Aaron Schmitz ([schmitza1@hawkmail.newpaltz.edu](mailto:schmitza1@hawkmail.newpaltz.edu))

CPS493 – Introduction to Data Science

6 May 2019

Professor Min Chen

Final Report

THE NEWS HEADLINE SARCASM DETECTOR

Table of Contents

|  |  |
| --- | --- |
| Cover Page | p.1 |
| Principal Investigator | p.2 |
| Title of Project | p.2 |
| Mentoring | p.2 |
| Contents of Zipped Folder | p.2 |
| Introduction | p.3 |
| Background | p.5 |
| Approach and Implementation | p.5 |
| Experiment Results and Discussion | p.11 |
| Conclusion | p.14 |
| References | p.14 |
| Contents of Zipped Folder | p.14 |

**Principal Investigator**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task | Member 1 | Member 2 | Member 3 | Total |
| Introduction | Aaron Schmitz  [schmitz1@hawkmail.newpaltz.edu](mailto:schmitz1@hawkmail.newpaltz.edu) | | | 100% |
| Background | 100% |
| Implementation | 100% |
| Experiment Results and Discussion | 100% |
| Conclusion | 100% |
| Other contribution and explain | 100% |

**Title of Project**

THE NEWS HEADLINE SARCASM DETECTOR

**Mentoring**

Professor Min Chen, Department of Computer Science, SUNY-New Paltz  
[chenm@newpaltz.edu](mailto:chenm@newpaltz.edu)

**Contents of Zipped Folder**

* HeadlineSplitter.java - MapReduce function
* Main.java - Main program
* MainAccuracyTest.java - For testing accuracy
* WordCount.java - Standard Hadoop WordCount function
* CountSort.java - Modified WordCount for sorting
* mappedData - Mapped dataset
* reducedData - Mapped-reduced dataset
* allWords - All words from all headlines counted
* seriousWords - Words from serious headlines counted
* sarcasticWords - Words from sarcastic headlines counted
* sd.jar - Runnable .jar file to run the program!

**Introduction**

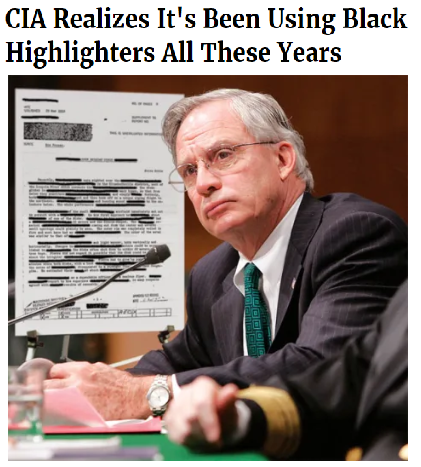
**Project Motivation**

I’ll let these images speak for themselves:



Every one of these four “news” headlines can be found on social media, but only *one* of the above images is “sarcastic” (i.e. a joke, written for comedic reasons – not *actual* news). If you ask me, most of those headlines each sound ridiculous (in their own way), but unfortunately, *three-out-of-the-four* are real! The least ridiculous headline—the one about unemployment—is the sarcastic one; written by *The Onion*, a satirical news website. All the others are real.

Sometimes the sarcastic/joke/satire news is easy to distinguish, like these, for example:

  
Both of these are jokes.

As seen in the first example, sometimes it’s not so easy to tell them apart, and somebody, somewhere out there, is going to fall for them. If one were to come across any of these articles on social media, they may take it at face value. Most people don’t have the time/knowledge/will to fact check everything they read on Facebook (and many will only read the headline), and the news is supposed to report true events. Currently, there’s a lot of pressure on social media networks (mostly Facebook) about their (mis)handling of information.

*Which is why all social media platforms should buy my project!*

**Aims and Objectives**

Make a program that can differentiate serious news from sarcastic news with reasonable accuracy!

**Report Structure**

My report uses the terms “serious” and “sarcastic” to refer to the news headlines. *Serious* will mean any news article that is meant to be taken seriously, such as real breaking news stories. *Sarcastic* will mean any news that is *not* meant to be taken seriously, it’s lighthearted, a joke. I will also provide many examples of my project in action, predicting whether a news headline is likely or unlikely to be sarcastic.

**Background**

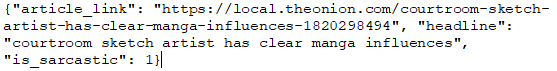
Recently, Facebook came under fire because of the *Cambridge Analytica* (CA) fiasco, when a political consulting firm used data science algorithms to target certain individuals with news recommendations to influence elections around the globe. There should probably be some fact-checking algorithms put in place to verify the reliability of news first, to prevent another *Cambridge Analytica*-esque conundrum in the future. Plus, a sarcastic text identifier would just be very useful in general.

**Approach and Implementation**

My program was written with the Eclipse IDE, coded in Java, with a Hadoop MapReduce cluster, and a Kaggle dataset.

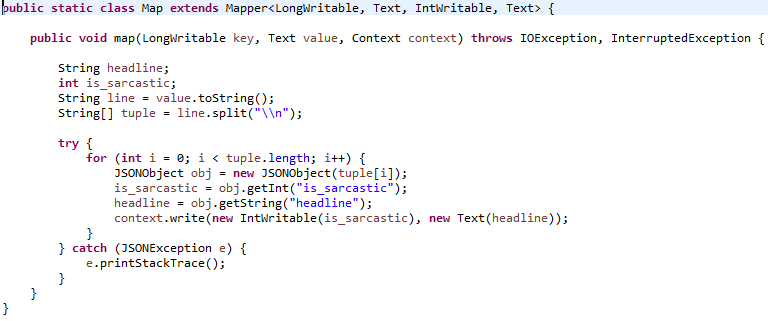
**First,** the dataset is a JSON format document containing 26,708 news headlines, their links, and if they are sarcastic or not. It was obtained from Kaggle, described as a dataset specifically made for anyone willing to try and make a news headline sarcasm detector (i.e. me).

Dataset Format:(article\_link, headline, is\_sarcastic)

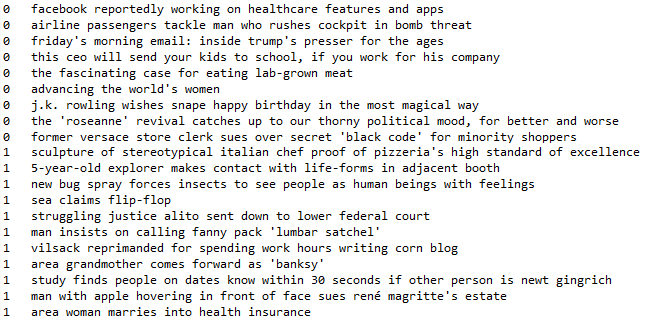


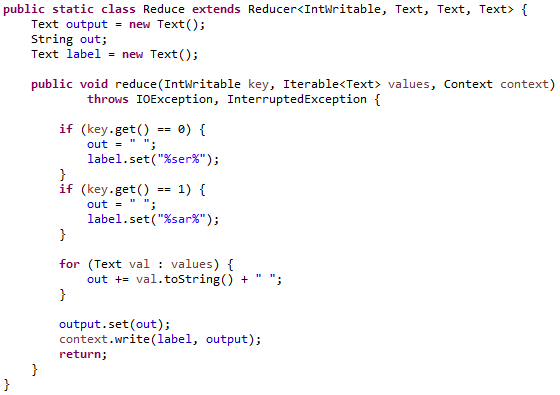
(article\_link is not used)

**Second,** the code is broken into two classes: *HeadineSplitter*, and *Main*. HeadlineSplitter contains the MapReduce function. As the name suggests, it splits the headlines and sorts them based on if they are sarcastic or not. HeadlineSplitter merely generates the data file for Main, and Main is where the actual “sarcasm detection” takes place.

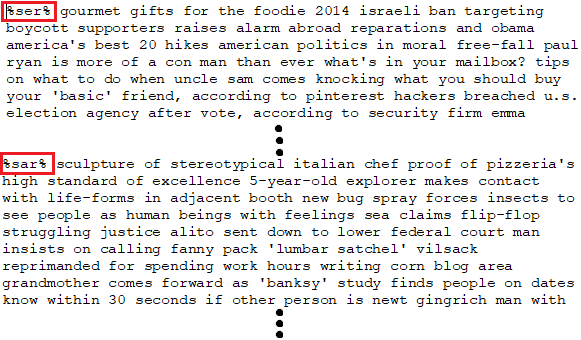
Map Function: (key Sarcasm\_Headlines\_Dataset.json, value Text) 

The Map function parses the .json file and separates the headlines into two groups: 0 and 1.  
0 denotes serious headlines; 1 denotes sarcastic headlines.

Mapped .json: (key is\_sarcastic, value headline) 

Reduce Function: (key is\_sarcastic, value headline) 

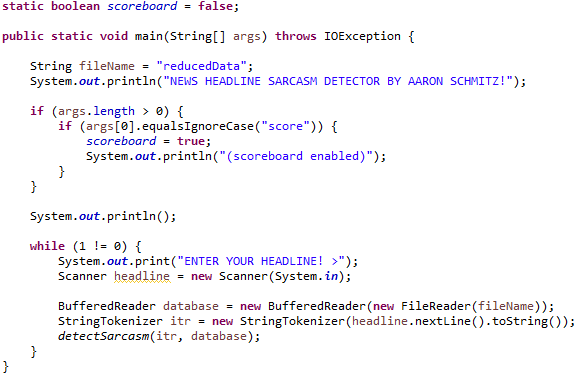
The Reduce function parses every line in the mapped .json, then concatenates them all onto a single line depending on if they were serious or sarcastic. Each line starts with a unique token so the final project can recognize where the lines begin/end (%ser% = serious; %sar% = sarcastic).

Reduced .json: (key is\_sarcastic, value headlines\_concatenated) 

It doesn’t look like it, but the file is only two lines: %ser% and %sar% are where they start. These are only small snippets of the map-reduced json. The original file contains over 26,000 headlines!

**By calculating the frequency of the words that appear in each group, the final product will be able to predict whether an inputted news headline is likely or unlikely to be sarcastic (ideally with a reasonable degree of accuracy).**

**Third,** the Main code is where the sarcasm detection happens.

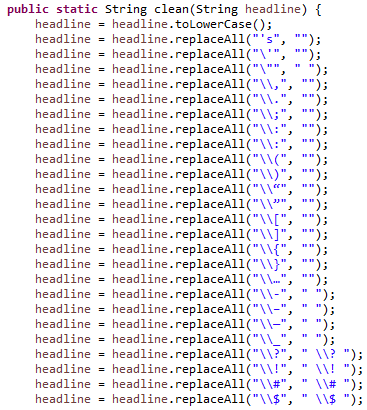
Main

The Main block opens the map-reduced file and asks for the user input. Once entered, the detectSarcasm method is called. This loop will continue forever, allowing for multiple runs. There is also an optional scoreboard which’ll display the total results at the end (off by default; can be enabled by typing “score” when running the program).

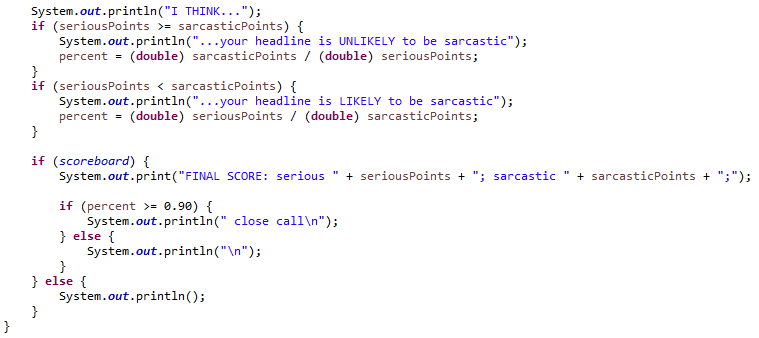
detectSarcasm(headline, database) 

The detectSarcasm() function parses the headline for each word, then counts up every occurrence of the word in the data file. Each time the parsed word(s) appears in the serious group, serious points go up by one, and each time the parsed word(s) appears in the sarcastic category, sarcastic points go up by one. **Note:** there’s a limit on how many points each word can give. Since my algorithm relies on grouping, it’s highly susceptible to outliers. In this case, any words that appear vastly more times in one category than the other (e.g. a word appears a thousand times in the serious group, but zero times in the sarcastic group), such a case would highly skew the results. **Also note:**  each word passes through a clean(word) method. This method filters out characters and common words.

clean(word)



Special characters like question marks and exclamation points aren’t removed, but rather separated so the program reads them as separate words. I figured those symbols were important for understanding the context of the headline, so the algorithm would be more accurate. Single quotes are removed, for “word’s” and “word” are the same. Also, all words are downshifted to lowercase.

detectSarcasm(headline, database) **PART 2**

Once the program is complete, it’ll make its prediction based on how many serious/sarcastic points were counted. If more serious points were counted, the program will print the message “…your headline is UNLIKELY to be sarcastic”, and the reverse if more sarcastic points were counted. If the scoreboard is enabled, the program will also print out the total number of points.

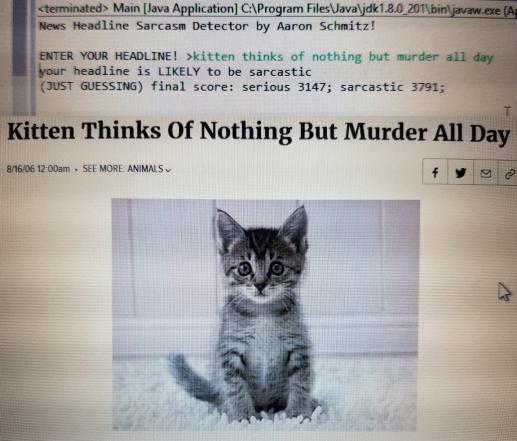
**Experiment Results and Discussion**

****

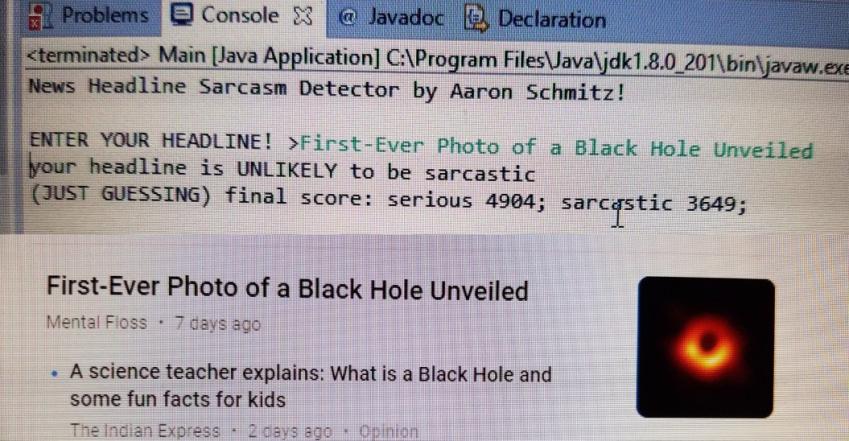
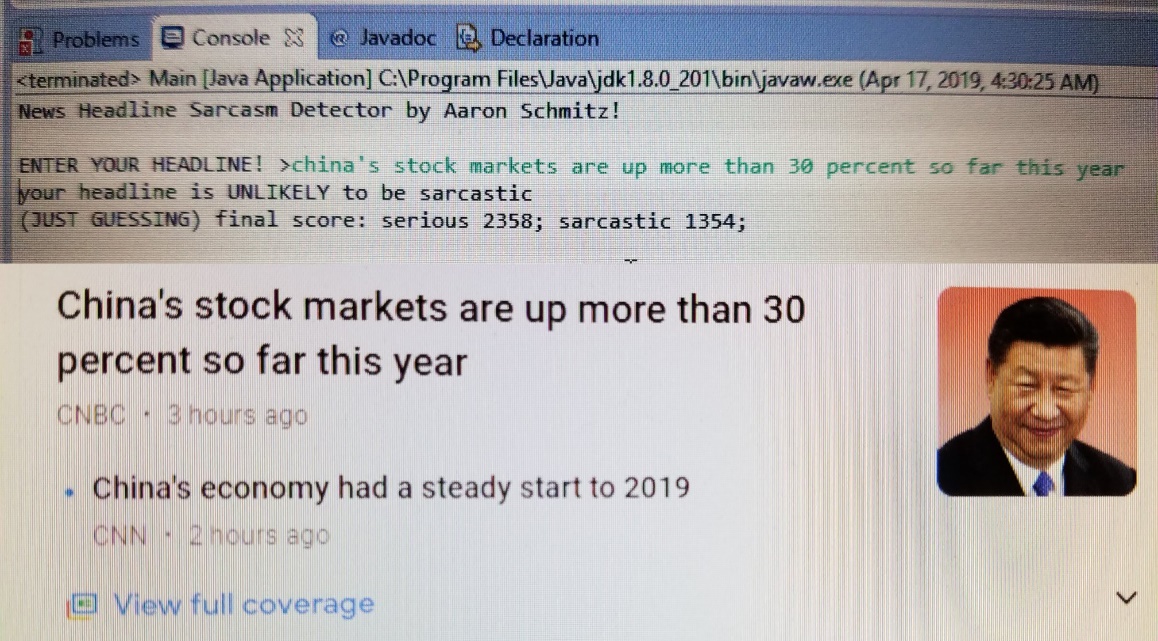
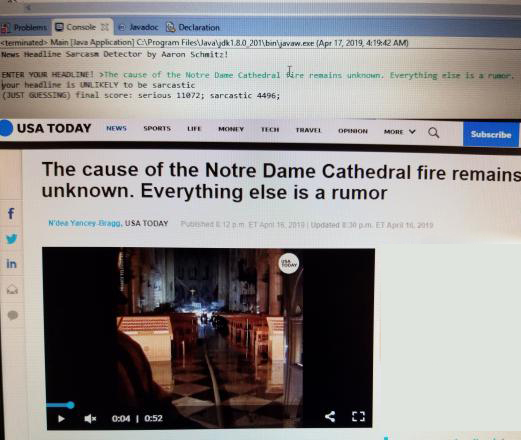
Despite being just a simple word counting algorithm, it’s surprisingly accurate. It correctly predicts the four examples from the introduction, despite their ridiculousness at face value. I ran an accuracy test for the program multiple times, using every headline from the original dataset, and counting the amount correct predictions. At maximum, I was able to get about an 80% accuracy rate:  (21,251 correct predictions out of 26,708)

**Some more examples:**

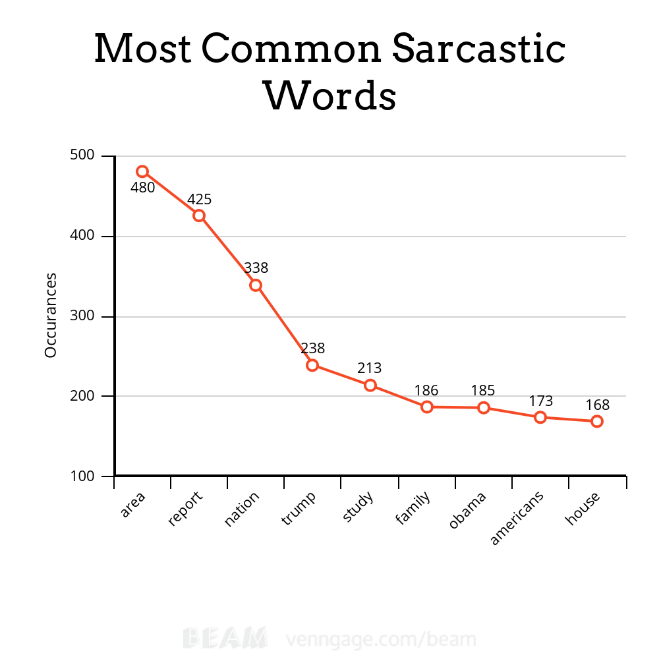
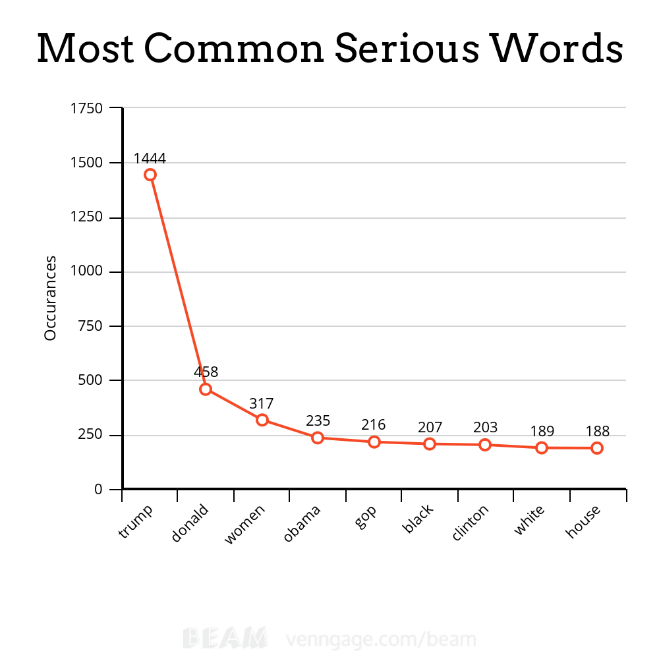
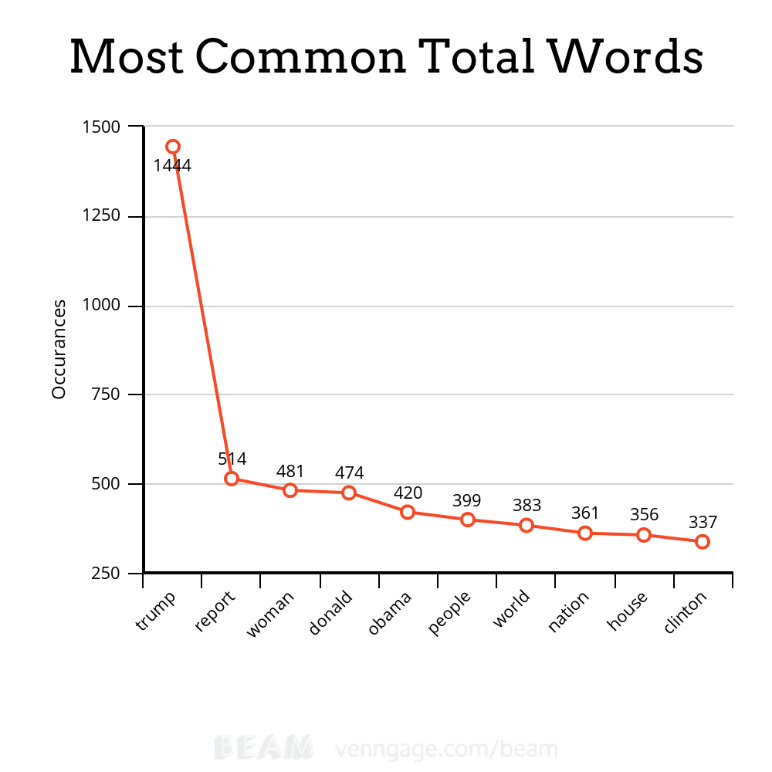
These two pulled off *The Onion*

(A headline from my project proposal)

Three headlines pulled off Google News



**Some charts** (numbers gathered via word count):



“Trump” is the biggest outlier in the dataset, and the entire reason for putting a limit on how many points each word could give. Any headline with “Trump” in it was guaranteed to always have (much) more serious points, hence the need for a point limit. Through testing, I found the point limit of 140 for both categories yielded the most accurate results. I would’ve tested more, but it takes a long time to check all 26,708 headlines. Too much data and not enough computing power *was* a recurring theme throughout the semester, after all.

**Conclusion**

I was very surprised with the accuracy of my project. Initially, I was quite skeptical that merely counting the frequency of words in a headline could accurately predict the tone of an entire article, but I counted on the sheer size of the dataset to deliver accurate results. I never expected to get anywhere near an 80% accuracy rate from such a simple program. I would’ve been happy with just a 51% accuracy rate.

With some machine learning, and/or help from a professional linguist, a much better program with a much higher accuracy rate could definitely be made. Perhaps one could find a use for the article link as well. Ideally one would be able to make a sarcasm detector that gets it right *every, single,* time. And then there would never be any more debate about the credibility of modern-day journalism!   
Overall, this was a fun project, and I’m happy with the results. This course made me *very* interested in data science, and just how far it can go. Data is the new oil.

**References**

**Zipped Folder** contains the dataset, map-reduced files, the code for HeadlineSplitter, Main, and MainAccuracyTest (for testing accuracy – takes a long time to do). The folder also contains lists of counted words used to make the graphs. Lastly, the folder also contains sd.jar, a runnable .jar file for running the program (it’s user interactive!)

**Sources**

Beam (for making graphs): <https://venngage.com/blog/beam/>

Cambridge Analytica: <https://www.nytimes.com/2018/03/19/technology/facebook-cambridge-analytica-explained.html>

Common Words: <https://en.wikipedia.org/wiki/Most_common_words_in_English>

Dataset: <https://www.kaggle.com/rmisra/news-headlines-dataset-for-sarcasm-detection>

Hadoop MapReduce: <https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html>

Google News: <https://news.google.com/?hl=en-US&gl=US&ceid=US:en>

The Onion: <https://www.theonion.com/>