>Cleaning Data</td>
<td>75</td>
</tr>
<tr>
<td>Single Word and Entity Based Summary Techniques</td>
<td>77</td>
</tr>
<tr>
<td>Clustering</td>
<td>79</td>
</tr>
<tr>
<td>Complete Event Summaries</td>
<td>80</td>
</tr>
<tr>
<td>Evaluating Gold Standard Summaries</td>
<td>87</td>
</tr>
<tr>
<td>Developer's Manual</td>
<td>88</td>
</tr>
<tr>
<td>Flow of Development</td>
<td>88</td>
</tr>
<tr>
<td>Data Processing</td>
<td>91</td>
</tr>
<tr>
<td>Maintaining and Extending</td>
<td>91</td>
</tr>
<tr>
<td>List of Files Submitted</td>
<td>93</td>
</tr>
<tr>
<td>Lessons Learned</td>
<td>108</td>
</tr>
<tr>
<td>Timeline</td>
<td>108</td>
</tr>
<tr>
<td>Challenges Faced</td>
<td>109</td>
</tr>
<tr>
<td>Solutions Developed</td>
<td>110</td>
</tr>
<tr>
<td>Conclusions and Future Work</td>
<td>112</td>
</tr>
<tr>
<td>Future Work</td>
<td>112</td>
</tr>
<tr>
<td>Conclusions</td>
<td>113</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>115</td>
</tr>
<tr>
<td>References</td>
<td>116</td>
</tr>
<tr>
<td>Appendices</td>
<td>126</td>
</tr>
<tr>
<td>Appendix A: Extractive Summaries</td>
<td>126</td>
</tr>
<tr>
<td>Appendix B: Abstractive Summaries</td>
<td>133</td>
</tr>
<tr>
<td>Appendix C: Gold Standard Summaries</td>
<td>137</td>
</tr>
</tbody>
</table>
Table of Figures

1. Flowchart outlining dataset cleaning workflow .................................................. 11
2. Dataset cleaning parameters ............................................................................. 15
3. List of entity types supported by SpaCy ............................................................ 33
4. Flowchart detailing all clustering techniques used ........................................... 39
5. Results of K-means clustering ........................................................................... 43
6. Number of K-means clusters vs. sum of squared error within cluster .......... 43
7. Different results produced from clustering ......................................................... 45
8. Flowchart of development process ................................................................. 90
9. Original Gantt chart for 10/8-11/16 ................................................................. 109
# Table of Tables

1. Most frequent words in the Solar Eclipse corpus .................................................. 18
2. Most frequent words in the Facebook corpus ....................................................... 19
3. Most frequent WordNet synsets from the Solar Eclipse corpus ......................... 22
4. Most frequent WordNet synsets from the Facebook corpus ............................... 24
5. The most common tagged nouns, verbs, and adjectives from the solar eclipse corpus. Adverbs not included. ................................................................. 26
6. The most common tagged nouns, verbs, and adjectives from the Facebook corpus. Adverbs not included. ................................................................. 27
7. Stemming Results .................................................................................................. 29
8. Lemmatization Results .......................................................................................... 30
9. Named Entity Results ............................................................................................ 35
10. LDA results, 5 topics (based on word frequency) ................................................ 36
11. LSA results (top 5 topics, based on word frequency) ......................................... 37
12. Two Paragraph Summary Clusters ..................................................................... 51
13. A list of planned pieces of information to be extracted from the corpus for the template summary ......................................................................................... 53
14. Example Single Document Abstractive Summaries .......................................... 60
15. A comparison of Cluster 0 and 24 summaries pre-and post-abstraction .......... 63
16. Calculated ROUGE results .................................................................................. 70
17. ROUGE results based on sentences ..................................................................... 70
18. Entity Coverage results ....................................................................................... 71
Executive Summary

As technologies for mass communication have continued to progress, the amount of text available to be read has increased rapidly, requiring new strategies for identifying relevant and important information. Summaries are desirable for events or topics where a large body of information exists, so that the reader does not need to read redundant, irrelevant, or unimportant information. Automated methods can summarize a larger volume of source material in a shorter amount of time, but creating a good summary with these methods remains challenging.

This paper presents the work of a semester-long project in CS 4984/5984 to generate the best possible summary of a collection of 10,829 web pages about the Facebook-Cambridge Analytica data breach, with some early prototyping done on 500 web pages about the 2017 Solar Eclipse.

This report covers implementations and results of natural language processing techniques such as word frequency, lemmatization, and part-of-speech tagging, working up to the creation of a complete human-readable summary. Data cleaning was critical, as text was initially interspersed with boilerplate and HTML, and there were many articles from different sources with identical text. Documents were categorized using k-means clustering on the topics resulting from Latent Semantic Analysis, which was useful both for filtering out irrelevant documents and for ensuring that all relevant topics in the ongoing news coverage were present in the summary even if most documents contained only the initial details. Extractive, abstractive, and combination methods were tested and evaluated for their ability to generate full summaries of the data breach and its consequences.

The summary subjectively evaluated as best was a purely extractive summary built from concatenating summaries of document categories. This method was
coherent and thorough, but involved manual tuning to select categories and still had some redundancy. All attempted methods are described and the less successful summaries are also included. This report presents a framework for how to summarize complex document collections with multiple relevant topics. The summary itself identifies information which was most covered about the Facebook-Cambridge Analytica data breach and is a reasonable introduction to the topic.
Introduction

This paper documents our work for the Fall 2018 section of CS 4984/5984: Big Data Text Summarization. The objective of the class was to take the text of a large number of articles scraped from the Internet pertaining to a specific event and summarize them in a human-readable format. The first few techniques (specified in detail in the modules below) were prototyped on a dataset of 500 articles about the 2017 solar eclipse, with later work being done on articles about the Facebook data breach. For the Facebook data set, a subset of 500 articles was again used for prototyping, with final results being from the entire set of 10829 articles.

PySpark was used to interface with a Hadoop cluster for some units, but with the 2922 articles remaining after removing duplicate or empty pages and cleaning documents extensively, it took a reasonable amount of time to run the Python code for most units in parallel on a single machine.

The project began with some smaller tasks such as finding the most frequent words, part of speech tagging, and lemmatization. These tasks were useful for familiarizing us with natural language processing and the content of our dataset without getting overwhelmed by the huge task in front of us.

In the end, we created full summaries using various methods: extractive, abstractive, and template based. All of these approaches are described and most yielded decent results, with the extractive summary done on clustered topics being our best summary. The cluster-based extractive summary consists of two 300-word paragraphs with each paragraph covering a different set of clusters. The first paragraph focuses on general information about the event and the second paragraph is about the effects of the data breach. Although the result is a little choppy due to it being a concatenation of sentences from different articles, it gives a fairly good and broad overview of the entire event.
Literature Review

The Natural Language Toolkit (NLTK) textbook provided a great starting point for all of the early modules in this project. [1] gave insight into creating frequency distributions, doing part of speech tagging with lemmatization and stemming, and creating WordNet synsets [2]. For anyone who needs to understand the basics of natural language processing with Python, this book is essential.

The method pioneered by Chen and Bansal [3, 4] uses a combination of neural networks and reinforcement learning to create abstractive summaries from single documents. They first select the most relevant sentences extractively using one neural network, then rewrite those sentences abstractively using a second neural network. To bridge the two networks, they use reinforcement learning, with a sentence-level reward policy based on ROGUE score. The ‘agent’ being trained by the reinforcement learning is the extractive summarizer. This model replaced the pointer-generator model [5, 6], which flips between extracting text and writing novel text during the decoding step, as the new state-of-the art.

[7] uses a variety of approaches to extract named entities from an article. The first approach is ne_chunk, which classifies named entities based on three labels: PERSON, ORGANIZATION, and GPE (geopolitical entity). NLTK's ne_chunk requires that the input be a list of pos-tagged (part-of-speech tagged) words. The other approach Li uses is SpaCy [8], which classifies named entities based on multiple labels, including PERSON, ORGANIZATION, LOCATION, LANGUAGE, DATE, and TIME. SpaCy does not require a pos-tagged input, as it interprets the grammar of the sentence as part of its processing pipeline.

The method pioneered by Banerjee, Mitra, and Sugiyama [9,10] uses a four step approach to building an abstractive summary. As described in the paper, the first step is to identify the most important document inside the provided corpus.
Following that, clusters of similar sentences are generated by aligning sentences to the ones in the most important documents. Third, K-shortest paths from each sentences in the clusters are generated using a word-graph structure. Finally, sentences are selected from this generated set of shortest paths using an integer linear programming model while maximizing readability and content.
Our approach is broken into each sub-task that we did in building our final text summaries. This included doing some preprocessing, like cleaning the data set first. After cleaning the data (and re-cleaning it as we found new bad data during our work), we set out to complete each of the ten tasks assigned to us. A lot of the steps were able to be done concurrently, with each team member taking the lead on select tasks.
Cleaning the Data

In both the solar eclipse and Facebook breach datasets, there were a number of unusable records that had to be filtered out. We also found many seemingly valid records that just contained JavaScript or navigation items, rendering it to be garbage. The flowchart in Figure 1 shows the data flow process.

![Flowchart](image)

Figure 1. Flowchart outlining dataset cleaning workflow

**jusText Cleaning and Initial Modifications**

First, the script `ArchiveSpark_WarcToJson.scala` was used to extract all text and HTML within the WARC file. It also used the page metadata to keep only the most recent version of any pages which were scraped more than once (based on URL) and then exported the data as a .json file. Earlier versions of this script kept only the text within the `<body>` tag, and then all text and HTML within the `<body>` tag. By keeping more HTML later into the cleaning process, the files were larger and took longer to process, but the tags could be used for processing the text and information could be scraped out of the header.
Then, the script *jusTextCleaningBig.py* was run. The goal of this script was to extract readable text bodies from each document’s HTML using jusText [11,12] and then remove records that had garbage/unusable/empty results.

First, *jusTextCleaningBig.py* runs jusText on each entry. jusText is a Python package that parses the HTML and then uses a heuristic algorithm to decide whether each `<p>` tag is relevant text or boilerplate. The main metrics used are whether a paragraph is grammatical English text and the density of links in the paragraph. By running jusText on every record, the dataset which was left contained human readable English text from all our documents.

The file *jusTextCleaningBig.py* also uses the Python package Beautiful Soup [13] to scrape a best-guess publication date out of the <meta> tags in each article, which was an early plan for classifying articles, but it was not possible to get correct publication dates for all articles using this method.

At this point, there were still a significant number of documents with ‘bad’ information, such as completely empty pages, 404s, etc. *jusTextCleaningBig.py* uses the `.contains()` function to remove entries whose titles contained words and phrases like “forbidden”, “error”, or “access denied”, as we had multiple entries with little or no text that were due to errors accessing the page. The exact list of phrases used to filter bad documents is given below.

- Forbidden
- Page not found
- 404
- 403
- 408
- ERROR
- Access Denied
We observed that all documents with titles containing one or more of these phrases did not contain any useful text.

We observed that there were a good chunk of records with absolutely no text in them after parsing. jusText had completely filtered out the text from some documents, such as those that just redirected you to a different location or those that were in a language other than English. We ran a quick filter query to remove all these documents that were left with no text in them.

Following this process, once we were down to about 6285 documents, we realized that all our data was encoded in ASCII which was causing some issues. Some of our text such as Å, å, 0, ū was appearing as hex characters such as /xc/xf and influencing our results. To get rid of these we had to decode the ASCII and re-encode in UTF-8. The code to do this can be found in remove_hex_characters.py.

**jusText Evaluation Results**

Since jusText is an older external library, we were naturally slightly skeptical about its performance and parsing quality. To make sure that jusText did not remove any important documents, we ran some analysis and evaluated the results on both the small and big datasets.

The smaller dataset contained 475 documents, and when using jusText to clean them, all text was removed from 154 documents, leaving empty strings. This accounts to removing 32% of the small dataset. Since the removal of this much of the dataset was initially a concern, we confirmed that all 154 documents were indeed of no value to us.

- 54 / 154 deleted documents consisted only of inaccessible pages such as 404, error pages, page not found, forbidden, 403, etc. (35%)
- 55 / 154 deleted documents were in a different language. (36%)

13
- 45 / 154 deleted documents were pages that ended up redirecting you to different locations resulting in the HTML body not containing anything relevant. (29%)

The large dataset had 2398 documents deleted. Although this seems like a lot, the deleted documents were similar to the small dataset.

- 994 / 2398 documents were in a foreign language. (41%)
- 1404 / 2398 documents contained video links, broken pages, 404, 403, page not founds, redirection links, etc. (59%)

Conclusion: Almost 100% of the deleted documents were indeed bad, and jusText performed very well. This does not guarantee that all bad documents from the entire dataset were deleted; it only guarantees that no good documents were deleted.

Parameter based cleaning + Similar Document Removal

Once all the steps before this stage have been completed, we then sent our dataset through furthur_cleaning.py. The job of this script is to remove documents that classify as ‘bad’ documents based on whether they pass or fail some conditions. The input to this file is expected to be a .json file containing a series of JSON objects with the following fields: {title, originalurl, text}. Note that text is a large string in the case it is a single huge document. Since we also support functionality to perform evaluations based on paragraphs, text can also be an array of strings where each entry represents a paragraph. The code handles both inputs correctly.

These were the filters based on which we perform cleaning. These filters can be seen in Figure 2. Note that these conditions were generated over time as we found bad documents in our dataset.
Our final step was removing similar documents.

While working with our dataset and creating preliminary summaries, we observed that our text contained many duplicate sentences. It did not take us long to realize that there are a huge number of documents that are exact copies of one another, or at the very least have borrowed most of their text from other
articles. This was leading to a lot of duplicate sentences in our corpus and negatively influencing both document classification and summarization.

Additionally, we also noticed that there were many documents with the exact same title, and as expected, they also contained the exact same text.

To tackle this, we wrote a script called remove_similar_documents.py that compares every pair of documents and deletes the smaller document if their similarity exceeds a provided threshold. To check for similarity between two documents, the algorithm takes the intersection of the set of all unique sentences in both documents and compares it to the smaller document.

∀ document $D_i$, $D_j$ where $\text{len}(D_j) < \text{len}(D_i)$ :

Let $S_k$ be the set of unique sentences for document $D_k$

$\text{Similarity}(i, j) = \frac{\text{length}(S_i \cap S_j)}{\text{length}(S_j)}$

The algorithm to remove all document pairs with a similarity score greater than the threshold will do the following:

1. Sort all documents in descending order of size
2. Get similarity score for all document pairs $(i,j)$ where $j > i$
3. If similarity is above threshold, delete document $j$

This algorithm executes in $O(n^2)$ time in the worst case. When there is a large proportion of similar or duplicate documents, as in our corpus, the true runtime is far less than $O(n^2)$. Since a large number of documents get deleted and the running space gets pruned very rapidly, $n$ comparisons are knocked out with every removal.

The best threshold value we found was 0.4, meaning that if two documents are more than 40% similar, then we delete the smaller document. At the end of this process, we were left with 2922 documents! Although this is fairly small,
compared to what we started off with, our summaries and topics came out to be much better, as we will see in the following sections.

**Module 1: Most Frequent Words**

The goal of this module was to get a list of most common important words based on their cumulative frequency in all records in the dataset. If raw word counts were used, stopwords such as “and” and “the” made up the majority of the most frequent words. To correct for this, we removed all words in the NLTK stopword list and added our own, which was mostly comprised of punctuation marks and some additional words like “would”, “though”, and “upon”.

In later tests, due to a large density of non-standard punctuation being inserted at some point in the HTML-parsing workflow, we largely switched from using a set list of punctuation to only keeping words at least 3 letters long. You can see in the Solar Eclipse data (Table 1) that there are still several punctuation marks and other artifacts being counted as words, which ultimately had to be filtered out by only allowing ASCII characters in `remove_hex_characters.py`, as described in the Cleaning the Data section of the report. This is because characters such as â are represented by multiple characters in UTF-8, and in Python 2.7, the `len()` function counts them as longer strings. Tables 1 and 2 show our results for this module.
Table 1. Most frequent words in the Solar Eclipse corpus

<table>
<thead>
<tr>
<th>Word</th>
<th>Frequency</th>
<th>Word</th>
<th>Frequency</th>
<th>Word</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>eclipse</td>
<td>5374</td>
<td>see</td>
<td>300</td>
<td>event</td>
<td>151</td>
</tr>
<tr>
<td>solar</td>
<td>2222</td>
<td>one</td>
<td>294</td>
<td>nasa</td>
<td>150</td>
</tr>
<tr>
<td>sun</td>
<td>1249</td>
<td>â</td>
<td>266</td>
<td>view</td>
<td>144</td>
</tr>
<tr>
<td>total</td>
<td>967</td>
<td>2017</td>
<td>220</td>
<td>city</td>
<td>140</td>
</tr>
<tr>
<td>totality</td>
<td>639</td>
<td>oregon</td>
<td>206</td>
<td>earth</td>
<td>137</td>
</tr>
<tr>
<td>moon</td>
<td>597</td>
<td>viewing</td>
<td>189</td>
<td>park</td>
<td>133</td>
</tr>
<tr>
<td>aug.</td>
<td>595</td>
<td>â</td>
<td>187</td>
<td>minutes</td>
<td>126</td>
</tr>
<tr>
<td>glasses</td>
<td>499</td>
<td>via</td>
<td>176</td>
<td>first</td>
<td>120</td>
</tr>
<tr>
<td>people</td>
<td>367</td>
<td>images</td>
<td>173</td>
<td>near</td>
<td>116</td>
</tr>
<tr>
<td>path</td>
<td>330</td>
<td>time</td>
<td>162</td>
<td>state</td>
<td>116</td>
</tr>
<tr>
<td>2017</td>
<td>319</td>
<td>eclipses</td>
<td>159</td>
<td>says</td>
<td>107</td>
</tr>
</tbody>
</table>
Table 2. Most frequent words in the Facebook corpus

<table>
<thead>
<tr>
<th>Word</th>
<th>Frequency</th>
<th>Word</th>
<th>Frequency</th>
<th>Word</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>facebook</td>
<td>29192</td>
<td>social</td>
<td>1954</td>
<td>could</td>
<td>1144</td>
</tr>
<tr>
<td>data</td>
<td>20389</td>
<td>one</td>
<td>1908</td>
<td>apps</td>
<td>1094</td>
</tr>
<tr>
<td>cambridge</td>
<td>7215</td>
<td>zuckerberg</td>
<td>1887</td>
<td>used</td>
<td>1075</td>
</tr>
<tr>
<td>analytica</td>
<td>5718</td>
<td>n't</td>
<td>1851</td>
<td>time</td>
<td>1074</td>
</tr>
<tr>
<td>information</td>
<td>4772</td>
<td>app</td>
<td>1770</td>
<td>google</td>
<td>1074</td>
</tr>
<tr>
<td>users</td>
<td>4234</td>
<td>also</td>
<td>1687</td>
<td>companies</td>
<td>985</td>
</tr>
<tr>
<td>people</td>
<td>3710</td>
<td>media</td>
<td>1639</td>
<td>ads</td>
<td>920</td>
</tr>
<tr>
<td>company</td>
<td>3346</td>
<td>new</td>
<td>1525</td>
<td>election</td>
<td>910</td>
</tr>
<tr>
<td>privacy</td>
<td>2879</td>
<td>use</td>
<td>1507</td>
<td>wylie</td>
<td>895</td>
</tr>
<tr>
<td>campaign</td>
<td>2750</td>
<td>news</td>
<td>1494</td>
<td>firm</td>
<td>837</td>
</tr>
<tr>
<td>trump</td>
<td>2652</td>
<td>political</td>
<td>1474</td>
<td>kogan</td>
<td>804</td>
</tr>
</tbody>
</table>

Both of these results have some noise and some words that show up in multiple forms (Facebook and Facebook’s, total and totality, viewing and view, eclipse and eclipses), as well as a number of synonyms. But you can start to get an impression of what each dataset is generally about. The eclipse dataset has a lot of words referring to viewing, eclipse glasses, and astrological bodies. The Facebook dataset has more words related to people and entities involved (Cambridge Analytica, Facebook, Mark Zuckerberg, etc.) and words referring to privacy and the US election. These words are mainly helpful as a topical overview and a beginning idea for which kinds of words are most important in each dataset.
The most recent code for this module is in *Unit1.py*, and returns the counts both by printing to console and writing a CSV file.
Module 2: WordNet Synsets

The goal of this module was to use WordNet to combine frequency counts for synonyms. WordNet is a database that relates a large number of English words as synonyms [2]. It also has relationships such as hypernyms and hyponyms (“car” is a hypernym of “SUV”, and “SUV” is a hyponym of “car”), but we decided not to use these relationships mostly because our most frequent words in both the eclipse and Facebook datasets didn’t have any obvious hyponym or hypernym relationships. Additionally, other teams found that converting to hypernyms generally made their frequency data too general without collapsing any duplication of meaning.

In WordNet, each English word can have multiple “senses” for its different definitions, each of which is tied to a specific part of speech (POS). This means that each word had to be tagged as either a verb, noun, adverb, or adjective (the only parts of speech included in WordNet), and then the word sense had to be disambiguated (WSD) from all possible senses for that word based on context.

The algorithm we used for this task was the Lesk algorithm, based off the work of Michael Lesk [14]. It assigns a sense to a word such as “drew” in a sentence such as “He drew his bow” by maximizing the overlap between the words present in the dictionary definition of a given sense of “drew” and the words present in the dictionary definitions of the other words in the sentence (the relevant definitions of “bow” and “drew” might both mention a weapon, for instance).

The original prototype only included POS-tagging, but after developing the lemmatizer for module 4, we went back and modified the code to lemmatize the words before WSD. The results shown below, in Table 3 and Table 4, lemmatize the text before WSD.
Table 3. Most frequent WordNet synsets from the Solar Eclipse corpus

<table>
<thead>
<tr>
<th>Synset</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>['eclipse', 'occultation']</td>
<td>5784</td>
</tr>
<tr>
<td>['solar']</td>
<td>2563</td>
</tr>
<tr>
<td>['sunlight', 'sunshine', 'sun']</td>
<td>1700</td>
</tr>
<tr>
<td>['witness', 'find', 'see']</td>
<td>1407</td>
</tr>
<tr>
<td>['suppose', 'say']</td>
<td>1281</td>
</tr>
<tr>
<td>['moonlight', 'moonshine', 'Moon']</td>
<td>1038</td>
</tr>
<tr>
<td>['totality']</td>
<td>942</td>
</tr>
<tr>
<td>['methamphetamine', 'methamphetamine_hydrochloride', 'Methedrine', 'meth', 'deoxyephedrine', 'chalk', 'chicken_feed', 'crank', 'glass', 'ice', 'shabu', 'trash']</td>
<td>783</td>
</tr>
<tr>
<td>['time']</td>
<td>730</td>
</tr>
<tr>
<td>['people']</td>
<td>700</td>
</tr>
<tr>
<td>['twenty-one', '21', 'xxi']</td>
<td>655</td>
</tr>
<tr>
<td>['way', 'path', 'way_of_life']</td>
<td>648</td>
</tr>
<tr>
<td>['watch', 'view', 'see', 'catch', 'take_in']</td>
<td>627</td>
</tr>
<tr>
<td>['one']</td>
<td>603</td>
</tr>
<tr>
<td>['event']</td>
<td>576</td>
</tr>
<tr>
<td>['suffer', 'sustain', 'have', 'get']</td>
<td>554</td>
</tr>
</tbody>
</table>
The results are similar to the results from module 1 for this particular dataset, although the word “total” (the occurrences of which were split across multiple synsets) fell significantly below other words which existed in the original text in more diverse word forms. There are far more verbs (watch, search/look, travel, say, see) higher up on the frequency list of synsets here, probably largely because verbs tend to exist in more forms.

Another interesting thing to note is the high frequency of the word sense attributed to “methamphetamine”. This was not the case before lemmatization, and it seems that the lemmatizer changed every instance of the word “glasses” referring to the eclipse glasses into “glass”, which WordNet correctly recognizes as a different word with different meanings. Other issues seem more straightforward. The “state of matter” synset more likely refers to the United States, and the “way of life” synset more likely refers to the path of totality. We suspect that the short definitions in the WordNet dictionary made WSD for some of these more abstract words with many senses difficult.
Table 4. Most frequent WordNet synsets from the Facebook corpus

<table>
<thead>
<tr>
<th>Synset</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>['datum', 'data_point']</td>
<td>23244</td>
</tr>
<tr>
<td>['suppose', 'say']</td>
<td>13439</td>
</tr>
<tr>
<td>['ship's_company', 'company']</td>
<td>10329</td>
</tr>
<tr>
<td>['user']</td>
<td>9715</td>
</tr>
<tr>
<td>['Cambridge_University', 'Cambridge']</td>
<td>9579</td>
</tr>
<tr>
<td>['use']</td>
<td>9294</td>
</tr>
<tr>
<td>['information', u'selective_information', 'entropy']</td>
<td>8530</td>
</tr>
<tr>
<td>['people']</td>
<td>7406</td>
</tr>
<tr>
<td>['witness', 'find', 'see']</td>
<td>6060</td>
</tr>
<tr>
<td>['political_campaign', 'campaign', 'run']</td>
<td>5724</td>
</tr>
<tr>
<td>['take', 'make']</td>
<td>5566</td>
</tr>
<tr>
<td>['privacy', 'privateness', 'secrecy', 'concealment']</td>
<td>5460</td>
</tr>
<tr>
<td>['besides', 'too', 'also', 'likewise', 'as_well']</td>
<td>5304</td>
</tr>
<tr>
<td>['social']</td>
<td>5297</td>
</tr>
<tr>
<td>['time']</td>
<td>5132</td>
</tr>
<tr>
<td>['trump']</td>
<td>4609</td>
</tr>
<tr>
<td>['metier', 'medium']</td>
<td>4573</td>
</tr>
</tbody>
</table>
The only big differences in these results versus the non-lemmatized version are that the counts of most of the synsets increased and “trump” showed up much higher on this list.

WordNet synsets are not a particularly good tool for looking at text about the Facebook data breach. It drops pretty much all proper nouns like “Facebook” and “Zuckerberg”, misinterprets the proper nouns it keeps such as “Cambridge” as referring to the university and not Cambridge Analytica, and also misinterprets many technically-related words. Seen above, “company” is interpreted as being related to ships, and lower down on the list are instances where “platform” is interpreted as being related to weapons and “profile” is also misinterpreted.

The final code for this unit is found in Unit2Lemmas.py. An earlier version of the code which uses part-of-speech tagging but does not lemmatize is found in Unit2Prototype.py.
Module 3: Parts of Speech Tagging

The goal of this module was to use part of speech tagging to improve the frequency counts for important words. To accomplish this, we used NLTK’s in-built pos_tag() tool, which is an n-gram tagger that takes each clause (string of words separated by punctuation such as a period or comma) and classifies each word based on the context of the sentence. It is a pre-trained machine learning model.

While we determined that certain parts of speech were more interesting for the different datasets, this module was mostly useful to assist the accuracy of other algorithms, like stemming and word sense disambiguation.

Table 5. The most common tagged nouns, verbs, and adjectives from the solar eclipse corpus. Adverbs not included.

<table>
<thead>
<tr>
<th>Nouns</th>
<th>Frequency</th>
<th>Verbs</th>
<th>Frequency</th>
<th>Adjectives</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>eclipse</td>
<td>5373</td>
<td>said</td>
<td>949</td>
<td>solar</td>
<td>2563</td>
</tr>
<tr>
<td>sun</td>
<td>1637</td>
<td>see</td>
<td>713</td>
<td>total</td>
<td>1364</td>
</tr>
<tr>
<td>moon</td>
<td>1001</td>
<td>get</td>
<td>388</td>
<td>first</td>
<td>373</td>
</tr>
<tr>
<td>totality</td>
<td>942</td>
<td>watch</td>
<td>323</td>
<td>partial</td>
<td>312</td>
</tr>
<tr>
<td>glasses</td>
<td>726</td>
<td>viewing</td>
<td>306</td>
<td>american</td>
<td>272</td>
</tr>
</tbody>
</table>
Table 6. The most common tagged nouns, verbs, and adjectives from the Facebook corpus. Adverbs not included.

<table>
<thead>
<tr>
<th>Nouns</th>
<th>Frequency</th>
<th>Verbs</th>
<th>Frequency</th>
<th>Adjectives</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>facebook</td>
<td>28434</td>
<td>said</td>
<td>9335</td>
<td>social</td>
<td>5297</td>
</tr>
<tr>
<td>data</td>
<td>23231</td>
<td>used</td>
<td>3931</td>
<td>new</td>
<td>4481</td>
</tr>
<tr>
<td>cambridge</td>
<td>9524</td>
<td>facebook</td>
<td>2753</td>
<td>political</td>
<td>3633</td>
</tr>
<tr>
<td>information</td>
<td>8529</td>
<td>use</td>
<td>2642</td>
<td>personal</td>
<td>3142</td>
</tr>
<tr>
<td>users</td>
<td>8080</td>
<td>make</td>
<td>2482</td>
<td>many</td>
<td>2135</td>
</tr>
</tbody>
</table>

As can be seen in Tables 5 and 6, the verbs in the solar eclipse dataset were all related to seeing, and are thus more specific to the corpus than those from the Facebook set. The nouns are largely the most characteristic of the corpora, however. Aside from adverbs in the solar eclipse set like “totally” and “completely”, the adverbs do not offer much insight.

The code for this unit is found in Unit3.py.
Module 4: Words/Word Stems of Discriminating Features

The goal of this module was to improve the most frequent word count by either lemmatizing or stemming the words before calculating their frequencies. We decided to try both lemmatization and stemming, to see which results were more relevant.

For lemmatization, we used NLTK’s WordNetLemmatizer. This was a fairly straightforward process. We were able to reuse a lot of the most frequent word code, where we just lemmatized each word before counting the frequencies. All of this code can be found in Unit4_Lemmas.py.

For stemming, we tried two different NLTK-supported stemmers, Porter and Snowball. After some research, we decided to use Snowball with an English dictionary instead of Porter, because it is newer and it is widely held to be better than the Porter stemmer [1]. We briefly looked at using the Paice/Husk stemmer, but it is too aggressive of a stemmer to be useful to us; it “overstems,” meaning it cuts words down too much to the point where the sense of many words was lost, which is not good for automatically generating a summary [15]. The stemming process was the same as the lemmatization process, just stemming each word before counting it instead of lemmatizing (and can be found in Unit4_Stems.py). Results from both can be found in Tables 7 and 8.
Table 7. Stemming Results

<table>
<thead>
<tr>
<th>Word</th>
<th>Frequency</th>
<th>Word</th>
<th>Frequency</th>
<th>Word</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>facebook</td>
<td>32043</td>
<td>social</td>
<td>2012</td>
<td>call</td>
<td>1201</td>
</tr>
<tr>
<td>data</td>
<td>20419</td>
<td>polit</td>
<td>1966</td>
<td>googl</td>
<td>1191</td>
</tr>
<tr>
<td>cambridg</td>
<td>7285</td>
<td>work</td>
<td>1958</td>
<td>firm</td>
<td>1164</td>
</tr>
<tr>
<td>use</td>
<td>7143</td>
<td>time</td>
<td>1898</td>
<td>year</td>
<td>1155</td>
</tr>
<tr>
<td>user</td>
<td>6950</td>
<td>n't</td>
<td>1851</td>
<td>post</td>
<td>1152</td>
</tr>
<tr>
<td>compani</td>
<td>6569</td>
<td>advertis</td>
<td>1706</td>
<td>could</td>
<td>1144</td>
</tr>
<tr>
<td>analytica</td>
<td>6175</td>
<td>also</td>
<td>1687</td>
<td>friend</td>
<td>1141</td>
</tr>
<tr>
<td>inform</td>
<td>5272</td>
<td>elect</td>
<td>1663</td>
<td>set</td>
<td>1087</td>
</tr>
<tr>
<td>peopl</td>
<td>4083</td>
<td>media</td>
<td>1656</td>
<td>get</td>
<td>1084</td>
</tr>
<tr>
<td>campaign</td>
<td>3757</td>
<td>report</td>
<td>1636</td>
<td>wyli</td>
<td>1018</td>
</tr>
<tr>
<td>app</td>
<td>3749</td>
<td>new</td>
<td>1526</td>
<td>kogan</td>
<td>996</td>
</tr>
<tr>
<td>trump</td>
<td>3267</td>
<td>news</td>
<td>1494</td>
<td>access</td>
<td>995</td>
</tr>
<tr>
<td>privaci</td>
<td>2884</td>
<td>make</td>
<td>1358</td>
<td>million</td>
<td>920</td>
</tr>
<tr>
<td>zuckerberg</td>
<td>2154</td>
<td>ani</td>
<td>1305</td>
<td>want</td>
<td>913</td>
</tr>
<tr>
<td>person</td>
<td>2072</td>
<td>account</td>
<td>1246</td>
<td>collect</td>
<td>906</td>
</tr>
<tr>
<td>one</td>
<td>2066</td>
<td>share</td>
<td>1224</td>
<td>target</td>
<td>890</td>
</tr>
</tbody>
</table>

The order of the most frequent words after stemming did not change much; however, the counts of each word went up quite a bit. This is due to the fact that words like “company” and “companies” are being combined into one word count.
of “compani.” This should give us a better representation of word frequency, but poses an issue because it stems words down to non-English words (e.g., “compani”).

Table 8. Lemmatization Results

<table>
<thead>
<tr>
<th>Word</th>
<th>Frequency</th>
<th>Word</th>
<th>Frequency</th>
<th>Word</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>facebook</td>
<td>28328</td>
<td>use</td>
<td>1771</td>
<td>kogan</td>
<td>835</td>
</tr>
<tr>
<td>data</td>
<td>20757</td>
<td>zuckerberg</td>
<td>1723</td>
<td>account</td>
<td>830</td>
</tr>
<tr>
<td>cambridge</td>
<td>7044</td>
<td>new</td>
<td>1602</td>
<td>personal</td>
<td>825</td>
</tr>
<tr>
<td>analytica</td>
<td>5598</td>
<td>news</td>
<td>1588</td>
<td>firm</td>
<td>811</td>
</tr>
<tr>
<td>information</td>
<td>5303</td>
<td>political</td>
<td>1552</td>
<td>user</td>
<td>810</td>
</tr>
<tr>
<td>people</td>
<td>3972</td>
<td>media</td>
<td>1546</td>
<td>facebooks</td>
<td>797</td>
</tr>
<tr>
<td>users</td>
<td>3795</td>
<td>time</td>
<td>1181</td>
<td>friends</td>
<td>784</td>
</tr>
<tr>
<td>company</td>
<td>3659</td>
<td>apps</td>
<td>1150</td>
<td>...</td>
<td>722</td>
</tr>
<tr>
<td>campaign</td>
<td>2910</td>
<td>could</td>
<td>1115</td>
<td>may</td>
<td>717</td>
</tr>
<tr>
<td>privacy</td>
<td>2885</td>
<td>google</td>
<td>1093</td>
<td>get</td>
<td>710</td>
</tr>
<tr>
<td>trump</td>
<td>2669</td>
<td>election</td>
<td>1060</td>
<td>online</td>
<td>700</td>
</tr>
<tr>
<td>social</td>
<td>2028</td>
<td>used</td>
<td>1004</td>
<td>says</td>
<td>695</td>
</tr>
<tr>
<td>n't</td>
<td>1928</td>
<td>companies</td>
<td>952</td>
<td>see</td>
<td>691</td>
</tr>
<tr>
<td>app</td>
<td>1889</td>
<td>wylie</td>
<td>896</td>
<td>digital</td>
<td>682</td>
</tr>
<tr>
<td>one</td>
<td>1879</td>
<td>access</td>
<td>896</td>
<td>work</td>
<td>663</td>
</tr>
<tr>
<td>also</td>
<td>1775</td>
<td>ads</td>
<td>880</td>
<td>make</td>
<td>654</td>
</tr>
</tbody>
</table>
The lemmatization shows very similar results to those obtained with stemming. Many of the first few words show up with higher frequencies because they have been combined with their lemmas. The big difference here between the stemming and lemmatization is that lemmatization did not cut off words into non-English stems. So although the lemmatization counts are not as high for some words as the stemmed counts, since the words are not broken down quite as far, they stay real words.

Thinking of our future plans for using this data, we decided that we should use the lemmatized version of the most frequent words. Even though the stemmed words grouped together a few of the words better, the fact that it turns a good amount of words into non-English words (e.g., “people” into “peopl”, “cambridge” into “cambridg”) was a non-starter. The most likely use for this information is to fill the slots in our template summary. We definitely want to have English words for this use, without having to do extra processing to turn the stemmed words back into complete words.
Module 5: Frequent and Important Named Entities

The goal of this module was to extract the frequent and important named entities. A named entity is anything that can be denoted with a proper name. They can include persons, locations, amounts of money, organizations, etc. Extracting these named entities is an important factor in text summarization as, for example, we can see the important persons involved in the topic of interest.

We considered many different approaches in determining the most appropriate method to accomplish this module. The approaches we considered using are NLTK’s ne_chunk and SpaCy. NLTK’s ne_chunk tool takes a list of words that have been pos-tagged and returns a list of words with their corresponding classifier, or named entity type [16]. SpaCy, on the other hand, requires a sentence or document as an input and returns the list of words with their corresponding classifier just like NLTK ne_chunk does [17]. Since SpaCy doesn’t require a pos-tagged input, we decided to proceed with our named entity recognition using SpaCy. The list of named entity types supported by SpaCy is shown in Figure 3.
SpaCy has another in-built feature which gives the frequency of each type of named entity. This is important when trying to get a good picture of the text that is being summarized. SpaCy overall is a very convenient tool to use for NER as it can tokenize, tag parts of speech, and then locate named entities with a single function call. One of the challenges with using SpaCy, however, is that sometimes the same word can be classified as two different types of named entities. In our dataset, the word “trump” was classified as both PERSON (which makes sense given the articles that mention President Trump in our dataset) and, oddly
As enough, as *NORP*, which is the classifier given to nationalities or political or religious groups. Another challenge with SpaCy is that its extraction of entities depends on capitalization, more than we originally thought. Certain entities weren’t extracted because they weren’t capitalized in the dataset. SpaCy also has other limitations. The word “elizabeth” showed up as an *ORDINAL* entity. It’s possible that SpaCy classifies any word that ends in “th” as *ORDINAL* since that’s how other ordinal entities such as “fourth” and “fifth” end. There are also some phrases in the *PRODUCT* which don’t make sense, such as “assets/180427113024-24-koreas.” In fact the only thing that could be considered a product that was classified as a *PRODUCT* is “bmw.” Figuring out how to deal with the limitations of SpaCy could be a future direction to look into so that our results will make more sense and be more relevant to our dataset.

Table 9 shows the most frequent named entities for various entities. Final code for this unit can be found in *Unit5.py*. 
### Table 9. Named Entity Results

<table>
<thead>
<tr>
<th>ORDINAL</th>
<th>[(first', 3874), ('third', 2858), ('second', 764), ('3rd', 142), ('', 126), ('elizabeth', 66), ('fourth', 66), ('fifth', 64), ('2nd', 32), ('firstly', 26)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRODUCT</td>
<td>[('morning', 16), ('592', 12), ('assets/180427113024-24-koreas', 8), ('twtr.n', 6), ('d.c', 6), ('bmw', 4), ('43/50 7', 4), ('1,730/$1,750/au$2,400', 4), ('7:05pm', 2), ('&amp;', 2)]</td>
</tr>
<tr>
<td>TIME</td>
<td>[('morning', 186), ('this morning', 122), ('night', 114), ('hours', 98), ('60 minutes', 78), ('friday night', 46), ('afternoon', 46), ('last night', 44), ('overnight', 44), ('minutes', 42)]</td>
</tr>
<tr>
<td>MONEY</td>
<td>[('$15 million', 94), ('billions of dollars', 76), ('40,000', 68), ('1', 64), ('100,000', 46), ('millions of dollars', 46), ('$10 billion', 42), ('2', 38), ('$5 million', 34), ('$1 billion', 34)]</td>
</tr>
<tr>
<td>DATE</td>
<td>[('2016', 4386), ('monday', 1820), ('today', 1742), ('2015', 1528), ('2014', 1520), ('last week', 1026), ('friday', 1000), ('tuesday', 938), ('2012', 800), ('sunday', 796)]</td>
</tr>
<tr>
<td>QUANTITY</td>
<td>[('a ton', 88), ('as many as 50 million', 52), ('50m', 14), ('tons', 12), ('four-metre', 12), ('two', 10), ('a few feet', 10), ('elon', 8), ('625,000 pounds', 8), ('a tonne', 6)]</td>
</tr>
<tr>
<td>LOC</td>
<td>[('', 48), ('earth', 18), ('premises', 16), ('the middle east', 6), ('6/11', 4), ('the moon 02:01 billionaire', 4), ('thequint.com/whatsapp', 2), ('the moon', 2), ('malta.[83][84', 2), ('communities', 2)]</td>
</tr>
</tbody>
</table>
Module 6: Important Topics

6.1 LSA/LDA

We applied Latent Dirichlet Allocation (LDA) and later Latent Semantic Analysis (LSA) to do topic modelling. We use the results of topic modelling later to do clustering, since we found many similar topics that we want to combine. We first applied LDA using Gensim’s implementation, with number of topics ranging from 5-45, but the selected keywords and printed headlines from the main articles of the topic didn’t organize it into anything that looked like coherent categories to us. Results can be seen in Table 10.

Table 10. LDA results, 5 topics (based on word frequency):

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.012**&quot;app&quot; + 0.006**&quot;call&quot; + 0.005**&quot;collect&quot; + 0.005**&quot;make&quot; + 0.005**&quot;share&quot;</td>
</tr>
<tr>
<td>1</td>
<td>“campaign” + 0.009**&quot;app&quot; + 0.008**&quot;work&quot; + 0.008**&quot;kogan&quot;</td>
</tr>
<tr>
<td>2</td>
<td>0.012**&quot;campaign&quot; + 0.006**&quot;person&quot; + 0.006**&quot;elect&quot; + 0.005**&quot;polit&quot; + 0.005**&quot;access&quot;</td>
</tr>
<tr>
<td>3</td>
<td>0.004**&quot;year&quot; + 0.004**&quot;googl&quot; + 0.004**&quot;share&quot; + 0.004**&quot;report&quot; + 0.003**&quot;person&quot;</td>
</tr>
<tr>
<td>4</td>
<td>0.004**&quot;group&quot; + 0.004**&quot;post&quot; + 0.004**&quot;news&quot; + 0.003**&quot;make&quot; + 0.003**&quot;say&quot;</td>
</tr>
</tbody>
</table>

We then applied Gensim’s implementation of LSA, to see if it would perform better. We found it was necessary to use the TF-IDF scores instead of word frequency to get good results. LSA created much better topics than LDA, although we found that it would create many similar looking topics (such as topics 3-4 seen in Table 11). LSA has no parameters to tune, and topics are ranked by “explained variance” so typically you just select the first few topics and get rid of all the rest. Each document has some weight towards each topic, so the representative
documents for a topic are the ones with the highest weight. The LSA results can be found in Table 11.

Table 11. LSA results (top 5 topics, based on word frequency):

<table>
<thead>
<tr>
<th>0</th>
<th>0.189&quot;kogan&quot; + 0.141&quot;cambridg&quot; + 0.131&quot;wyli&quot; + 0.130&quot;said&quot; + 0.127&quot;analytica&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• trump consultants harvested data from 50 million facebook users: reports</td>
</tr>
<tr>
<td></td>
<td>• donald trump consultants harvested data from 50 million facebook users: reports</td>
</tr>
<tr>
<td></td>
<td>• revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach</td>
</tr>
<tr>
<td></td>
<td>• what to know about facebook's cambridge analytica problem</td>
</tr>
<tr>
<td></td>
<td>• facebook suspends data-analysis firm cambridge analytica from platform - business insider</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>0.324&quot;kogan&quot; + 0.197&quot;wyli&quot; - 0.171&quot;ftc&quot; - 0.137&quot;android&quot; + 0.119&quot;suspend&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• facebook suspends trump campaign data firm cambridge analytica</td>
</tr>
<tr>
<td></td>
<td>• presstv-facebook suspends trump campaign data firm</td>
</tr>
<tr>
<td></td>
<td>• massachusetts launches probe into cambridge analytica's use of facebook data</td>
</tr>
<tr>
<td></td>
<td>• donald trump consultants harvested data from 50 million facebook users: reports</td>
</tr>
<tr>
<td></td>
<td>• facebook bans trump campaigns data analytics firm for taking user data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2</th>
<th>-0.440&quot;ftc&quot; - 0.184&quot;decre&quot; + 0.176&quot;android&quot; - 0.150&quot;committe&quot; + 0.132&quot;text</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• grassley, ftc, states turn screws on facebook amid data flap - politico</td>
</tr>
<tr>
<td></td>
<td>• u.s. and british lawmakers demand answers from facebook chief executive mark zuckerberg - chicago tribune</td>
</tr>
<tr>
<td></td>
<td>• ftc confirms facebook investigation over data privacy concerns</td>
</tr>
<tr>
<td></td>
<td>• facebook may have violated ftc privacy deal, say former federal officials, triggering risk of massive fines - the washington post</td>
</tr>
<tr>
<td></td>
<td>• ftc is investigating facebook over cambridge analytica's use of personal data, source says</td>
</tr>
</tbody>
</table>
6.2 Clustering

Once we have the results of topic modelling from LSA, the idea is to use the topic vs. document matrix, and perform some kind of clustering in n-dimensional space, to group documents together. The vector we position in n-dimensional space is represented by each column of the matrix. The rows represent each topic and how much a document is weighted in that topic or dimension. Note that LSA provided us 200 topics with 2922 documents, so the dimensions of the topic matrix is (200 x 2922)

There are a total of four things to consider when clustering documents, based on topic weights:

➔ Clustering Algorithm
➔ Distance Function
Since we were not sure which algorithms with what parameters would work well, we decided to write a script to run all the specified clustering algorithms. `clustering.py` runs the clustering algorithms with a variety of numbers of topics and cluster sizes and saves the results as a .json file. We ran four kinds of clustering algorithms: K-means, Single Linkage Clustering, Average Linkage Clustering and Complete Linkage Clustering. We re-ran each of these algorithms for a number of topics sizes, inside which we also varied the number of cluster numbers. See Figure 4 for a visual representation of everything we tried.

![Flowchart detailing all clustering techniques used](image)

Therefore, we ran the four clustering algorithms a total of 280 times, all with different parameters. The goal was to see which combination performs the best.
Observe that the number of clusters is centered around half the number of topics. We evaluated the performance by two visual metrics:

1. Plotting the results on a 2-D graph with the first two topics as axes
2. Printing the headlines of 20 randomly selected documents per cluster

Our team manually evaluated the results for each kind of clustering algorithm, made observations, and then finally narrowed it down to a few categories.

**Note:** Before using Euclidean distance, it is necessary to normalize the matrix! This causes the distance to be proportional to cosine distance, so that the clustering results are equivalent to clusters created using cosine distances. This is preferable because the magnitude of each vector (the amount of weight it has towards each topic) should not matter when computing distances between vectors.

**General Observations**

We made the following observations on looking at the results:

➔ For all of the agglomerative hierarchical clustering techniques (single, average, complete), the results of clustering by cosine distance and Euclidean distance after normalizing were exactly the same. This is consistent with what we had expected.

➔ Single and Average linkage algorithms were producing one or two massive clusters, and the results did not look good. This is probably because edges selected between documents must have just ended up forming one giant chain as most documents are indeed placed close to each other.

➔ The best plots and clusters were made for those that had about half the number of topics as clusters.

➔ A lower number of topics seems to generally cluster better. This makes sense because LSA generates topics such that the variance in the dataset is
explained more by the first ones than the last ones. Therefore, by eliminating a larger number of later topics, the clustering focuses more on topics that matter! This observation inspired us to attempt reweighting the topic x document matrix, which we talk about below.

An interesting idea we came up with was to reweight topic vectors for each document such that more importance is given to the first few topics. LSA, which is a variant of Principal Component Analysis, will order topics such that the most important ones appear first, and the least important ones appear later. Our clustering algorithms do not take this importance into account by default and care equally about each topic. Therefore what we decided to do is reweight the vectors such that topic 1 is weighed far more than topic 200.

We reweighted each topic vector in each document using the following transformation:

$$d_{ij} \rightarrow d_{ij} / (\frac{i^2}{200})$$

We pick 200 since there are 200 topics, and dividing the $i^{th}$ topic by $i^2$ ensures that we exponentially lower the scores, and thereby importance, of topics as we go down. Topic 1 is now far more important during clustering than topic 200.

**Finalized Clustering**

Now that we know the clustering algorithms that work, we had to select the best topic and cluster sizes. Ultimately, we settled on using K-Means with normalized Euclidean distance and complete linkage with cosine distance. We also run both
these algorithms for the reweighted topic matrix. The topic sizes we selected were [100, 80, 60, 40, 20]. Number of clusters is centered at half the number of topics.

We retrieved a total of 120 different results that we analyze in the next section. Each ‘result’ is comprised of a plot and corresponding text file that contains titles of 40 random documents in the cluster.

Finally at the end of the clustering process and after evaluating everything, we picked the following two as the best ones:

1. K-Means, 40 topics, 16 clusters (Original Topic Weights)
2. K-Means, 40 topics, 24 clusters (Reweighted Topic Weights)

### 6.3 Plotting Clusters

We created a script `plot_clusters.py` to generate plots for all of the various clustering algorithms we tried. To visualize the clusters, we plotted document weights towards the first 2 principal components from LSA against each other, and color-coded them by cluster (this can be seen in Figure 5). Since the earliest principal components explain most of the variance, we decided that this provided a decent visual representation, and found that a clean-looking plot would usually correspond to good clusters.
Figure 5. Results of K-means clustering

We used a scree plot with k-means to determine the approximate number of clusters to use. To generate this plot, we ran k-means on the data with varying numbers of clusters (1-600) and plot the error for each cluster (sum of squared distances to cluster centroid). This produces a decaying exponential, and the recommended number of clusters is at the inflection point. On our dataset, this
inflection point occurred around \((num\_topics / 2)\), which is why we use values for \(num\_clusters\) centered around that, as illustrated in Figure 6.

### 6.4 Final Topic Results

The two clustering results we discuss in this section are:

1. K-Means, 40 topics, 16 clusters (Original Topic Weights)
2. K-Means, 40 topics, 24 clusters (Reweighted Topic Weights)

These seemed to have the best results of everything we analyzed. The clusters formed coherent plots, and the 40 documents we evaluated were clustered well together.

For example, the following headlines are from a cluster that clearly talks about what the FTC did in response to the data breach and how they have putting pressure on Facebook

- ftc confirms facebook data security investigation
- facebook is now under ftc investigation for the cambridge analytica data scandal  breaking news
- facebook's privacy practices are under investigation, ftc confirms | technology | the guardian
- the ftc is officially investigating facebook's data practices | wired
- ftc, states increase pressure on facebook on privacy  the denver post
- ftc confirms probe into facebook data misuse scandal  techcrunch - blogmag | eq4c
- ftc confirms probe into facebook and cambridge analytica data scandal
- ftc launching investigation into facebook over privacy practices - market business news
- ftc is investigating facebook over cambridge analytica
Figure 7: Different results produced from clustering

As can be seen in Figure 7, each algorithm produces three things: the plot used to visualize the clusters, the txt file so we read the contents of the clusters in a human-friendly format, and clusters.json that contains the entire texts of all documents in each cluster. The .json file is used as input into the extractive and abstractive summaries.

We also manually evaluated and wrote a one-line description for each cluster, so that we can tag the main idea behind each cluster and reference it more easily later. This was just for helping make it easier to evaluate the clusters, and does not affect our summaries at all. The two .json files we produced are combined into one using combine_clusters.py, which re-numbers the clusters and concatenates the 2 .json files together. Note that in each individual clustering output, each article appears in only one cluster, but after combining 2 cluster output files, each article will appear in 2 clusters.

Each line in the final output file (clusters.json) that is used to generate summaries, is of the following format:
{ "title": [Array of titles of each document in this cluster], "originalurl": [Array of URLs of each document in this cluster], "text": [Array of texts of each document in this cluster], "description": The main idea of this cluster, "clusterid": The id of this cluster }
Module 7: Extractive Summary of Important Sentences

We used two main approaches to create extractive summaries, explained below.

7.1 Overall Summary Approach

The first approach we tried was extracting the most important sentences from each article, merging those sentences into one “article”, and then extracting the overall most important sentences from that. To accomplish this we used Gensim’s summarize function, which uses TextRank [18]. The first summary generated included the big concepts from the corpus, but it contained many duplicate sentences. To improve this, we simply checked whether a sentence was a duplicate of any sentence already in the summary before adding it to the summary. This reduced the repetitiveness of the summary, but there were still many sentences that were not exact matches containing the same information.

After running the extractive summary code on the small and large data set, we were still seeing some repetitiveness. One thing we noticed was that the majority of news sources represented in our corpus only contributed a single document. Websites such as the New York Times or The Guardian which were more reputable primary news sources, however, contributed multiple articles to the dataset. Given this observation, we decided to try creating summaries only using articles from websites above a certain threshold of articles contributed. We ended up writing a Python script called select_top_k_documents.py that filters the corpus to only those documents that belong to the top $k$ most frequent news sources (based on domain name). The code created a mapping between document and domain name, then selected only documents from websites within the top 100. This performed well. Our hypothesis was that it is likely that the less common news sources provided documents of lower quality. From the results we obtained, it seems that our hypothesis was correct. The reason this change
performed well is that repetition was reduced and only the most credible important information present in the top 100 sources was selected. The issue with less documents was that we would observe irrelevant content, and too many documents induced repetition. A k-value of 100 seemed to balance this out very well.

We shortened our corpus down to the top $k$ documents with multiple $k$ values, and generated extractive summaries from them. We selected the following values for $k$: [5, 20, 50, 100, 200, 300]

After examination, we had the following results:

- $k=5$ and $k=20$ created summaries with unnecessary details which covered only a small number of the events that happened during the incident. We theorize that the extractive summarizer was not able to get a wide enough variety of input text from the low number of articles and homogenous sources.
- $k=200$ and $k=300$ generated summaries that were too broad and contained a wide variety of irrelevant information. By allowing so many sources, many extracted sentences were from opinion pieces and junk articles.
- $k=100$ generated the best results, with a good balance and wide coverage of incidents during the data breach. There was very little repetition (although it was not non-existent), and grammatically made sense.

Note that all these results can be found in the submitted files (with names following the format: extractive_summary_top<#>.py).

In conclusion, the summary created from the top 100 websites produced the best results; however, there were still some duplicate ideas in that summary that we wanted to get rid of.

We wanted to find a way to compare sentence similarity and figuring out what percent similar they needed to be to say that they were “duplicates.” We used
difflib.SequenceMatcher in order to compare the similarity of each of the sentences as we added them to the summary [19]. Through a trial and error process and reading through the summaries each time, we tested differing percent similarities. We found that removing sentences that were more than 50% similar yielded the best results. Based on word count, about 26.5% of the summary was removed for being too similar to other sentences already in the summary. To highlight the importance of this step, Appendix A section A.1 contains a list of duplicate sentences that were removed from our final summary. This code can be found in Unit7.py. The final results from this method of extractive summarization can be found in Appendix A section A.2. The only thing that had to be done manually was going back and adding the correct capitalization, since we had converted all the text to lowercase in order to make data cleaning at the beginning easier.

7.2 Clustered Summaries Approach

The second approach was to use the clusters we created and then apply the extraction on each topic to create a more well-rounded summary. Our first extractive summary was not bad, but it failed to mention anything about Facebook stock falling or the Senate hearing, which were important parts of the event as well. The basic approach taken here was to first create a summary for each cluster. Once that was done, we would combine these and then remove the duplicate ideas again. This was our initial plan but it had to be adjusted a little bit.

First of all, we ended up with 40 clusters, but not all of them were relevant, or at the very least were less significant than some of the others. So, instead of creating a summary for every cluster, we decided to select on the best clusters to summarize. This included trying topics we labelled as “general” to get the gist of the event and topics we labeled as Congress testimony, as well as others. We had
to do some experimentation on which clusters to use to get the best results. One strange finding was the appearance of sentences like “broadsheet leaked scandal mention revealed steps breach respond facebook boss mark zuckerberg apologized for the data breach that was (1) __________ last week” in our summary. These originated from a fill-in-the-blank exercise from an ESL lesson plan website that used news stories to create exercises to teach English. We removed those web pages from our dataset.

After summarizing each of the clusters we selected, we combined all of the summaries into one big summary. This summary proved to be too long and also a bit repetitive. To fix this, we decided to take a similar approach to the first method of extractive summarization we tried; we removed all of the similar and repetitive sentences and then created another summary of all of the cluster summaries concatenated together. This summary (generated by Unit7_clusters.py) was still a little bit too repetitive and still did not capture some of the key topics we were looking for, like the Senate hearing or stock prices dropping.

Our final step in creating a good extractive summary was to make two summaries, each with a different group of clusters. This way we could really emphasize topics which were important to the event but covered in fewer articles and thus not showing up in the earlier single-paragraph summary. We decided that the first paragraph should be general information about the data breach, and that the second would be about the effects of it: the Senate hearing and Facebook stock dropping. The code to generate this final summary can be found in Unit7_split_cluster_summary.py. In our final, best summary from the clustered approach, we ended up using the clusters and topics found in Table 12 for each paragraph.
The final extractive summary created using clustering can be found in Appendix A section A.3. Similar to the first summary, capitalization was added manually after. After getting results from all of our summarization methods, we determined that the two paragraph cluster based extractive approach yielded the best results.
Module 8: Values for Each Slot Matching Collection Semantics

In trying to decide which pieces of information were most important to the collection that could be generalized to other similar collections, we originally struggled as there are multiple kinds of events in the corpus. The Cambridge Analytica scandal was predominantly covered as a data breach [20], but it also led to several government hearings and testimonies in different countries [21], the passage of a law [22], and stock prices falling [23], among other fallout [24].

Ultimately, the plan was to use a combination of two event type summaries: a data breach or security event and a legal hearing. The data breach template was written based on information in GDPR incident reporting paperwork and advice from graduate student Ziqian Song who has been researching data breaches (personal correspondence). The legal hearing template was modified from a NIST template for guided summarization of investigations or trials [25]. The final list of tokens and phrases we planned to collect is in Table 13.

We originally planned to do as much as possible with regular expressions, and prototyped a date-finding code using regular expressions, the progress of which can be found in Unit8RegexPrototype.py. The code utilizes the Python packages num2words [26] and word2number [27] to convert all word representations of numbers in the text (e.g., 50 million) to their numeric representation (e.g., 50000000). This standardized different forms of reporting numbers and also allowed for easier regular expressions to be written for the actual value-finding.
Table 13. A list of planned pieces of information to be extracted from the corpus. Values are included for the pieces of information where they were gathered (highlighted in green), but time ran out before code could be written to extract all desired entities.

<table>
<thead>
<tr>
<th>Cambridge Analytica Data Breach</th>
<th>Senate Hearing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entity</strong></td>
<td><strong>Extracted Information</strong></td>
</tr>
<tr>
<td>1. Date/time of breach</td>
<td>2014</td>
</tr>
<tr>
<td>2. Number of affected users</td>
<td>50000000</td>
</tr>
<tr>
<td>3. Name of affected website/service/company</td>
<td>facebook</td>
</tr>
<tr>
<td>4. Type of data exposed in breach</td>
<td>information from the accounts friends profiles as well as updates, likes, and in some cases private messages</td>
</tr>
<tr>
<td>5. Name of attacker</td>
<td></td>
</tr>
<tr>
<td>6. Name of Whistleblower</td>
<td></td>
</tr>
<tr>
<td>7. Date/time breach made public</td>
<td></td>
</tr>
<tr>
<td>8. Company statement</td>
<td></td>
</tr>
<tr>
<td>9. Changes enacted by company</td>
<td></td>
</tr>
</tbody>
</table>
We originally tried this approach because it would be faster and easier to parallelize than using named entity recognition or other methods that required computationally-intensive language models. Additionally, spaCy, the package many other teams were using for this task, recognized things like “last Monday” as dates with the same entity label as “October 10th”, which would not be particularly helpful for trying to pick out dates that would be meaningful in a summary. But due to the journalistic style of using things like “last Monday” more than the full date when reporting on events, and due to the fact that many dates we were trying to extract were a month or year value only, our regular expression approach could not easily pick out dates from other numbers.

We ran `Unit8RegexPrototype.py` on specific document clusters corresponding to articles about the senate hearing or general data breach, hoping to find the hearing date (10) and either the breach date (1) or the date the Cambridge Analytica scandal broke (7). March 17, the date of the scandal breaking (7) was only the 4th most common date found in clusters 0 and 2, with only 15 total occurrences. It was beaten out by other dates later in March. April 10, the date of the senate hearing (10), was the most common date found in clusters 11 and 30, but we still decided at this point to attempt to use spaCy for date and entity recognition instead.

The date of the breach, 2014, was extracted using spaCy named entity recognition to pull dates from sentences that had been filtered to include some word which lemmatized to “steal”, “harvest”, “collect”, or “obtain”. The number of users whose data was leaked and the website they were users of were obtained from the same regular expression. The correct results in the above cases were determined by selecting the most common result.

The phrase that describes the type of data which was leaked was also acquired using spaCy, but instead of using named entity recognition, the parse tree functionality was used to get the full object clauses relating to any instances of
the verbs “steal”, “harvest”, “collect”, or “obtain”. The results included several instances of the word “data” and some very long phrases which appeared to be poorly-parsed as a result of lack of punctuation. Thus, we filtered out results with extreme length (by number of tokens). While this improved the data, the three most common clauses were still “no passwords or sensitive pieces of information”, “millions of facebook profiles of us voters”, and “private information from the facebook profiles of more than 50 million users”, none of which were very descriptive. We initially attempted to filter for clauses containing some more generic word such as “data” or “personal” or “private” to make our solution more generalizable to other corpora about data breaches, but the results were still not greatly improved by these filters and we decided that they would already cause issues for certain corpora if the correct value was, for instance, “social security numbers” or “passwords”. Thus, based on our knowledge of the answer we were hoping to extract for this corpus, we only kept clauses which contained the words “and” and “friends”. This greatly increased the relevance of the remaining answers. However, it is worth noting that this filtering process will likely need to be changed to be more generalizable.

For the type of data leaked in the breach, we ultimately decided to use the longest result remaining (“information from the accounts friends profiles as well as updates , likes , and in some cases private messages”) rather than the most frequent (“personal information about themselves and their friends”) because it was more correct and more specific.

A prototype code is also included in the file Unit8.py to extract the name of the whistleblower (6), which operates by finding any names adjacent to the word “whistleblower” or “whistle-blower” in the text. However, when we went to run the code, we found that it was not picking up any tokens and that, in fact, no instance of “christopher wylie” in the corpus was recognized as a named entity by spaCy. We believe that this is due to the lack of capitalization in cleaned.json.
and clusters.json. Due to time constraints, we decided not to rewrite and rerun all of our cleaning code to generate a corpus with capitalization included, and instead shifted focus to the more promising abstractive and extractive summary results.

The entities we did extract are listed in green in Table 13, and the code used is in Unit8.py. We used regular expressions in this code where possible, as it was a much faster process even without parallelizing. Running the initial natural language modelling necessary for spaCy on our entire corpus took about an hour on a high-end personal computer, and each extraction could take large chunks of time as well. While this is not necessarily prohibitive to run, it made prototyping and testing potential solutions very slow. Thus, regular expressions have a distinct speed advantage.
Module 9: Readable Summary Explaining Slots and Values

Templates were written to contain the values from the two event type collections in Table 13. They are as follows, using the numbering scheme from Table 13:

In -1-, the data of -2- users of -3- was compromised. -4- was illegally obtained by -5-. The incident was made public by -6- on -7-. -3- has said -8- and will -9-.

On -10-, -11- was brought before the -12- as part of an investigation into -13-. -11- said that -14-. As a result of the hearing, -15-.

We attempted to make the two halves of this summary as general as possible so that the code could be repurposed for other similar incidents while still covering the important data to this specific collection. We might expect the filled summary to look something like this (filled manually), although extracting many of these long tokens would be difficult:

In 2014, the data of 50 million users of Facebook was compromised. Private personal details from profiles of consenting users and their unconsenting friends was illegally obtained by Aleksander Kogan. The incident was made public on March 17, 2018 by Christopher Wylie. Facebook has said that “We know that this was a major violation of people’s trust, and I deeply regret that we didn’t do enough to deal with it.” and will continue to redesign its privacy settings in response to criticism.

On Tuesday April 10, 2018, Mark Zuckerberg was brought before the Senate Commerce and Judiciary Committees as part of an investigation into data privacy on social media and the appropriate use of user data. Mark Zuckerberg said that he was in favor of increased regulation of Facebook and apologized for past mistakes in handling of user data. As a result of the hearing, no legislation has yet been passed.
There were three approaches we took for generating abstractive summaries of the Facebook breach corpus. All our approaches are explained below, with the first one being the most successful of the abstractive results.

### 10.1 Pointer-Generator Network (PGN)

Researchers at Stanford University and Google Brain developed an attentional sequence-to-sequence network for abstractive single-document summarization called a Pointer-Generator Network (PGN) [5,6]. This architecture makes modifications to a traditional sequence-to-sequence approach in order to address issues in generating summaries that are factually accurate, non-repetitive, and can contain out-of-vocabulary words.

PGN can be thought of as a mixture between an abstractive and extractive summarizer, as it has a variable probability at each decoder step to either use a pointer to copy words from the input document or to generate new words based on the vocabulary, both guided by the attention distribution. It also uses a coverage vector, a sum of all previous attention distributions, as an input in the attention calculation, making the likelihood of repetition lower.

This methodology, developed in 2017, increases the accuracy of fact reproduction while still allowing for the use of novel words. The original paper trained the model on a dataset of CNN and DailyMail articles and this pre-trained model was available on GitHub [5].

**Setup**

The code for training, evaluating, and generating summaries with PGN models is available on GitHub [6].
The original code for scraping the CNN/DailyMail dataset and outputting it in the format needed for training, evaluation, or summarization is available on GitHub [28].

Code for taking .json files of the type output by *jusTextCleaningBig.py* was written (based on the above) by classmate Chreston Miller and is available on GitHub [29].

The pre-trained model was saved using TensorFlow’s checkpoint system, and as such could only be loaded in an environment running the same version of TensorFlow (1.2.1). This meant that the model could not be loaded on the ARC cluster and several code modifications (Python 3 compatibility, added flexibility to specify file locations) were made to the above codebase in order to run on an available system with a compatible GPU. ROUGE evaluation ability was also removed, as we were planning to use the model for unsupervised summarization only and were not interested in ROUGE comparison to gold standards.

The vocabulary file from the CNN/DailyMail dataset was used based on advice from multiple classmates that using a dataset-specific vocabulary file made little difference in the quality of summaries. The vocabulary file is produced by [28]

**Results**

PGN creates a 1-3 sentence summary from an article-sized input document. It has a hard upper limit of 120 tokens, but is generally self-limiting below that cutoff. Three different approaches were tried with PGN.

**Approach 1: Single Document Summarization**

As a proof of concept and an assessment of how well the pre-trained model could handle the dataset, we decided to use the code as intended for single-document summarization and summarize each document in the collection individually. We were concerned that the technical nature of the dataset and its density of names
would present a challenge, but the resulting summaries were largely factually and grammatically correct. Table 14 contains some of the summaries generated as examples.

Table 14. Example Single Document Abstractive Summaries

<table>
<thead>
<tr>
<th>senador amy klobuchar of minnesota, a democratic member of the senate judiciary committee, went so far as to press for mark zuckerberg, facebook's chief executive, to appear before the panel to explain what the social network knew about the misuse of its data to target political advertising and manipulate voters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>the calls for greater scrutiny followed reports on saturday in the new york times and the observer of london that cambridge analytica, a political data firm founded by stephen k. bannon and robert mercer.</td>
</tr>
<tr>
<td>facebook responded to reports that political data firm cambridge analytica gained access to data on 50 million of its users.</td>
</tr>
<tr>
<td>she did not say when regulators had started looking into facebook.</td>
</tr>
<tr>
<td>later on tuesday, cambridge analytica said facebook's search “would potentially compromise a regulatory investigation.”</td>
</tr>
<tr>
<td>&quot;the british government has called the allegations very concerning.</td>
</tr>
<tr>
<td>eu justice chief vra jourov said on friday, “it’s clear that data of europeans have been exposed to a huge risk and i am not sure if facebook took all the necessary steps to implement change.” exclusive/digital services that collect users data, like facebook and gmail, will be pulled under eu consumer protection rules.</td>
</tr>
<tr>
<td>possible sanctions will be raised to up to 4% of a company's turnover.</td>
</tr>
<tr>
<td>two former federal officials who crafted the landmark consent decree governing how facebook handles user privacy say the company may have violated that decree when it shared information from tens of millions of people.</td>
</tr>
<tr>
<td>the group included both the 270,000 facebook users who downloaded a psychological testing app and the facebook friends of those people.</td>
</tr>
</tbody>
</table>
We were particularly impressed with the correct assignment of people to their roles, such as in “Christopher Whyle, a former employee of Cambridge Analytica”, and “Political data analytics company Cambridge Analytica is affiliated with Strategic Communication Laboratories (SCL)”. It does often leave pronouns undefined, as in “separate from his university work” (which should be referring to Kogan) and “she did not say when regulators had started speaking to Facebook” (antecedent unknown from context). The summaries do have mostly correct punctuation, but it would need further processing to be human-readable (as in the replacing of -lrb- with a parenthesis and removal of extra spaces).

**Approach 2: Abstractive Summarization of Extractive Summaries**

In order to end up with a reasonably-sized abstractive summary that covered the breadth of information in the dataset, the file `abstracts_to_json.py` re-aligns the abstractive summaries made by PGN (which are outputted as .txt files) with the URL, full text, and other metadata of the original article it was based on (from `clusters.json`). Then, this new file, `abstracts.json`, can be run through a slightly modified version of the extractive summarization method from Module 7, using `abs_to_ext_by_cluster.py`. The abstractive summaries for all documents within clusters identified as relevant are concatenated and then an extractive summary is generated. The same two-paragraph method is used as in Module 7.
The final summary using this approach, both in raw ASCII format and with manual corrections for punctuation and capitalization, is found in Appendix B section B.1.

By doing extractive summarization after individual document PGN, the redundancy-removing properties of the coverage vector were overridden and the output summary is more repetitive than the corresponding extractive summary. It also has some issues with grammar and factual correctness in the second paragraph. By more logically ordering the way the cluster summaries print, the summary flow could likely be improved somewhat.

**Approach 3: Extraction of Summaries By Cluster**

An extractive summary was generated for each cluster using a modified version of the Module 7 code, `all_cluster_summaries.py` (mainly modified to use all clusters), and then they were processed into .story files using `txt_to_story.py` and summarized using PGN, resulting in one abstractive summary per cluster. The best results were selected by hand and concatenated.

Unfortunately, several of the clusters (e.g., Cluster 24 and Cluster 0) had nearly-identical output despite having inputs which only partially overlapped, as can be seen in Table 15. The table also makes it apparent that the PGN is still partially extractive, with some rather large sentence fragments such as “Damian Collins, chairman of the digital, culture, media and sport committee, also accused the chief executive” preserved between input and output. This may be due to the relatively small size of document input in this workflow.
Table 15. A comparison of Cluster 0 and 24 summaries pre- and post-abstraction, with differences in the abstractive output highlighted.

<table>
<thead>
<tr>
<th>Input (Extractive)</th>
<th>Cluster 0</th>
<th>Cluster 24</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>collins, the british lawmaker, said he planned to call alexander nix, the chief executive of cambridge analytica, to return to parliament and answer questions about testimony last month in which he claimed that the company never obtained or used facebook data. nix told the committee last month that his firm had not received data from a researcher accused of obtaining millions of facebook users personal information. adam schiff, the top democrat on the house intelligence committee, called for cambridge analytica to be thoroughly investigated and said facebook must answer questions about how it came to provide private user information to an academic with links to russia. meanwhile, mark warner, the u.s. senator who is vice chair of the intelligence committee, has also requested that zuckerberg, along with other tech ceos, testify in order to answer questions about facebooks role in social manipulation in the 2016 election. during tuesdays committee hearing, wylie suggested facebook may have been aware of the large-scale harvesting of data carried out by cambridge analyticas partner gsr even earlier than had been previously reported. damian collins, chairman of the digital, culture, media and sport committee, also accused the chief executive of cambridge analytica, alexander nix, of deliberately misleading parliament and giving false statements to the committee following allegations it was passed personal data from facebook apps without the consent of</td>
<td>adam schiff, the top democrat on the house intelligence committee, called for cambridge analytica to be thoroughly investigated and said facebook must answer questions about how it came to provide private user information to an academic with links to russia. meanwhile, mark warner, the u.s. senator who is vice chair of the intelligence committee, has also requested that zuckerberg, along with other tech ceos, testify in order to answer questions about facebooks role in social manipulation in the 2016 election. during tuesdays committee hearing, wylie suggested facebook may have been aware of the large-scale harvesting of data carried out by cambridge analyticas partner gsr even earlier than had been previously reported. damian collins, chairman of the digital, culture, media and sport committee, also accused the chief executive of cambridge analytica, alexander nix, of deliberately misleading parliament and giving false statements to the committee following allegations it was passed personal data from facebook apps without the consent of</td>
</tr>
</tbody>
</table>
Out by Cambridge Analytica's partner GSR even earlier than had been previously reported. Damian Collins, chairman of the digital, culture, media and sport committee, also accused the chief executive of Cambridge Analytica, Alexander Nix, of deliberately misleading parliament and giving false statements to the committee following allegations it was passed personal data from Facebook apps without the consent of the individuals.

Mark Warner, the chief executive of the intelligence committee, has also requested that Zuckerberg, along with other tech CEOs, testify in order to answer questions about Facebook's role in the 2016 election. During Tuesday's committee hearing, Wylie suggested Facebook may have been aware of the large-scale harvesting of data carried out by Cambridge Analytica's partner GSR even earlier than had been previously reported. Damian Collins, chairman of the digital, culture, media and sport committee, also accused the chief executive of deliberately misleading parliament.

Mark Warner, the U.S. senator who is vice chair of the intelligence committee, has also requested that Zuckerberg, along with other tech CEOs, testify in order to answer questions about Facebook's role in social manipulation in the 2016 election. During Tuesday's committee hearing, Wylie suggested Facebook may have been aware of the large-scale harvesting of data carried out by Cambridge Analytica's partner GSR even earlier than had been previously reported. Damian Collins, chairman of the digital, culture, media and sport committee, also accused the chief executive of deliberately misleading parliament.
No individual cluster summary features redundant information, as is the goal of PGN, but since the input summaries are very similar for some clusters, the outputs for some are very similar. A modification to the extractive summary code or the workflow could potentially address this issue. Interestingly, even including the cluster summaries that were not included in the final draft included in the report, the 50 million figure is much less prominent than in most other summary methods, and in fact different summaries mention slightly different figures (including the 270,000 users who downloaded the original app). The summaries have a much heavier focus on the various investigations, companies, and names. As in the previous PGN workflow, there are still some issues with pronoun confusion, and the incidence of incomplete sentences has gone up, likely as an artefact of the occasionally-fragmented and relatively short extractive input.

The final output of this method, with manual corrections for punctuation and capitalization, can be found in Appendix B section B.2.

10.2 FastAbsRL

In addition to the pointer generator network, we also applied FastAbsRL [3]. The authors of the paper provide a pre-trained model online, so we downloaded that and applied it to our dataset. They trained the model on the CNN and DailyMail datasets, which should be similar to our data but over a broader domain of topics.

Results

On our dataset, FastAbsRL always creates a 1-sentence summary for each article. The algorithm decides on its own how long the summary should be, although you can specify the maximum length. Here are some example outputs (one sentence per article):

facebook’s facebook app has been embroiled in controversies over its data sharing.
theresa may has said she expects cambridge analytica and facebook to cooperate fully.
facebook says it does not collect the content of calls or text messages.
facebook has been keeping records of android users calls and texts.
facebook says logging is part of the #deletefacebook campaign.
facebook responded in a blog post on sunday.
facebook questioned about pulling data from android devices sunday, march 25th.
facebook says it has never sold any of the collected metadata.
facebook users have filed a lawsuit against the social-networking giant.
facebook pointed out that the call log was "a widely used practice to begin"

To combine these single-article abstractive summaries into a full-length summary of the dataset, we first divide the single-article abstractive summaries into clusters using the results from Module 6. We then run Gensim’s extractive summarizer on each cluster. To create the final abstractive summary, we run the extractive summarizer on several clusters of single-article abstractive summarizations. This is implemented in make_summary.py, which also has an interface to manually choose which clusters should be used, the order they will appear, and the number of sentences for that cluster. Here is an example of a summary extracted from one cluster of single-document abstract summaries:
(Cluster 2)
cambridge analytica improperly used data from some 50 million facebook users.
facebook ceo mark zuckerberg says the social media giant has violated their privacy.
facebook says it’s suspending cambridge analytica after finding data privacy policies.
facebook says the data of more than 50 million users were inappropriately used by us president donald trump.
britain is concerned about allegations that the data firm cambridge analytica exploited facebook data to use millions of peoples profiles without authorization.
facebook suspended cambridge analytica, a data firm that helped president donald trump win the 2016 election.
facebook has suspended cambridge analytica, a data company that helped donald trump win the presidential election.
australia is investigating whether local personal information was exposed in the facebook data breach the australian information and privacy commissioner.
See Appendix B section B.3 for the full abstractive summary. For our dataset, we found that the PGN (Section 10.1) had a better final summary, which is contrary to what we expected, since FastAbsRL reports better performance on ROUGE scores [3]. However, this may be because of the difference in processes used to obtain the final abstractive summary from the single-document summaries. It also may be due to some unique characteristic of our dataset, which is much more specific than the CNN-DailyMail dataset that PGN and FastAbsRL were both trained and tested on.

### 10.3 AbTextSumm

AbTextSumm is the implementation for the integer linear programming (ILP)-based abstractive text summarizer explained in the relevant paper [8].

The code can be found on GitHub at [9].

**Results**

The algorithm was successful in creating abstractive summaries, but unfortunately the algorithm produces sentences that are very short and end abruptly. See *AbTextSum_Summary.txt* for an example abstractive summary produced. Here are some example bad sentences from the summary:

- “The company and cambridge analytica, a firm founded by stephen k.”
- “The social media site, and sheera frenkel from san francisco.”
- “On tuesday in menlo park, calif., where the company is based.”
- “Warner, democrat of minnesota, and robert mercer, the wealthy republican donor.”
These sentences clearly end abruptly with grammatical errors, and make absolutely no sense. Furthermore, each summary was taking a long time to produce and we deemed this approach not applicable in our scenario.
Evaluation

ROUGE Evaluation

ROUGE, which stands for Recall-Oriented Understudy for Gisting Evaluation, is a set of metrics used to evaluate automatic summarization methods [30]. It consists of five evaluation metrics, three of which we used in the evaluation of our summaries. The five metrics are ROUGE-N (which consists of ROUGE-1 and ROUGE-2), ROUGE-L, ROUGE-S, and ROUGE-SU [30]. ROUGE-N is the co-occurrence of n-grams. Within this type of metric is ROUGE-1, which is the overlap of 1-grams (each word) between the two types of summaries, and ROUGE-2, which is the overlap of bi-grams (two words) between the two types of summaries. ROUGE-L looks at sentence structure similarity. ROUGE-S compares skip-bigrams, which are any pair of words that occur in the same sentence order between the summaries. ROUGE-SU builds on the ROUGE-S metric and looks at 1-grams on top of skip-bigrams.

We ran eval.py which was given to us by the Teaching Assistant for the course, Liuqing Li, to compare the hand-written gold standard summary Team 12 created for us to the automatically generated summaries we produced. Instructions for running this can be found in the User’s Manual. The results of our ROUGE evaluation are shown in Tables 16-18.
Table 16. Calculated ROUGE results

<table>
<thead>
<tr>
<th>Summary Type</th>
<th>ROUGE-1</th>
<th>ROUGE-2</th>
<th>ROUGE-L</th>
<th>ROUGE-SU4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster-based extractive</td>
<td>0.06557</td>
<td>0.0</td>
<td>0.06557</td>
<td>0.01198</td>
</tr>
<tr>
<td>Extractive</td>
<td>0.10714</td>
<td>0.0</td>
<td>0.10714</td>
<td>0.01316</td>
</tr>
<tr>
<td>PGN Approach 2</td>
<td>0.10169</td>
<td>0.0</td>
<td>0.0678</td>
<td>0.01863</td>
</tr>
<tr>
<td>PGN Approach 3</td>
<td>0.1</td>
<td>0.0</td>
<td>0.1</td>
<td>0.01829</td>
</tr>
<tr>
<td>FastAbsRL</td>
<td>0.09091</td>
<td>0.0</td>
<td>0.09091</td>
<td>0.01099</td>
</tr>
</tbody>
</table>

Table 17. ROUGE results based on sentences

<table>
<thead>
<tr>
<th>Summary Type</th>
<th>Max ROUGE-1</th>
<th>Max ROUGE-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster-based extractive</td>
<td>0.50000</td>
<td>0.27273</td>
</tr>
<tr>
<td>Extractive</td>
<td>0.75000</td>
<td>0.38889</td>
</tr>
<tr>
<td>PGN Approach 2</td>
<td>0.60000</td>
<td>0.27273</td>
</tr>
<tr>
<td>PGN Approach 3</td>
<td>0.60000</td>
<td>0.37500</td>
</tr>
<tr>
<td>FastAbsRL</td>
<td>0.66667</td>
<td>0.50000</td>
</tr>
</tbody>
</table>
Table 18. Entity Coverage results

<table>
<thead>
<tr>
<th>Summary Type</th>
<th>Entity Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster-based extractive</td>
<td>20.93%</td>
</tr>
<tr>
<td>Extractive</td>
<td>20.93%</td>
</tr>
<tr>
<td>PGN Approach 2</td>
<td>18.60%</td>
</tr>
<tr>
<td>PGN Approach 3</td>
<td>27.91%</td>
</tr>
<tr>
<td>FastAbsRL</td>
<td>18.60%</td>
</tr>
</tbody>
</table>

As Table 16 shows, our basic extractive summary achieved the highest ROUGE scores; however, we decided that the best summary we produced was the cluster-based extractive summary because it was more human readable and flowed better than the other summaries. The actual readability and coherency of a summary is one thing that ROUGE scores do not take into account, but that definitely makes a big difference to people when they read a summary. It is for that reason that we decided that despite its lower scores, the cluster-based extractive summary was actually our best.

**Solr Indexing**

We also did Solr indexing for our dataset. The main purpose of this was to make it easier for Team 15 to search our dataset while they were creating their Gold Standard summary for our team’s event.

The process for creating the Solr core was not too involved. Once we had our data in .json format (explained in User Manual), we first had to copy the .json into our
home folder on the Blacklight VM where Solr is hosted. Once that was done, we logged into Blacklight. We were given a Python script that converted our .json to the correct Solr format, called `json_formatter.py`. We ran the converter script, then used the following curl command to add it to our Solr indexing:

```bash
curl 'http://localhost:8983/solr/big_team<#>/update/json?commit=true'
--data-binary @[<solr_part JSON>] -H 'Content-type:application/json'
```

The first time we added our data to the Solr core was before finishing our data cleaning, so once we updated our .json with more cleaning techniques, we first had to delete the old data from Solr using the following curl command:

```bash
```

Then we had to update our Solr index by redoing the entire process with the updated .json.

As a team, we did not make much use of the Solr core for our dataset and processing. We did, however, use it when we were researching to make the Gold Standard for Team 15, the Maryland Shootings. This process is described in more detail below.

**Gold Standard Summaries**

**Gold Standard Creation for Maryland Shooting**

To create the gold standard summary for Team 15 covering multiple Maryland shootings, we decided to have 3 team members independently write their own summaries of the dataset, and then combined details from the 3. One team member reviewed 50-100 individual articles total from the team’s collection, starting with random articles. Additional articles on specific aspects were selected by searching for keywords in Team 15’s Solr index that showed up in the
Wikipedia articles for the most prominent recent Maryland shootings [31,32] and the team's presentations on most common words and bigrams. Another team member based the summary on only articles cited in the aforementioned Wikipedia pages. The third team member searched Team 15's Solr index and looked through the 50th - 150th article so as to not only gather information from the first articles in the dataset. Then the third team member searched for articles that included more details about the events. We found that different team members would pick up on different details, so combining details from the 3 summaries ensured that the final summary was comprehensive and fair. We chose details for the final summary based on whether they were found in multiple of summaries and their subjective importance or interest.

For details not present in all summaries, we used Solr to check for the information in the collection, and if it was not present or was present in no more than 2 unique articles, we flagged it as a minor detail in the collection. Our final summary is found in Appendix C.1.

**Gold Standard For Facebook Data Breach**

Another team, Team 12, was tasked with creating a Gold Standard summary for our dataset which covered the Facebook data breach. This was given to us after our final summaries were generated in order to be used as an evaluation method, as discussed in the ROUGE Evaluation section above. The summary is found in Appendix C.2.
User’s Manual

This project supports a variety of tasks that deal with creating a summary of a collection of web pages. Below are descriptions of the tasks supported and tutorials for each.

Cleaning Data

jusText Cleaning

Set-Up

Make sure you have a distributed computing system with the following requirements installed:

- Python 2.7.x
- PySpark 2.2.0
- Spark 2.2.0
- Scala 2.11.8
- Java 1.8.0_171
- Hadoop

Running and Interpreting Output

1. This codebase is intended to be used to convert a WARC file and a CDX file (or multiple WARC and CDX files) into a single .json file. First, place your zipped WARC file and unzipped CDX file into the Hadoop distributed file system (HDFS) using hadoop fs -put. Then, run the following files using Spark submit:
   a. ArchiveSpark_WarcToJson.scala - This file takes a zipped WARC file (.warc.gz) and a CDX file (.cdx), and will output a .json file of article
texts with duplicate articles from the same URL removed. Change the file paths in the top of the file as appropriate to your HDFS setup.

b. *jusTextCleaningBig.py* - This file takes the .json file that is the output of the previous step in which each document is a row, parses the page HTML, and returns plain article text with the boilerplate removed. It also does some filtering and attempts to grab a publication date from the metadata of the page where possible. The output file has one JSON object per row, each representing an article with the fields “isopubdate”, “originalurl”, “text”, and “title”.

c. Note: to run *jusTextCleaningBig.py*, you will need to pass

FacebookBreackSummarization_PySparkExtPkgs.zip to the submitted job using the argument --archives. For example:

```
spark2-submit --archives FacebookBreackSummarization_PySparkExtPkgs.zip#pyspark jusTextCleaningBig.py
```

2. Remove your files from HDFS using *hadoop fs-get*.

3. Run the resulting directory of .json files through the file *jsonFileConcat.py* to turn the multiple files returned by the worker nodes into a single long .json.

4. The .json file should now contain all of the relevant data from your web pages.

5. Expected runtimes (on a dataset of similar size)
   a. ArchiveSpark_WarcToJson.scala: <10 minutes (cluster with 20 nodes)
   b. jusTextCleaningBig.py: 5-30 minutes (cluster with 20 nodes)
   c. jsonFileConcat.py: < 1 minute

**Final Cleaning**

**Set-Up**

75
1. Follow the steps above in the **JusText tutorial** to get a properly formatted `.json` file

**Running and Interpreting Output**

1. Further finalized cleaning can be done once the JusText cleaning has been completed, and you have a file where each line is a JSON object representing a document (`cleaned.json`)
2. To perform further cleaning, run the following files in the given order
   a. `remove_hex_chars.py` - This file takes `cleaned.json` as the input and produces another `.json` file that has been re-encoded in UTF-8
   b. `further_cleaning.py` - This file takes the output of the previous file as input and produces another `.json` file that has removed documents matching specific parameters, as described previously in this report
   c. `remove_similar_documents.py` - This file takes the output of the previous file as input and produces another `.json` file that has removed documents which are categorized as too similar.
2. Now you will have `cleaned.json` containing only good data
3. Expected runtimes (on a dataset of similar size)
   a. `Remove_similar_documents.py`: 20 minutes
   b. `Further_cleaning.py`: 2-5 minutes
   c. `Remove_hex_chars.py`: < 5 seconds

**Single Word and Entity Based Summary Techniques**

**Most Frequent Words**

**Set-Up**

1. Make sure most recent version of NLTK is installed by running “pip install -U nltk”
2. Get your data ready
   a. Documents need to be in a .json file which contains a ‘text’ key with
      the article text.
   b. If you have a .warc, you can use `ArchiveSpark_WarcToJson.scala` to
      convert it to the correct .json format, and follow the `jusText cleaning
      tutorial` above to clean it.
3. Edit `Unit1.py` to refer to your .json file in the “input_file_path” global
   variable at the top of the file.
4. Edit `Unit1.py`'s main function to write to a .csv with a name appropriate to
   your dataset, by changing the following line:
   ```
   writeCSV(final_results, ".", "fb_big_frequentwords.csv")
   ```
   and replacing “fb_big_frequentwords.csv” with a file name of your
   choosing.

**Running and Interpreting Output**

1. Use Python 2.7 to run the code by using the following command:
   ```
   python2.7 Unit1.py
   ```
2. Results will be in output to the console and a .csv in order of most frequent
   to least frequent. The output will contain a word followed by its word
   count in the dataset.
3. Expected Runtimes (on a dataset of similar size): < 1 minute

**Named Entities**

**Set-Up**

1. Make sure most recent versions of NLTK and spaCy are installed by
   running “pip install -U nltk” and “pip install -U spacy”
2. Install a spaCy model for the English language using “python -m spacy download en”. You will need to run this command as an administrator and have a C++ installed.

3. Get your data ready
   a. Documents need to be in a .json file which contains a ‘text’ key with the article text.
   b. If you have a .warc, you can use ArchiveSpark_WarcToJson.scala to convert it to the correct .json format, and follow the jusText cleaning tutorial above to clean it.

Running and Interpreting Output

1. Be sure to use Python 2.7 when running this code.
2. Run Unit5.py with the documents in a .json file to extract the named entities from the data.
3. Expected runtimes (on a dataset of similar size): ~5 minutes

Clustering

Set-Up

1. Make sure recent version of NLTK and Gensim are installed by running “pip install -U gensim” and “pip install -U nltk”
2. Get your data ready
   a. Documents need to be in a .json file which contains a ‘text’ key with the article text.
   b. If you have a .warc, you can use ArchiveSpark_WarcToJson.scala to convert it to the correct .json format, and follow the jusText cleaning tutorial above to clean it.

Running and Interpreting Output

1. Be sure to sure Python 2.7 when running all this code.
2. Run `clustering.py` on the .json dataset (you may need to change the filepath as it is hardcoded) to create the cluster results file.

3. Run `plot_clusters.py` with the cluster results file as input to create lots of plots and .txt files containing headlines.
   a. You can assess the quality of a cluster by looking at the plot and by reading the headlines. The plot is just the first 2 dimensions of LSA, which we find is decently representative as those dimensions carry the most variance. The headlines are a random selection from the cluster.

4. Run `combine_clusters.py` to create clusters.json, which is used later on in the final summary scripts.

5. Expected runtimes (on a dataset of similar size)
   a. Clustering.py: 15 minutes to 1 hour depending on how many different algorithms are being run
   b. Combine_clusters.py: < 1 minute
   c. Plot_clusters.py: < 10 minutes
   d. Screeplot.py: 10-15 minutes

---

**Complete Event Summaries**

There are two complete methods of creating overall summaries provided in our solutions - extractive and abstractive.

**Extractive**

**Set-Up**

1. Make sure recent versions of NLTK and Gensim are installed by running “pip install -U gensim” and “pip install -U nltk”
2. Get your data ready
a. For overall approach
   i. Documents need to be in a .json file which contains a ‘text’ key with the article text.
   ii. If you have a .warc, you can use `ArchiveSpark_WarcToJson.scala` to convert it to the correct .json format, and follow the `jusText cleaning tutorial` above to clean it.

b. For cluster based approach
   i. Documents need to be in a .json file which contains JSON objects with keys: ‘text’ which is a list of all the articles’ text in that cluster and ‘clusterid’ to identify each cluster.
   ii. You can create clusters by following the `clustering tutorial` above.

3. Edit `Unit7_clusters.py / Unit7_split_cluster_summary.py` to include only the desired cluster IDs (if you are doing the overall approach, you can skip this step).
   a. `Unit7_clusters.py`
      i. In the “evaluate” function, change the if statement checking cluster IDs to only perform the summaries on the desired clusters by adding and subtracting from that list in the form:
         ```
         if c_id == 5 or c_id == 8 or ....
         ```
   b. `Unit7_split_cluster_summary.py`
      i. In the “evaluate” function, change the if statement checking cluster IDs to only perform the summaries on the desired clusters by adding and subtracting from that list in the form:
         ```
         if c_id == 5 or c_id == 8 or ....
         ```
      ii. In the “main” function, inside the for loop, change the if statement to group the correct clusters in each paragraph. The
clusters listed in the if statement will be in the first paragraph and all those not listed will be in the second paragraph.

4. Edit `Unit7.py / Unit7_clusters.py / Unit7_split_cluster_summary.py` to use the correct input_file_path (global variable at the top of the file) referring to your .json file.

Running and Interpreting Output

1. Be sure to use Python 2.7 to run the code with the following command:
   ```
   python2.7 <file_name>
   ```

2. To redirect the output to a file, instead of directly to the console, change the command to be:
   ```
   python2.7 <file_name> > <output_file>
   ```

3. Expected runtimes (on a dataset of similar size)
   a. `Unit7.py` = ~8 minutes
   b. `Unit7_clusters.py` = ~12 minutes
   c. `Unit7_split_cluster_summary.py` = ~6:30 minutes

4. Expected output
   a. Print statements explaining the progress of the program running
   b. A summary of your event data collection

Abstractive - Pointer Generator

Set-Up

The code provided in this unit, unlike most, is set up to work with Python 3.6.x. At present, some of the package versions required to load the pre-trained model will not work with Python 3.7, and the original Pointer Generator codebase was modified, among other things, to work with Python 3 instead of Python 2.
Install the following packages, paying attention to versions:

- pip3 install tensorflow-gpu==1.2.1
- pip3 install protobuf==3.6.0
- pip3 install http://download.pytorch.org/whl/cu80/torch-0.4.1-cp36-cp36m-win_amd64.whl
- pip3 install torchvision

Download a current version of Stanford CoreNLP [33] and add set the environmental variable CLASSPATH=path/to/stanford-corenlp-3.9.2.jar.

To convert clusters.json to the tokenized files necessary to run PGN:

1. Convert your input text to .story files:
   a. For starting with clusters.json:
      i. python json_to_story.py -f path/to/clusters.json -o path/to/output/directory
   b. For starting with an individual .txt file for each document:
      i. python txts_to_story.py -f path/to/input/directory -o path/to/output/directory
   c. For starting with a single .txt file where each line is a different document:
      i. python txt_to_story.py -f path/to/input.txt -o path/to/output/directory

2. python make_datafiles.py -w path/to/directory/from/step/one -m test.bin
   a. This tokenizes the story files and converts them to serialized binary files. It creates chunked files in case they are needed for performance, but the single test.bin file will contain all input documents.
   b. Do not change the name of test.bin.
3. Expected runtimes (on a dataset of similar size):
   a. json_to_story.py, txt_to_story.py, txts_to_story.py: <1 minute
   b. Make_datafiles.py: <2 minutes

Running and Interpreting Output

You will need your corpus as a series of .bin files and a url_metadata.txt file from the setup, a vocab file (the included vocab file from the CNN/Daily News corpus is usually sufficient), and the pretrained model (included in FacebookBreakSummarization_PretrainedPGN.zip). If you wish to run the code more than once, you will have to rename, delete, or move the generated decode_test_400maxenc_4beam_35mindec_120maxdec_ckpt directory.

1. python run_summarization.py --mode=decode --data_path=path/to/test.bin --vocab_path=path/to/vocab --log_root=path/to/logging/directory --exp_name=pretrained_model --max_enc_steps=400 --max_dec_steps=120 --coverage=1 --single_pass=1
   a. When mode=decode and single_pass=1, the output summaries will be written to numbered TXT files.
   b. Do not change any of the model parameters (max_enc_steps, max_dec_steps, and coverage)
   c. Your output will be written in $DATA_PATH/$EXP_NAME/decode_test_400maxenc_4beam_35mindec_120maxdec_ckpt/decode. $DATA_PATH/$EXP_NAME should point to the unzipped contents of FacebookBreakSummarization_PretrainedPGN.zip.

2. python abstracts_to_json.py -j path/to/clusters.json -m path/to/url_metadata.txt -o path/to/output/abstracts.json -s path/to/txt/files/from/step/one
   a. This code will re-align the output of run_summarization.py with clusters.json based on the URL and will create a new output .json file
which contains an additional “abstracts” field with the per-document abstractive summaries.

3. The output of this method will be a single 1-3 sentence abstractive summary for each document passed in, and we would advise running this through `abs_to_ext_by_cluster.py` to generate a single readable summary. You can also try other methods whereby you subset your documents first and concatenate them, or where the documents passed in are some sort of summary themselves.

4. Expected runtimes (on a dataset of similar size):
   a. `run_summarization.py`: 10-30 minutes
   b. `abstracts_to_json.py`: <1 minute
   c. `abs_to_ext_by_cluster.py`: 10-20 minutes

---

**Abstractive - FastAbsRL**

**Set-Up**

1. Obtain the code with the pretrained model and CNN-DailyMail dataset from GitHub [4]

**Running and Interpreting Output**

1. The readme explains how to preprocess the CNN-DailyMail dataset and run it with the pretrained model. We did not have to make any modifications to get it to run on Ubuntu 18 with the original dataset.
2. To convert a .json to a .story format, run `json_to_story.py`
3. Modify the `ChenRocks/cnn-dailymail` repo to process only the .story files desired.
4. Expected runtimes (on a dataset of similar size)
   a. `Make_summary.py` - is an interactive interface so will take as long as user wants it to.
Abstractive - AbTextSum

Setup

1. Clone the repository
2. Unfortunately, it seems like the codebase cannot be used directly and some changes need to be made so follow the steps below
   a. Install all the dependencies in requirements.txt from pip after updating your packages
   b. Add sys.path.insert(0, '/home/path_to_AbTextSumm)
   c. Modify
      
      PROJECT_DIR=os.path.dirname(os.path.dirname(os.path.realpath(__file__))))+"/"

3. You will need a binary-formatted ARPA language model for this to work. So follow the directions in https://kheafield.com/code/kenlm/ to install kenlm
4. Convert your entire corpus into a list of sentences using fb_sentences.py and create a text file
5. Install SRILM, which you will use to generate the ARPA language model.
7. Convert fb.arpa to a binary file using bin/build_binary fb.arpa fb.binary

Running and Interpreting Output

1. Be sure example.py expects the correct binary language model file
2. Run example.py using Python 2.7
3. Output will be an abstractive summary based on the .bin input
4. Expected runtimes (on a dataset of similar size)
   a. Dataset_to_sentences.py - < 5 seconds
Evaluating Gold Standard Summaries

Setup

1. Make sure pythonrouge is installed by running “pip install git+https://github.com/tagucci/pythonrouge.git” [34]
2. Get your data ready
   a. Put both the gold standard and generated summaries in golden/ and predict/ folders, respectively. Use the Github link shown above to see proper file and folder formats.

Running and Interpreting Output

1. Rouge_para
   a. Make sure ROUGE-L = True in the code, and then run eval.py with the parameters “-t 1 -g golden/ -p predict/”
2. Rouge_sent
   a. Run eval.py with the parameters “-t 2 -g golden/nameofgoldstandardsummary.txt -p predict/nameofpredictedsummary.txt”
3. Cov_entity
   a. Run eval.py with the parameters “-t 3 -g golden/nameofgoldstandardsummary.txt -p predict/nameofpredictedsummary.txt”
4. Expected runtimes (on a dataset of similar size):
   a. eval.py: ~0 s
Developer’s Manual

Flow of Development

The flowchart in Figure 8 presents the overall workflow we utilized for this project, starting with the original data and following it through all of the modules through to the final summaries.

We started with our corpus as a .warc file containing the raw web pages collected about our event. We then ran code to convert that to a .json and then clean that data up - meaning removing any bad data, such as 404 errors, removing any irrelevant data such as headers and JavaScript, and removing any egregiously duplicate information. cleaned.json is used in almost every unit following its creation.

Once we had the cleaned.json containing the clean text of all the good web pages, we finished working on Unit 1: finding the most frequent words. It was easiest to complete that unit first and then build Units 2, 3, 4, and 5 off of that since the following units also dealt with getting frequency counts. A lot of the Unit 1 code could be reused for the next four units by implementing additional processing before getting the word frequency counts.

Units 6 through 10 did not build as easily off of Unit 1 as the earlier units. Unit 6 only used cleaned.json to find the LSA topics, and then used the LSA topics to do clustering. The clustering that came off of Unit 6 did play into the later development in Unit 7, the extractive summaries. Extractive code development started by just running cleaned.json data through Gensim’s summarize function in multiple stages, but those results did not cover everything we wanted them to. This is where the clustering output came in, clusters.json. Cluster membership was used to split documents prior to summarization in Unit7_clusters.py and
Unit7_split_cluster_summary.py in order to get a broader range of topics. Editing which clusters are used and which are grouped together for the multi-paragraph summary yield the best results (steps for this can be found in the User’s Manual).

Units 8 and 9 aimed collectively to extract a set list of important words or phrases and slot them into a template which could be make a human-readable English summary. We originally intended to use regular expressions for this workflow, an older method for attempting to make rapid summaries of a type of event which occurs frequently. However, this proved unwieldy with the variety of irregularities in the natural language text. It was also a departure from the technologies built in previous units, using only clusters.json to limit the search space for any given word or phrase to articles likely to contain it. Building upon the Unit 5 code, which uses spaCy to tag named entities, we discovered that the flexibility of navigating and filtering the text in spaCy was very powerful. However, it was also much slower than regular expressions, and we ultimately decided to focus our development efforts on Units 7 and 10 for generating the final human-readable summary. More details can be found in Modules 8 and 9 above.

Unit 10, the abstractive summaries, follows two flows. The first, labeled “1,” takes clusters.json and runs each cluster through the abstractive summarizer. After that is done, each abstractive summary is concatenated together and that is run through the extractive summarizer in order to yield the final summary. The second flow, labeled “2,” takes the extractive summarizer and creates summaries for each cluster. Then it adds all of those together and runs it through the abstractive summarizer, which gives the final summary output. In depth explanations of our methodology can be found above in the Module 10 section.
Data Processing

The data provided were web pages, meaning that they had both HTML and other machine-readable but inconsistent and irrelevant scripting along with natural language with all of the typos and idiosyncrasies that come with it. Early filtering steps involved removing boilerplate; removing HTML, JavaScript, and CSS; removing 404 forbidden and other error pages; removing splash pages; removing duplicate URLs or duplicate articles; and removing foreign language pages. We converted all text to lowercase to improve the ease of text comparison, although we later realized that this made named entity recognition a more difficult task. An in-depth explanation of our data processing and cleaning process can be found in the Data Cleaning section of the Approach.

In general, we wrote most of our code to start from either cleaned.json or clusters.json depending on whether document category was relevant. However, there were additional cleaning steps in the code for some units. For many units, we filtered out a list of stopwords (extended from that packaged with NLTK) to prevent common tokens from appearing at the top of our results. The lemmatization code (Unit 4) was used to improve the Unit 2 results. TF-IDF filtering was used before topic modelling to improve the quality of the topics.

Maintaining and Extending

For information on how to set-up and run the various deliverables we have provided, see the User’s Manual section.

There are several enhancements and extensions that could make the project better. One useful enhancement would be to create some modules for commonly-used functions, since we ended up copy-pasting certain things like data reading function in between many files. One area that could be improved is
the pipeline used to create `cleaned.json` and `clusters.json`, as both require several steps. We could automate the data processing more, which would make changes easier to implement.

Another key improvement that can be made is to develop a standard for coding practices. Although it may seem like a high coding standard is not key, but in reality having a consistent codebase is crucial to a project of this scale. For example, different members of our team had different ways of handling input/output for the code. To be able to maintain our repository well, a useful future extension would be to add setting input and output file paths through command line parameters.

Along those same lines, there are a few discrepancies in the languages used to write this codebase. There is Scala code exclusively in the initial processing step of `ArchiveSpark_WarcToJson.scala` to convert the .warc and .cdx files into a .json. Integrating this step into `jusTextCleaningBig.py` or into a separate Python script would be good for cutting down on the number of languages involved in the project, the number of steps required for processing, and the number of times files have to be retrieved and concatenated from distributed storage. There are also a few pieces of code, notably `Unit8.py` and the PGN-based extractive summarizer, that were written in Python 3.6 due to difficulties in getting the dependencies working with Python 2.7, which the rest of the code was written in to be able to run on the Hadoop cluster that was available during the course. It would be wise to streamline the code to a single version of Python and to reduce the number of different parallel processing workflows used (PySpark, multithreading, TensorFlow GPU, etc.) to decrease the overall complexity and number of dependencies in the project.

Due to the timeframe of the class, only pretrained models were used for abstractive summarization and the way that the model for PGN, in particular, was saved led to a restriction in the usable version of TensorFlow. We would
advise training the model from scratch (using the CNN/Daily Mail corpus or another corpus deemed to be more relevant) in a new version of TensorFlow and saving it using some mechanism other than checkpoints which do not have the same level of version dependence. This modernization to a new version of TensorFlow would make the code easier to run.

The Module 8 code is unfinished. In order to support the current attempted method of named entity extraction using spaCy, the early cleaning code would need to be modified to work without converting all code to lowercase. Then queries would have to be written that could extract relevant entities for the remaining slots in Table 13. If the code were to be finished, the next step for that module would be trying to ensure that it is generalizable enough to extract good entities from other collections about data breaches.

List of Files Submitted

FacebookBreachSummarization_CodeAndResults.zip:

This is the main repository of code and results which is available alongside this report. It should be sufficient to follow along with the User’s Manual and reproduce the results of this report. See the additional convenience files listing below.

➔ select_top_k_documents
  ◆ select_top_k_documents.py
    • Input: .json with all cleaned article text, one article per line (cleaned.json)
    • Output: .json with only the top k documents’ text (k can be specified in the global variable at the top of the file) (new_cleaned.json)
  ◆ extractive_summary_top5.txt
- An extractive summary performed on the top 5 documents
  - extractive_summary_top20.txt
    - An extractive summary performed on the top 20 documents
  - extractive_summary_top50.txt
    - An extractive summary performed on the top 50 documents
  - extractive_summary_top100.txt
    - An extractive summary performed on the top 100 documents
  - extractive_summary_top200.txt
    - An extractive summary performed on the top 200 documents
  - extractive_summary_top300.txt
    - An extractive summary performed on the top 300 documents
- new_cleaned.json
  - .json created with only the top k documents’ text

→ jusText_Cleaning_Code
  - ArchiveSpark_WarcToJson.scala
    - Input: .warc containing all of the web pages of the dataset
    - Output: .json format containing all the relevant information from the web pages (text, URL, timestamps)
  - jusTextCleaningSmall.py
    - Input: a .json containing full web page text for all of the documents, as produced by ArchiveSpark_WarcToJson.scala. Intended for use with relatively small data sets.
    - Output: a single .json containing relevant data and cleaned text for each web page, as well as an estimated publication date.
  - jusTextCleaningBig.py
    - Input: a .json containing full web page text for all of the documents, as produced by ArchiveSpark_WarcToJson.scala. Intended for use with larger data sets.
• Output: One .json file per worker node, each containing relevant data and cleaned text for a subset of web pages, as well as an estimated publication date. Must be concatenated into one file by something such as jsonFileConcat.py

◆ remove_hex_chars.py
  • Input: .json file containing all documents in ASCII encoding
  • Output: .json file containing all documents in UTF-8 encoding

◆ furthur_cleaning.py
  • Input: .json file containing all documents, one on each line
  • Output: .json file after deleting documents based on parameters

◆ remove_similar_documents.py
  • Input: .json file containing all documents, one on each line
  • Output: .json file where similar documents have been removed

◆ cleaned.json
  • .json with all cleaned article text, one article per line

◆ jsonFileConcat.py
  • Input: Path to folder containing all .json files (as produced by jusTextCleaningBig.py)
  • Output: Single .json file that concatenates all the files in the input folder

➔ Unit_1_most_freq_words
  ◆ Unit1.py
    • Input: .json with all cleaned article text, one article per line (cleaned.json)
    • Output: .csv with frequency counts of all tokens (words) in the dataset, not including stop words.

◆ eclipse_frequentwords.csv
Counts of all word occurrences (not including stop words) in all documents in the eclipse dataset, ordered by frequency.

**fb_big_frequentwords.csv**
- Counts of all word occurrences (not including stop words) in a subset of documents in the Facebook dataset, ordered by frequency.

**fb_small_frequentwords.csv**
- Counts of all word occurrences (not including stop words) in all documents in the Facebook dataset, ordered by frequency.

**Unit_2_WordNet_synset**
- **Unit2Lemmas.py**
  - Input: .json with all cleaned article text, one article per line (cleaned.json)
  - Output: Count of most frequent Synsets based on lemmatized words
- **Unit2Prototype.py**
  - Input: .json with all cleaned article text, one article per line (cleaned.json)
  - Output: Prints to console a count of most frequent Synsets based on POS-tagged tokens that have not been lemmatized.

**Unit_3_POS**
- **Unit3.py**
  - Input: .json with all cleaned article text, one article per line (cleaned.json)
  - Output: Four .csv files, one for each basic part of speech (adverbs, adjectives, nouns, verbs)
- **eclipse_common_adjectives.csv**
  - Counts of all adjectives in all documents in the eclipse dataset, ordered by frequency.
◆ eclipse_common_adverbs.csv
  • Counts of all adverbs in all documents in the eclipse dataset, ordered by frequency.
◆ eclipse_common_nouns.csv
  • Counts of all nouns in all documents in the eclipse dataset, ordered by frequency.
◆ eclipse_common_verbs.csv
  • Counts of all verbs in all documents in the eclipse dataset, ordered by frequency.
◆ facebook_common_adjectives.csv
  • Counts of all adjectives in all documents in the Facebook dataset, ordered by frequency.
◆ facebook_common_adverbs.csv
  • Counts of all adverbs in all documents in the Facebook dataset, ordered by frequency.
◆ facebook_common_nouns.csv
  • Counts of all nouns in all documents in the Facebook dataset, ordered by frequency.
◆ facebook_common_verbs.csv
  • Counts of all verbs in all documents in the Facebook dataset, ordered by frequency.

➔ Unit_4_Stemming_Lemmatizing

◆ Unit4_Stems.py
  • Input: .json with all the cleaned article text (cleaned.json)
  • Output: a list of most frequent stemmed words and their frequency counts (stem_output.txt)

◆ Unit4_Lemmas.py
  • Input: .json with all the cleaned article text (cleaned.json)
- Output: a list of most frequent lemmatized words and their
  frequency counts (lemma_output.txt)
  ◆ stem_output.txt
    - Frequency counts of all of the stemmed words in the Facebook
data set
  ◆ lemma_output.txt
    - Frequency counts of all of the lemmatized words in the
      Facebook dataset
  ➔ Unit_5_Named_Entities
    ◆ Unit5.py
      - Input: .json with all cleaned article text, one article per line
        (cleaned.json)
      - Output: A list of the most frequent named entities
  ➔ Unit_6_LSA
    ◆ Unit6.py
      - Input: .json with all the cleaned article jsons (cleaned.json)
      - Output: .csv file representing the topic x document matrix
  ➔ Clustering
    ◆ Lsa.csv
      - .csv file representing the topic x document matrix
    ◆ clustering.py
      - Input: .csv file representing the topic x document matrix
        (lsa.csv)
      - Output: .pickle file representing the clusters as a Python object
    ◆ Lsa_clusters.pickle
      - .pickle file representing the clusters as a Python object
    ◆ combine_clusters.py
      - Input: Array of .json files representing cluster contents and
descriptions
- Output: .json file containing the concatenated results along with cluster descriptions

◆ screeplot.py
- Input: .csv file representing the topic x document matrix (lsa.csv)
- Output: .png containing the screeplot to find optimal cluster number for k-means

◆ plot_clusters.py
- Input: .pickle file representing clusters and .csv containing the topic x document matrix
- Output: Plots of all the clusters

◆ clusters.json
- a .json with article text grouped into clusters (clusters.json)

◆ original_40topics_k-means_16clusters.txt
- Text file containing 30 headlines for each cluster with a configuration of 40 topics, k-means and 16 clusters

◆ reweighted_40topics_k-means_24clusters.txt
- Text file containing 30 headlines for each reweighted cluster with a configuration of 40 topics, k-means and 24 clusters

◆ original_40topics_k-means_16clusters_plot.png
- The plot of the first two dimensions against each other for each cluster with a configuration of 40 topics, k-means and 16 clusters

◆ reweighted_40topics_k-means_24clusters_plot.png
- The plot of the first two dimensions against each other for each reweighted cluster with a configuration of 40 topics, k-means and 24 clusters

◆ original_40topics_k-means_16clusters.json
- .json file containing all the documents in each cluster with a configuration of 40 topics, k-means and 16 clusters
  - reweighted_40topics_k-means_24clusters.json
- .json file containing all the documents in each reweighted cluster with a configuration of 40 topics, k-means and 24 clusters

→ Unit_7_extractive_summary
  ◆ Unit7.py
    - Input: .json with the top 100 sources of cleaned article text (new_cleaned.json)
    - Output: a single paragraph extractive summary based on best articles in dataset
  ◆ Unit7_clusters.py
    - Input: a .json with article text grouped into clusters (clusters.json)
    - Output: a single paragraph extractive summary based on only selected clusters of articles
  ◆ Unit7_split_cluster_summary.py
    - Input: a .json with article text grouped into clusters (clusters.json)
    - Output: a multi-paragraph extractive summary where different groups of clusters can be specified for each paragraph

→ Unit_8_9_template_summary
  ◆ Unit8.py
    - Input: a .json with the cleaned article text, one cluster per line, with the text and other fields stored in arrays (clusters.json). Note that it uses the order of clusters to determine where to
look for which tokens, and a different clustering method or data set would require changing these numbers.

- Output: Prints to the console a list of tokens filling each slot in the template, alongside the name of the slot being filled.
  (Incomplete)

◆ RegexResults
  - Unit8RegexPrototype.py
    - Input: a .json with the cleaned article text, one cluster per line, with the text and other fields stored in arrays (clusters.json). Note that it uses the order of clusters to determine where to look for which tokens, and a different clustering method or data set would require changing these numbers.
    - Output: A .csv file for every slot in the template summary, listing the matching tokens from the data set and their frequency of occurrence. (Incomplete)
  
- breachDates.csv
  - The potential dates of the data breach identified by Unit8RegexPrototype.py, ordered by frequency

- numUsers.csv
  - The potential numbers of users whose information was leaked, identified by Unit8RegexPrototype.py, ordered by frequency

- senDates.csv
  - The potential dates of the senate hearing identified by Unit8RegexPrototype.py, ordered by frequency

- websites.csv
The potential websites/companies/organizations responsible for the breach identified by
_Unit8RegexPrototype.py, ordered by frequency

➔ Unit_10_abstractive_summary

◆ AbTextSumm
  ● dataset_to_sentences.py
    ○ Input: .json file containing all documents in our dataset
    ○ Output: a text file with each line containing an individual sentence
  ● fb_sentences.txt
    ○ a text file with each line containing an individual sentence
  ● fb_language_model.arpa
    ○ Arpa file containing the language model for the Facebook dataset

◆ FastAbsRL
  ● Make_summary.py
    ○ Input: .json file representing documents in each cluster (clusters.json)
    ○ Output: Command line interactive output where you can generate summaries for clusters you want based on description

◆ PGN
  ● clean_data_for_pgn
    ○ json_to_story.py
      ◆ Input: a .json with the cleaned article text, one cluster per line, with the text and other fields stored in arrays (clusters.json).
Output: a directory of .story files, one per article, and two files all_urls.txt and url_metadata.txt which are keys to the identity of each .story file.

○ txt_to_story.py
  ◆ Input: a .txt with article text, one per line
  ◆ Output: a directory of .story files, one per article, and file all_urls.txt which is a key to the identity of each .story file.

○ txts_to_story.py
  ◆ Input: a directory of .txt files, one article per file
  ◆ Output: a directory of .story files, one per article, and file all_urls.txt which is a key to the identity of each .story file.

○ make_datafiles.py
  ◆ Input: a directory of .story files, one per article, and file all_urls.txt produced by one of the previous converters
  ◆ Output: a .bin file (and series of chunked .bin files, if you would like to use those for improved performance) containing the text and article ID for each document to be summarized. This is the input for run_summarization.py

○ README.md
  ◆ Some more detailed information on code sources for this unit and expected input/output/run instructions.

• vocab
  ○ A vocabulary file, based off of the CNN/Daily Mail dataset, which is used as input for the abstractive
summarizer. You most likely do not need to generate your own file for your data set, as pointer-generator is capable of extracting words not existing in its vocabulary.

- **abstracts_to_json.py**
  - Input: The directory of .txt files containing summaries (one per file) generated by the abstractive summarizer, the *clusters.json* file passed to *json_to_story.py*, and the *url_metadata.txt* generated by it. (Note that these are all order-sensitive, so be sure to not get files confused.)
  - Output: A .json file of the same format as *clusters.json* with an additional field “abstracts” that contains an array of individual document summaries for each cluster, in the same order as the URL, text, and other arrays.

- **abs_to_ext_by_cluster.py**
  - Input: *abstracts.json*, as output by *abstracts_to_json.py* or as included in this folder
  - Output: A modification of *Unit7_clusters.py*, this code outputs a single extractive summary made from the single-document abstractive summaries. It prints the summary to the console.

- **Pointer_generator**
  - **run_summarization.py**
    - Input: a test.bin file or series of chunked .bin files of the type created by *make_datafiles.py*, a vocab file of the type included with this project, and a directory containing the checkpoint files for the pretrained model (as is included with this project
in
FacebookBreakSummarization_PretrainedPGN.zip or through [6]).

◆ Output: a series of text files, one per input document, with the generated abstractive summaries.

◆ Note that all other files included have functions that are used by this script and will not be called directly from the command line.

○ __init__.py
○ attention_decoder.py
○ batcher.py
○ beam_search.py
○ data.py
○ decode.py
○ inspect_checkpoint.py
○ model.py
○ util.py
○ README.md

● abstracts.json

○ The output of the abstractive summarizer when run on each document individually, re-ordered and inserted into clusters.json

→ ROUGE_Evaluation

◆ eval.py

● Input: Predicted summary and gold standard summary as .txt files
● ROUGE scores and percentage of entities covered by the predicted summary
Supplementary code files:

These are larger archives which are not strictly necessary to recreate the results from this report, but which are provided for convenience. They are either time-consuming to compile, only available from non-stable sources (e.g., a google drive link), or were modified heavily to work with our data. Note that these are large files that will likely take a long time to download and/or unzip.

➔ FacebookBreachSummarization_PySparkExtPkgs.zip
   ◆ A .zip archive containing the source code for the external packages necessary to run FacebookBreachSummarization_CodeAndResults.zip/jusText_Cleaning_Code/JusTextCleaningBig.py and other pySpark code.
   ◆ Must be passed onto the worker nodes when submitting a PySpark job.
   ◆ Important packages included:
      ● jusText
      ● NLTK
      ● num2words
      ● word2number
      ● dateutil
      ● BeautifulSoup4
   ◆ This archive can be recreated as such:
      ● Make a clean Anaconda environment running the same version of Python as your worker nodes.
      ● Use pip to install the needed packages.
      ● Copy ~/.conda/envs into a folder named external_pkgs
      ● Zip external_pkgs

➔ FacebookBreackSummarization_FastAbsRLFork.zip
◆ A clone of the original FastAbsRL repository [4], modified to work with our corpus.

➔ FacebookBreachSummarization_FilesForAbsSum.zip

◆ facebook/story_files

  ● A directory of .story files, where each contains the raw text of a single article about the data breach, as output by a script like FacebookBreachSummarization_CodeAndResults.zip/Unit_10_abstractive_summary/PGN/clean_data_for_pgn/*_to_story.py

  ● Could be used along with all_urls.txt as the input to any version of the make_datafiles.py script.

◆ facebook/all_urls.txt

  ● A text file containing one URL per line, where each is an article present in fb_story_files. Since those file names are hashes of the URLs, this is later used to identify the files. Necessary input for make_datafiles.py.

◆ facebook/url_metadata.txt

  ● A version of all_urls.txt which also includes the cluster number for each document.

◆ make_datafiles.py

  ● A script which tokenizes the .story files in facebook/story_files and converts them to test/train/validation sets for a pytorch-based summarizer such as FastAbsRL.

  ● Input: a directory of .story files, one per article, and file all_urls.txt produced by one of the previous converters (FacebookBreachSummarization_CodeAndResults.zip/Unit_10_abstractive_summary/PGN/clean_data_for_pgn/*_to_story.py)

  ● Output:
- A series of .story files containing the tokenized text and article ID (hashed URL) for each document to be summarized.
- A directory finished_files containing .tgz files with test/train/validation data. The .tgz files need to be unzipped for use in the next step. These are the input for the code in `FacebookBreachSummarization_FastAbsRLFork.zip`.

→ FacebookBreachSummarization_PretrainedPGN.zip

- A .zip file containing the pretrained pointer-generator model for TensorFlow 1.2.1, as found at [6]. Must be unzipped and loaded by `FacebookBreachSummarization_CodeAndResults.zip/Unit_10_abstractive_summary/PGN:pointer_generator/run_summarization.py` in order to generate abstractive summaries.

- This is simply a stable copy of the model provided for convenience and longevity. [6] contains a link to a Google Drive download, which is where this was obtained.
Lessons Learned

Timeline

In the first month of the course we were able to split up the work between the five of us and finish modules 1 through 4. After the simpler units were done, we needed to do slightly more planning to keep us on track to finish the bigger units and summaries on time.

The Gantt chart in Figure 9 was created in early October to stay on track for the last few months of the semester. While we had originally hoped we could quickly finish document clustering, extractive summarization, and template summarization (Units 6-9) to focus on training our own deep learning-based abstractive model, we found that these tasks took longer than expected, many being finished about two weeks later than we had hoped and with the template summarization never getting finished to the extent we had envisioned. We realized that our plans were overly-ambitious, especially as no one on our team had prior experience with deep learning models, and in the end finished the abstractive unit by using pre-trained models and the guidance of classmates. We do not regret the extra time spent tuning the document classifier and extractive summarizer, as these units came together to produce the best summary we generated in the class.
Figure 9. Original Gantt chart for 10/8-11/16, at which point we had originally hoped to have all code done in order to spend the rest of the semester writing.

**Challenges Faced**

One of the other challenges we faced was finishing our overall summaries on time. We should have allocated more time to this process. Due to the size of our collection, whenever a code change was made, it took a while to run and get
results from it. If we had to do this project over again, we would have started thinking about the end goal sooner to avoid being rushed at the end.

We struggled in general with the breadth of topics covered in the corpus, especially when it came to deciding which information was relevant. There were multiple responses from governments and groups, and our data set contained articles about more than one recent Facebook security incident. We decided to leave out many smaller clusters with information on the international response, and there nothing about Vote Leave or the GDPR fallout in the final summaries as a result. There is also very little information about these other Facebook incidents in our final summaries, but what was there did occasionally lead to some issues as bad sequencing would occasionally lead to text about the Facebook Messenger call log breach being jumbled up with information about the major Cambridge Analytica scandal.

We also struggled for the early units with how many important terms in the corpus were proper nouns unrecognized by stemmers, lemmatizers, and WordNet. These tools were not used very much in later units for this reason, and any method which ignored out-of-vocabulary words was essentially useless for our dataset.

Another point of struggle for our team was figuring out the most optimal settings for performing clustering. Since most of us were not knowledgeable on the best kinds of clustering techniques to apply, we struggled with smartly reducing our options. Finally we just ended up trying everything and manually figuring out the best approaches, but this would have been far more refined if we knew more about each clustering algorithm.

**Solutions Developed**

To meet the overall goal of this course, which was to create a summary of an event given a collection of web pages about it, we ultimately came up with the
following solutions:

An extractive summary consists of sentences that appear within the articles in our data collection. We created two versions of this summary: one summarizing all of the web pages at once and one summarizing individual clusters of articles based on topics, then adding those together (both can be found in Appendix A). Of the two summaries, the clustering based approach yielded better results. There was less repetitiveness and more variety in the areas of the event covered. Overall, we found that the cluster based extractive summary was our best summary.

An abstractive summary consists of newly-generated sentences that summarize the content that appears in our data collection. Three different abstractive summary libraries were tested, and within each we tried different methods of processing the input and output. One method used FastAbsRL and used an extractive summary from selected clusters to summarize, one method utilized extractive summaries of each document cluster as the inputs for PGN, and one method used PGN to summarize each individual document in the corpus and then made an extractive summary of the concatenated summaries for each cluster. The method utilizing the PGN abstractive summarizer on the extractive summaries yielded the best results out of the abstractive methods tried, with the best grammar, least repetitiveness, and more accurate facts. It is slightly less repetitive than the final extractive summary, but the second paragraph (covering the various responses to the data breach) does not flow as well and contains fewer complete sentences. The abstractive summaries can be found in Appendix B and more details on the methods utilized can be found in the section on Module 10.
Conclusions and Future Work

Future Work

A lot of work was done over the course of the semester on this project; however, we did have some ideas that we ran out of time to try.

One thing we might be able to do to improve our final abstractive summary is refining the process of converting from FastAbsRL’s single document summaries to a final abstractive summary. Theoretically, the results of FastAbsRL should be better than the results of PGN, but we did not spend as much time on converting results of FastAbsRL to a single final summary. We also wanted to attempt to utilize an abstractive summarization model that was designed to turn multiple documents into a single summary, such as tensor2tensor [35], but we did not have time to train the model by the time we were looking into abstractive methods.

We were interested in trying to replace WordNet with a word2vec metric for word similarity, trained on a corpus such as HackerNews articles that would have greater coverage of tech company names and modern jargon. Training a word2vec algorithm on our corpus could also have helped with semantic understanding of relationships between entities, such as CEOs of companies and companies that were related to each other [36], a large part of the information contained in the corpus.

Our summarization relied heavily on clustering, so we could also potentially improve the quality of both our extractive and abstractive summaries by refining the results of clustering. In creating the summaries, we found that some clusters seemed to overlap heavily with other clusters that were separated with our current best approach (e.g., there were two clusters about India, two clusters...
about the senate hearing), so investigating and correcting this might lead to cleaner clusters and better summaries.

Future work should use the techniques developed for this corpus to summarize other, similar corpora (about other data breaches, about other political events or scandals, a larger corpus about Facebook, etc.) to see how the methods hold up. Unfortunately, due to the heavy reliance on document clustering and manual selection of clusters, the code would not be directly applicable to a new data set without tuning. Further development of the template summary process and generalization to ensure that it would work on other collections about data breaches, therefore, would be useful.

Finally, we would have appreciated the opportunity to have a larger data set of articles about the solar eclipse to have worked with, as it would have been interesting to see more developed summaries about the event, to see what categories of articles were written about the eclipse, and to see whether code written to summarize one solar eclipse could be used directly for articles about a different eclipse. It would also be interesting to see the effect our approach with the Facebook Data Breach would have on the solar eclipse dataset and to see if our summarization techniques for the Facebook Data Breach dataset would also produce good summaries for the solar eclipse dataset.

Conclusions

This report presents implementations for various Natural Language Processing and automated summarization techniques utilizing a collection of web pages about the Facebook-Cambridge Analytica data breach as a corpus. Four main methods of generating complete grammatical English summaries were tried: (1) a purely extractive method that did not follow a logical progression, was highly repetitive, and was often vague (e.g., “The company” instead of Facebook) but
had the best ROUGE scores likely due to the high coverage of named entities; (2) an extractive method that consisted of concatenating summaries based on individual document categories which was much more readable but did not have the same level of detail about organizations; (3) a mixed method that generated a single extractive summary from pointer-generator abstractive summaries of individual documents, which was slightly shorter and less repetitive than (2) but mainly due to lower readability; and (4) a mixed method that utilized the workflow from (3) but with FastAbsRL, which was more highly abstractive but had overall low coherency.

The results from (2), the extractive cluster-based method, were subjectively identified as best, but in removing overly repetitive sentences, names such as “Christopher Wylie”, “Alexander Nix”, and “Strategic Communication Laboratories” that often appeared in the same context were removed, resulting in lower ROUGE scores. This shows that when attempting to generate a single summary from a large body of documents, using a metric such as ROUGE to compare to a single gold standard rewards the presence of specific words, and does not take into account important factors like readability or coherency. Thus, we would suggest an iterative design process for this sort of task with frequent human assessment of summary quality, when focusing on a single event collection.
Acknowledgments

This project would not have been possible without the NSF-funded Global Event and Trend Archive Research (GETAR) project, IIS-1619028, used to create our collections.

We would like to thank first and foremost, Dr. Edward Fox (fox@vt.edu).

We would also like to thank the Teaching Assistant for this course, Liuqing Li (liuqing@vt.edu).

We would like to thank our fellow classmates for the sharing of their ideas, workflows, and in some cases code.

And, finally, all of the presenters and consultants who took time to help us as well:

   Xuan Zhang (xuancs@vt.edu)

   Srijith Rajamohan (srijithr@vt.edu)

   Michael Horning (mhorning@vt.edu)

   Matthew Ritzinger (mritzing@vt.edu)

   Ziqian Song (ziqian@vt.edu)
References


Appendices

Appendix A: Extractive Summaries

This appendix includes the extractive summaries that resulted from the three major approaches tried in Module 7. All summaries are comprised entirely of sentences that were present in the corpus of articles about the Facebook data breach, produced by the Gensim extractive summarizer [18] and cleaned up to fix capitalization and punctuation for readability. See Module 7 for more information on the creation of these summaries.

A.1 Summary with Similar Sentences Removed

Share Cambridge Analytica, a data analysis firm that worked on president Trump’s 2016 campaign, and its related company, Strategic Communications Laboratories, pilfered data on 50 million Facebook users and secretly kept it, according to reports in the New York Times, alongside the Guardian and the Observer.

San francisco Facebook has suspended Cambridge Analytica as it investigates whether the Donald Trump-connected data analysis firm failed to delete personal data that the social network says it improperly obtained from users, as many as 50 million, according to an explosive new report.

Over the weekend, the New York Times and the Observer of London posted a blockbuster investigative piece revealing that Cambridge Analytica, the firm brought on by the Trump campaign to target voters online, used the data of tens of millions of people obtained from Facebook without proper disclosures or permission.
While at Cambridge University, Kogan formed a company in the U.K. called Global Science Research, which created an app called 'thisisyourdigitallife' that offered Facebook users personality predictions, in exchange for accessing their personal data on the social network and more limited information about their friends -- including their "likes" -- if their privacy settings allowed it.

A.2 Overall Summary
The data firm suspended its Chief Executive, Alexander Nix (pictured), after recordings emerged of him making a series of controversial claims, including boasts that Cambridge Analytica had a pivotal role in the election of Donald Trump. This meant the company was able to mine the information of 87 million Facebook users even though just 270,000 people gave them permission to do so. And now, thanks to a whistleblower and two stunning reports in The Observer and The New York Times, we know that one of those developers siphoned data on more than 50 million Facebook users and shared them with the Trump campaign’s voter targeting firm, Cambridge Analytica, a company that has bragged it has psychological profiles on 230 million American voters, which it uses to target people online with emotionally precise digital messaging to influence elections. The demands came in response to news reports Saturday about how the firm, Cambridge Analytica, used a feature once available to Facebook app developers to collect information on 270,000 people and, in the process, gain access to data on tens of millions of their Facebook friends, few, if any, of whom had given explicit permission for this sharing. It’s the latest fallout from a controversy involving Cambridge Analytics, a company that reportedly bought data obtained by Facebook and used it to create voter profiles which were then reportedly shared with the Trump and pro-Brexit campaigns 6:53 "recent media reports regarding the use of personal information posted on Facebook for political purposes raise serious privacy concerns," privacy commissioner Daniel Therrien said in the statement. On Friday night, as Americans began settling into
the weekend, Facebook dropped a pretty substantial piece of news: the company said in a detailed blog post that it had suspended from its platform the political-data firm Cambridge Analytica, which worked with the Trump campaign during the 2016 election, after learning the company hadn’t deleted Facebook user data it had obtained in violation of the social networks policies. Carol Davidsen, who worked as the media director at Obama for America, claimed Obama campaign mined millions of people’s information from Facebook she said that Facebook was surprised at the ease with which they were able to ‘suck out the whole social graph’ but the firm never tried to stop them when they realized what was doing, and even told them they’d made a special exception for them they ‘were very candid that they allowed us to do things they wouldn’t have allowed someone else to do because they were on our side,’ she tweeted. Davidsen said that she felt the project was ‘creepy’ - ‘even though we played by the rules, and didn’t do anything I felt was ugly, with the data’ Davidsen posted this in the wake of the uproar over Cambridge Analytica, and their mining of information for the Trump campaign Facebook allowed the Obama campaign to access the personal data of users during the 2012 campaign because they supported the democratic candidate according to a high ranking staffer. Grewal said that the social networking giant learned earlier in the week that London-based Cambridge Analytica, used by campaigns to strategically target personalized political messages, and its parent company Strategic Communication Laboratories (SCL), had misused data collected on 270,000 Facebook users.

A.3 Cluster Approach Summary

Tom Pahl, the acting director of the Federal Trade Commission’s Consumer Protection Division, wrote in a statement Monday morning that the agency is investigating Facebook’s privacy practices a week after news broke that the Trump campaign’s political-data firm, Cambridge Analytica, inappropriately
obtained data on more than 50 million Facebook users and then allegedly lied about deleting it. Facebook is attempting to do a face-saving act following severe criticisms against it so that it is able to maintain its user base and therefore the flow of advertisements and advertisers and investor. The largest social media platform in the world is facing close scrutiny of its privacy policies and actions both in the U.S. and the U.K. Last week there were allegations against Facebook that it did nothing to prevent the use of personal data of approximately 50 million Americans by British consultancy Cambridge Analytica which allegedly had misused the data during the 2016 Presidential elections in the U.S. The firm was appointed to assist President Donald Trump during the campaign. This weekend, a man named Christopher Wylie spoke with the New York Times about a consulting company he founded called Cambridge Analytica that, according to him, developed Facebook ads for the Trump campaign with the help of Steve Bannon and data stolen from the pages of 50 million Facebook users (including personal details, rather than passwords or private information).

Facebook ended the day down nearly 7 percent, to US$172.56 making it the worst performing stock in the S&P 500, as the company sought to stem the damage from media reports that Cambridge Analytica, the U.S. data-mining arm of a Britain-based research firm, had improperly accessed personal details from nearly 50 million Facebook users to help Trump campaign advisers target political ads during the 2016 election. Calls for probe of misappropriation of the private information of tens of millions of Americans. Former Cambridge Analytica employee Chris Wylie said the company used information to build psychological profiles so voters could be targeted with ads. Wylie criticized Facebook for facilitating the process, saying it should have made more inquiries when they started seeing the records pulled a collection of powerful U.S. senators are demanding that Facebook explain how a third-party firm with ties to the Trump campaign was able to gain access to data on 50 million of its users. Washington revelations that a political data firm may have gained access to the
personal information of as many as 50 million Facebook users drew new bipartisan calls on Capitol Hill Monday for Facebook CEO Mark Zuckerberg and the heads of other social media companies to answer questions from Congress.
Appendix B: Abstractive Summaries

This appendix includes the abstractive summaries that resulted from the three most successful approaches tried in Module 10. All summaries were created by some combination of extractive and abstractive methods in order to deal with the large volume of articles in the Facebook data breach corpus. See Module 10 for more information on the creation of these summaries.

B.1 PGN Approach 2

The company announced a suite of new, more intuitive privacy controls Wednesday morning, including a way to download and delete data, a redesigned settings menu, and additional shortcuts for controlling private information. Tom Pahl, the acting director of the Federal Trade Commission’s Consumer Protection Division, wrote in a statement Monday morning that the agency is investigating Facebook’s privacy practices a week after news broke that the Trump campaign’s political-data firm, Cambridge Analytica, inappropriately obtained data on more than 50 million Facebook users and then allegedly lied about deleting it. Facebook CEO Mark Zuckerberg apologized on Wednesday for the social media website's role in what he previously called the “Cambridge Analytica Situation” wherein the research firm allegedly accessed 50 million Facebook user profiles improperly. Christopher Wylie, who previously revealed that consultancy Cambridge Analytica had accessed the data of 50 million Facebook users to build voter profiles on behalf of Donald Trump’s campaign, said AggregateIQ (AIQ) had built software called Ripon to profile voters. Facebook has announced new controls, privacy shortcuts, and tools to delete Facebook data but said these were in the works before the Cambridge Analytica scandal exploded. The invitation asking Zuckerberg to answer questions at an April 10 hearing comes as the Federal Trade Commission confirmed it’s investigating Facebook’s privacy practices after reports the company allowed political consulting firm
Cambridge Analytica to harvest 50 million users’ data. Analyst: “the risk here is that Facebook is paramount to the future of this company.” Facebook shares 6 percent and were on track for their worst day in more than three years on reports that a political consultancy worked on president Donald Trump’s campaign gained inappropriate access to data on more than 50 million users. Lawmakers in the United States, Britain, and Europe have called for investigations into media reports that political analytics firm Cambridge Analytica had harvested the private data on more than 50 million Facebook users to support Trump’s 2016 presidential election campaign.

B.2 PGN Approach 3

Peter Eavis lawmakers want answers on Facebook’s latest controversy, lawmakers in the U.S. and Britain want Mark Zuckerberg to explain how Cambridge Analytica, the political data firm founded by Steve Bannon and Robert Mercer, harvested private information from over 50 million user profiles. The company has suspended U.K.-based Cambridge Analytica amid suggestions over the weekend that the British firm had lied about deleting user data that it had gained through the use of a psychology-test application posted on facebook.com.

The Guardian, which was the first to report on U.S. elections, in late 2015, noted that the company drew on research spanning tens of millions of Facebook users, harvested largely without their permission. While it is unclear how many responses Global Science Research obtained through Mechanical Turk and how many it recruited through a data company, all five of the sources interviewed by the intercept confirmed that Kogan’s work on behalf of SCL involved collecting data from survey participants networks. The dossier includes a letter from Facebook’s lawyers, dated August 2016, in which he was asked to destroy data collected by GSR.
Cambridge Analytica received user data from Aleksandr Kogan. Agency says EI lawmakers “will investigate fully, calling digital platforms to account” in the UK, Damian Collins, the chair of Parliament's committee overseeing digital matters.

Zuckerberg is expected to testify in the coming weeks before the House Energy and Commerce Committee. One of the panels, the Senate Judiciary Committee, also asked the leaders of Google and Twitter to join him at its April hearing. Last week, Facebook officials appeared unprepared and uncertain during briefings with congressional staff. Mark Warner, the chief executive of the Intelligence Committee, has also requested that Zuckerberg, along with other tech CEOs, testify in order to answer questions about Facebook's role in the 2016 election.

During Tuesday's committee hearing, Wylie suggested Facebook may have been aware of the large-scale harvesting of data carried out by Cambridge Analytica's partner GSR even earlier than had been previously reported.

Damian Collins, chairman of the Digital, Culture, Media and Sport Committee, also accused the chief executive of deliberately misleading parliament.

The FTC investigation is connected to a settlement the agency reached with Facebook in 2011 after finding that the company had told users that third-party apps on the social media site, like games, would not be allowed to access their data. The demands came in response to news reports saturday about how the firm, Cambridge Analytica, used a feature once available to Facebook app developers to collect information on 270,000 people.
50 million Facebook profiles harvested for Cambridge Analytica in major data breach the data analytics firm that worked with Donald Trump's election team and the winning brexit campaign. Facebook users in developing techniques to support president Donald Trump's 2016 election campaign. Data analytics firm Cambridge Analytica harvested private information from 50 million. Facebook suspends Trump campaign data firm Cambridge Analytica. San : Facebook is suspending the Trump-affiliated data analytics firm Cambridge Analytica. He says Facebook and Cambridge Analytica have been implicated in a massive data breach. Facebook : social media company had secretly harvested profile data belonging to 50 million users. Facebook says it's suspended Cambridge Analytica, a data firm. Facebook is suspending the Trump-affiliated data analytics firm Cambridge Analytica. Facebook says it has suspended the account of Cambridge Analytica. Facebook's Mark Zuckerberg, CEO of Facebook, says it's not a breach of privacy. Facebook is reeling from the private data of more than 50 million users. He says the data breach was one of the largest in the history of Facebook. He says Facebook is being called to account for how it handled Cambridge Analytica’s misuse. Facebook needs to explain its role in data misuse of social media. Facebook suspended Cambridge Analytica over the weekend. Trump : Facebook accounts are 185 times larger than the company. Cambridge Analytica improperly used data from some 50 million Facebook users. Facebook set to be grilled by lawmakers over Trump-linked data firm Cambridge Analytica. he says Facebook is trying to protect users from potential privacy violations. The lawsuit alleges that Facebook did nothing to protect the privacy of users. Facebook says it was `failing to ensure that data was protected’. Facebook CEO mark zuckerberg says the Facebook data breach was `nothing short of horrifying’. Facebook has launched an investigation of the social networking
company. Facebook shares spiral as U.S. agency launches privacy investigation stocks. Facebook CEO Mark Zuckerberg says the social media giant has violated their privacy. Playboy quits Facebook over data privacy scandal. The U.S. Federal Trade Commission is going to investigate Facebook’s actions in protecting privacy. EU issues ultimatum for Facebook to answer data scandal questions. Facebook and Cambridge Analytica are facing a privacy violations class action lawsuit.

Cambridge Analytica improperly used data from some 50 million Facebook users. Facebook CEO Mark Zuckerberg says the social media giant has violated their privacy. Facebook says it’s suspending Cambridge Analytica after finding data privacy policies. Facebook says the data of more than 50 million users were inappropriately used by U.S. President Donald Trump. Britain is concerned about allegations that the data firm Cambridge Analytica exploited Facebook data to use millions of peoples profiles without authorization. Facebook suspended Cambridge Analytica, a data firm that helped President Donald Trump win the 2016 election. Facebook has suspended Cambridge Analytica, a data company that helped Donald Trump win the presidential election. Australia is investigating whether local personal information was exposed in the Facebook data breach the Australian Information and Privacy Commissioner. Data analytics firm Cambridge Analytica harvested private information from 50 million Facebook users in developing techniques to support President Donald Trump’s 2016 election campaign. Data analytics firm Cambridge Analytica harvested private information from 50 million Facebook users in developing techniques to support President Donald Trump’s 2016 election campaign.

Facebook says it has hired the forensics firm to conduct an audit of Cambridge Analytica’s systems. Facebook says it’s suspending Cambridge Analytica after finding data privacy policies. Facebook says the data of more than 50 million
users were inappropriately used by US President Donald Trump. Facebook hires
digital forensics firm to investigate Cambridge Analytica. The lawsuit alleges
Cambridge Analytica deceived millions of Illinois Facebook users. Facebook says
it is hiring a digital forensic firm to conduct an audit of Cambridge Analytica.
Britain is concerned about allegations that the data firm Cambridge Analytica
exploited Facebook data to use millions of people's profiles without authorization.
Analytics firm reportedly used data on millions of Facebook users to support the
Trump campaign. Facebook suspended Cambridge Analytica, a data firm that
helped president Donald Trump with the 2016 election. EU lawmakers will
investigate whether the data of more than 50 million Facebook users has been
misused.

San: Facebook is suspending the Trump-affiliated data analytics firm Cambridge
Analytica. He says Facebook and Cambridge Analytica have been implicated in a
massive data breach. Facebook says it's suspended Cambridge Analytica, a data
firm. Facebook is suspending the Trump-affiliated data analytics firm Cambridge
Analytica. Facebook says it has suspended the account of Cambridge Analytica.
British MPs have called on Facebook CEO to testify on the Cambridge Analytica
data scandal. Facebook CEO Mark Zuckerberg says he will not appear before a
U.K. parliamentary committee. Facebook CEO Mark Zuckerberg declined an invite
from the Parliament to come and testify over the current Cambridge Analytica
scandal. Mark Zuckerberg will testify in the U.K. over Facebook's Cambridge
Analytica scandal. British lawmakers want to question Facebook boss Mark
Zuckerberg over how millions of users got into the hands of Cambridge Analytica.
Appendix C: Gold Standard Summaries

This appendix contains two summaries that were written by students in the class for collections of events being automatically summarized. These summaries were both written so the best results of an automated summarization workflow could be evaluated using a ROUGE methodology. See the ROUGE Evaluation section for more details.

C.1 Gold Standard for Maryland Shooting Corpus

The gold standard summary we wrote for Team 15 based on articles present in their corpus. Their dataset contained information regarding two shootings that occurred in Maryland, the Capital Gazette newsroom shooting and the Great Mills High School shooting. Information on the workflow used to create this summary is found in the Evaluation section of this report.

Key:

Blue indicates a detail present in few (<50) articles in the dataset.

Yellow indicates a detail which is a human judgment on the actual content of the articles or otherwise not present in the dataset

Capital Gazette shooting

On Tuesday, June 28, 2018 at 2:33pm, 38 year-old Jarrod Warren Ramos opened fire on the glass front door of the Capital Gazette newsroom in Annapolis, MD [37]. Of the 11 employees there that day, 5 were killed: John McNamara, Gerald Fischman, Rob Hiaasen, Wendi Winters, and Rebecca Smith [38]. Two more were injured. Ramos used a 12 gauge pump-action shotgun and smoke grenades, and barricaded the back door to the newsroom to prevent people from escaping [39,40]. One Capital Gazette employee, Wendi Winters, grabbed a trash can and a recycling bin and charged the gunman, but was killed [41]. Law enforcement responded within 60 seconds, and arrested the gunman, who was found hiding under a desk [42]. Approximately 170 people were evacuated from the building [42].
The Capital Gazette is a small local newspaper owned by the Baltimore Sun. In 2011, the Capital Gazette reported on a case in which Ramos was convicted of harassing a former high school classmate. Ramos sued Eric Hartley and Thomas Marquardt, employees of the Capital Gazette, for defamation in 2012, and the lawsuit was dismissed. Ramos began to make threats against the Capital Gazette. Editor Thomas Marquardt called the police to report the threats in 2013, but nothing came of it.

Ramos has been charged with 23 total charges including five counts of first-degree murder and was held without bail. Ramos was assigned public defender William Davis. Davis claims that the facial recognition methods used to identify Ramos when his fingerprints were slow to process were impermissible. Ramos has pled not guilty to all charges and Davis has indicated that they are considering a plea of not criminally responsible by reason of insanity.

Several individuals associated with the Capital Gazette received letters postmarked the day of the shooting with Ramos’s personal information and indicating an “objective of killing every person present”, apparently signed by Ramos. The Capital Gazette also reported receiving politically-motivated threats after the initial coverage of the incident.

The Capital Gazette continued publishing the paper on schedule despite their loss. Several crowdfunding campaigns were set up for families, victims, and survivors of the attack, as well as a journalism scholarship memorial fund.

Shootings like these have sparked a call for tighter gun control regulations from some. A Maryland law was passed in April which gives family members and law enforcement the ability to temporarily restrict firearm access for individuals believed to be a risk, but the law does not go into effect until October. Many question whether the law could have been used against Ramos to prevent the attack.
After saying his “thoughts and prayers” were with the victims, President Trump initially refused requests to fly the American flag half-mast, but reversed his decision the Tuesday after the shooting [49].

**Great Mills High School shooting**

On Tuesday, March 20, 2018 at 7:57 a.m., 16 year-old Jaelynn Willey was shot in the halls of Great Mills High School in Great Mills, Maryland [50]. The bullet that hit Willey also struck 14-year-old Desmond Barnes in the leg [50]. Willey was rushed to the hospital and declared brain dead, then taken off of life support two days later after being declared brain dead [51]. The shooter, Austin Wyatt Rollins, was a 17 year-old Great Mills student [51]. Rollins used a Glock semi-automatic handgun to carry out the attack [52]. The gun was legally owned by Rollins’ father, but it should not have been in Rollin’s possession since Maryland law states that one needs to be over 21 to carry a gun [50]. Following the shooting, Rollins wandered around the school. The school resource officer Deputy First Class Blaine Gaskill confronted Rollins and ordered him to drop his gun [53]. 31 seconds after the confrontation, Rollins and Gaskill fired their weapons simultaneously [50]. Rollins fatally shot himself in the head and Gaskill shot him in the hand [50].

The school was put into lockdown for several hours following the shooting [52]. Law enforcement arrived to the scene at approximately 8:00 a.m., but by then the situation had been contained [54]. 1440 students were evacuated to a reunification center in nearby school Leonardtown High School [52]. Classes were cancelled for the rest of the week [55].

Investigators say that Rollins had a previous relationship with Jaelynn Willey that had recently ended [52]. In the month before the shooting, the school had investigated a potential school shooting threat and determined that the threat was unsubstantiated [56]. This was also less than a week after the students had
participated in a nation-wide walkout to protest gun violence and the lack of gun regulations as well as show support for the victims of the Marjory Stoneman Douglas High School shooting [56]. It was the 17th American school shooting in 2018 [43]. Following the shooting, school nurses have been trained in how to react to a school shooter, how to triage wounded children, and how to apply a tourniquet [57].

C.2 Gold Standard for Facebook Data Breach Corpus
The gold standard written about the Facebook data breach by Team 12. Authors: Frank Wanye, Matt Tuckman, Joy Zhang, Fangzheng Zhang, and Samit Ganguli.

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 [58]. 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
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 whistle- blower 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 [59]. 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 [59].

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 [60].

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 [61]. 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 [59].