# LAB2 REPORT

Data Aggregation, Big Data Analysis and Visualization

*Data Intensive Computing - CSE587*

**Video Link: [https://buffalo.box.com/s/lcdnfdwtou2n72zbd3sl59vgp6s8rlbp](https://buffalo.box.com/s/lcdnfdwtou2n72zbd3sl59vgp6s8rlbp)**

**Bhavik Gala**

UBIT Name: bhavikna
Person Number: 50248608

**Swati Nair**

UBIT Name: swatishr
Person Number: 50246994

## INTRODUCTION

This report contains the implementation details of Lab 2 i.e. how data was aggregated from Twitter and NY Times, how MapReduce was used to analyze the unstructured big data collected and how this data was visualized to get some meaningful insights from the data.

The search keyword that we used for data aggregation from Twitter and NY Times was **"gun control".**

## TASKS COMPLETED

1. Part 1: Worked on Python code examples given in Chapter 3, 4 and 5 from The Data Science Handbook.
2. Aggregating data related to search keyword "gun control" from multiple sources - Twitter and NY Times. Initially on small data i.e. 1000 tweets and 80 NYT articles.
3. Imported the given VM for Hadoop infrastructure and tested the basic commands to run the sample wordcount sample provided in the Hadoop.
4. Loaded the smaller data set into VM in two directories: one for Twitter and another one for NY Times
5. Executed the MapReduce word count Python program on each of the datasets which outputs a list of words and its count (i.e. the number of times the words occur in the tweets/articles)
6. Visualized these outputs as word clouds using D3.js in an HTML webpage
7. Repeated the above steps for whole dataset (over 50000 tweets and over 800 articles).
8. Similar to step 6, visualized the outputs for large dataset as word clouds
9. Analyzed the co-occurrence for the top ten words in both the datasets and collated the co-occurrences <word, co-occurring word> using MR

## WORKFLOW

**Data Aggregation:**

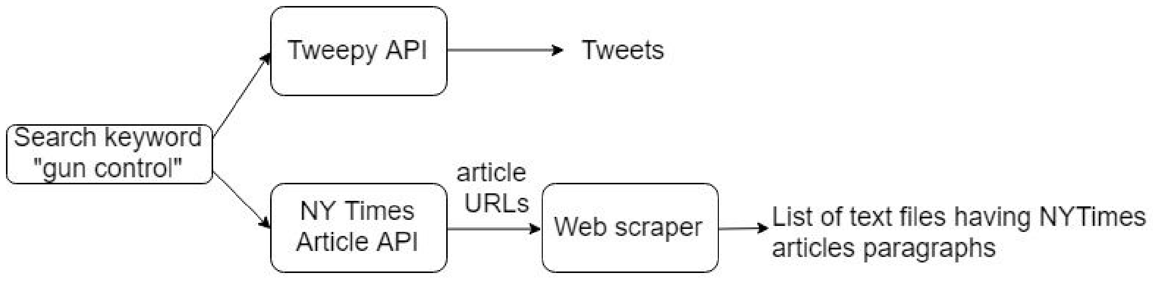

Tweepy API
Tweets
Search keyword "gun control"
NY Times Article API
article URLs
Web scraper
List of text files having NYTimes articles paragraphs

fig (a) Data Aggregation

**Mapper-Reducer and Word Cloud Visualization flow:**

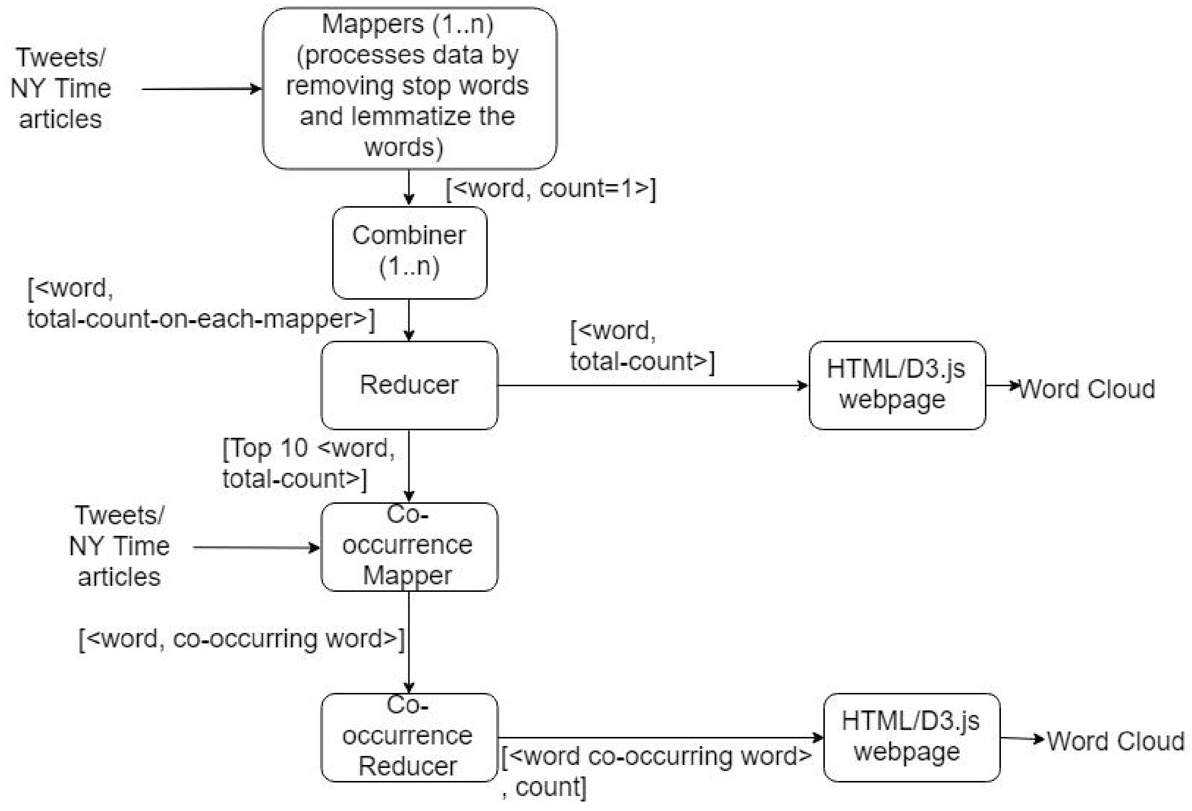


fig. (b) Mapper-Reducer and data visualization flow for word count and co-occurrence

## IMPLEMENTATION DETAILS

**Data Aggregation:**

1. Twitter Data:
    - 1.1. Used tweepy package in Python and for the search string “gun control”, stored the tweets’ full_text attribute in extended mode to get the entire text (240 characters).
    - 1.2. Also, if the tweet is a retweet, get the retweeted.full_text
    - 1.3. Store these tweets in a csv file, which will be further converted in a text file as MR input
    - 1.4. For small dataset, collected 1000 tweets and for whole dataset, gathered 50000 tweets
2. NY Times articles:
    - 2.1. Used articleAPI from nytimesarticle package in Python
    - 2.2. For the search string “gun control”, used api.search to get articles’ web_url from first two pages (i.e. 20 articles) for each day. Such retrieval made sure to get only relevant articles.
    - 2.3. Stored all the web_urls in a csv file. (around 800+ articles)
    - 2.4. Then, using BeautifulSoup package, scraped the paragraph tags discarding irrelevant p tags for all the web URLs saved in step c
    - 2.5. For all the 800+ articles, stored the article contents each in separate text file
    - 2.6. For small dataset, used 80 articles and for large dataset, a total of 824 articles was collected for big data analysis.

**Hadoop MapReduce:**

1. Word count MapReduce job:
    - 1.1. Both Map and Reduce jobs are implemented in python. Hadoop streaming api was used to execute the Map and Reduce jobs.
    - 1.2. Both Map and Reduce jobs get input from standard input i.e. stdin
    - 1.3. Mapper:
        - 1.3.1. File mapper.py. A function named singleWordCountMapper(). Number of mappers: Default, set by hadoop system, can be configured but weren’t configured in the lab.
        - 1.3.2. It reads line from stdin.
        - 1.3.3. Input line gets converted into a list of words.

- 1.3.4. A loop is run on this list.
- 1.3.5. In the loop, the word is passed through regex to remove any leading and trailing symbols. This is useful in case of hashtags, etc.
- 1.3.6. The word is then lemmatized, thus inflected forms of word get processed as single standard word. For example, walk, walks, walking, walked get processed as a single word walk.
- 1.3.7. Next step is to check if the word is in the corpus of stop words or not. For this purpose, stop words corpus of library spaCy is used, it contains around 305 common stop words. Hyperlinks are also ignored.
- 1.3.8. Words which are not stop words corpus or which are not. hyperlinks, are in the dictionary which acts as a combiner.
- 1.3.9. Words are stored in combiner with word as key and its count as value.
- 1.3.10. If the word appears for the first time, then it is added in combiner dictionary with count 1, else the count is incremented.
- 1.3.11. After all the input to the Map is processed, all the key value pairs in the combiner are emitted.

1.4. Reducer:

- 1.4.1. File reducer.py. Function named singleWordCountReducer(). Number of reducers: 1. This is entirely controlled by hadoop system.
- 1.4.2. Reducer reads a line from stdin.
- 1.4.3. Line is a of the format <word\tcount>. The line is unpacked into variables word and count.
- 1.4.4. Now if the word is new, which it is for the first time, current word is set to value of variable word, and current count is set to value of variable count.
- 1.4.5. If the next word that comes in is same as previous word, then the count is updated.
- 1.4.6. This goes on till a new word is encountered, at which point, the aggregated count for the previous word is emitted. And current word and current count variables are reset.
- 1.4.7. The process repeats till there is input in stdin for the Reduce worker.
- 1.4.8. The word, counts are emitted in the format {text:<word>, size:<count>}, so that output file can be easily used for visualization in d3.js.

2. Top 10 words were found out after sorting the output using command line utilities sed and sort.
3. Co-occurrence of top 10 words:
    - 3.1. Both Map and Reduce jobs are implemented in python. Hadoop streaming api was used to execute the Map and Reduce jobs.
    - 3.2. Both Map and Reduce jobs get input from standard input i.e. stdin
    - 3.3. Mapper:
        - 3.3.1. We implemented mapper as a class.
        - 3.3.2. File: cooccurrenceMapper.py.

          Class name:

          Mapper.

          Class Methods:

          cooccurrenceMapper(cooccurrenceList)

          Helper methods:

          formatInputWords(line)
          removeLeadingAndTrailingSymbolsFromWord(word)
          updateStopWords()

          Command line arguments:

          Filename of the file containing top 10 words. The file can contain as many words as desired.

          Number of mappers: Default, set by hadoop system, can be configured but weren't configured in the lab.
        - 3.3.3. The filepath of the file containing top 10 words is passed as an command line argument while invoking the file.
        - 3.3.4. The program then read the top words into a list. This list is passed to the cooccurrenceMapper method of the Mapper class.
        - 3.3.5. After reading the top 10 words into a list, object of Mapper class is created, and cooccurrenceMapper method is called with the top 10 word list as an argument.
        - 3.3.6. In the cooccurrenceMapper, each line is read repeatedly.
        - 3.3.7. For each line, list of words in the line is created, words are formatted with the same process used in simple word count method using the helper functions.
        - 3.3.8. Now loops on the top 10 words list are run to determing whether pairs from the top 10 words list appear in the input line(tweet or article paragraph) or not.
        - 3.3.9. If the pair appears in the same line, <top 10 word1, top 10

cooccurrring word2> is emitted.

3.4. Reducer:

3.4.1. We implemented reducer as a class.

3.4.2. File: cooccurrenceReducer.py
Class name:
Reducer
Class methods:
reduce()
updateCombiner(key, count=1)
emitWords(outputStringFormat)
Number of reducers: 1. This is entirely controlled by hadoop system.

3.4.3. When reducer code is invoked by the reduce job, object of the Reducer class is created and reduce method is called.

3.4.4. Reduce method reads the lines in stdin in a loop.

3.4.5. For each line in the input, format of the line is <top 10 word1, top 10 cooccurring word2>, the line is split in 2 variables word 1 and word 2.

3.4.6. A key = word1 + ‘ ‘ + word2 is generated and updateCombiner(key) function is called.

3.4.7. All the input <word1 + ‘ ‘ + word2>, count are stored in a dictionary called combiner.

3.4.8. updateCombiner(key, count=1) method is used to update the combiner dictionary.

3.4.9. If the key is new then it added in the combiner with count value as 1, else the count value is incremented.

3.4.10. Once all the lines are processed, the word, count pairs are emitted by calling the emitWords(outputStringFormat) method.

3.4.11. The output is formatted by the format specified by the variable outputStringFormat so that the output file can be easily used with d3.js for visualization.

**VISUALIZATION**:

1. Used HTML, D3.js, Bootstrap CSS to create a webpage for visualization of word clouds
2. Stored the output from the reducers in data folder as .js format and assigned the json array to a variable
3. Now in lab2.html, imported all the data scripts and d3.js, bootstrap libraries
4. You can click on different options on navigation bar to check out the corresponding word clouds.
5. Function load_data(words, selector) loads the d3 word cloud in the specified HTML tag for given data

## HOW TO RUN THE CODE

**Data Aggregation:**

Run the code in NairPart2.ipynb/GalaPart2.ipynb file for extracting the data from Twitter and NY Times

**Hadoop MapReduce job:**

Python environment needs to be setup before executing any map reduce job. All the environment requirements are specified in the conda_env.yml file. Before running the following command conda should be installed in the system.

Open terminal, and change the working directory to the wordcount folder and run:

```
> conda env create -f ./conda_env.yml
```

Activate environment:

```
> source activate py27
```

Download the nltk wordnet corpus:

```
> python
>>> import nltk
>>> nltk.download(‘wordnet’)
```

Start hadoop:

```
> $HADOOP_HOME/sbin/start-hadoop.sh
```

1. WordCount on twitter and NYtimes data:
   1.1. We need to copy the files to the hadoop file system. Use the following command to do so.

```
> hdfs dfs -put <path_to_input_files(no backslash)>
<name_of_directory_on_hdfs>
```

1.2. Now run MapReduce jobs. We need to specify our mapper and reducer program files and also load them to all the map and reduce workers.

```
> hadoop jar
<path_to_hadoop_streaming_api>/hadoop-streaming-*.jar
-file mapper.py -file reducer.py -mapper mapper.py
-reducer reducer.py -input <name_of_directory_on_hdfs>
-output <name_of_output_directory_on_hdfs>
```

Note that the output directory should not be present on the hdfs already, if so you need to remove it by the following command.

```
> hdfs dfs -rmr <name_of_directory_on_hdfs>
```

1.3. Copy the  output to local directory from hdfs.

```
> hdfs dfs -get <name_of_output_directory_on_hdfs>
<local_directory>
```

2. Cooccurrence MapReduce job:

2.1. We need to copy the files to the hadoop file system. Use the following command to do so.

```
> hdfs dfs -put <path_to_input_files(no backslash)>
<name_of_directory_on_hdfs>
```

2.2. Make sure top 10(n) words are stored in a file in the directory.

2.3. Now run MapReduce jobs. We need to specify our mapper and reducer program files and also load them to all the map and reduce workers. We also need to load the file, txt file,  containing words to all map reduce workers.

```
> Hadoop jar
<path_to_hadoop_streaming_api>/hadoop-streaming-*.jar
-file cooccurrenceMapper.py -file cooccurrenceReducer.py
-file ./<file_name_containing_top10_words.txt> -mapper
'cooccurrenceMapper.py
./<<file_name_containing_top10_words.txt>' -reducer
cooccurrenceReducer.py -input <name_of_directory_on_hdfs>
```

```
        -output <name_of_output_directory_on_hdfs>
```

2.4. Copy the output to local directory from hdfs.

```
> hdfs dfs -get <name_of_output_directory_on_hdfs>
<local_directory>
```

**Visualization using D3.js**

1. In Word Count Visualization folder, load the lab2.html file in browser (Chrome preferred)

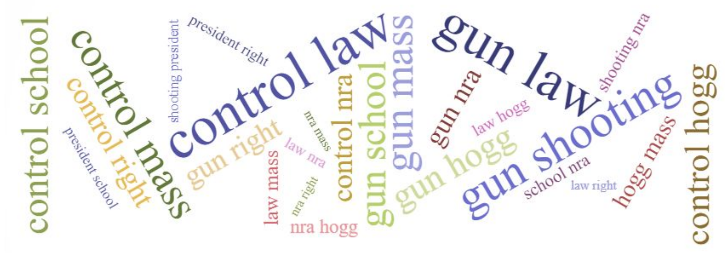
## RESULTS:

After running word count program and sorting the results we found out the following words to be in the top 10ish range. We call them 10ish because couple of words are from top 20 range, but are relevant to the context of gun control for example 'hogg', who is a survivor or shooting incidents, David hogg, another example is nra, which stands for national rifle association. Selecting words which are relevant to the context can provide us better insight.

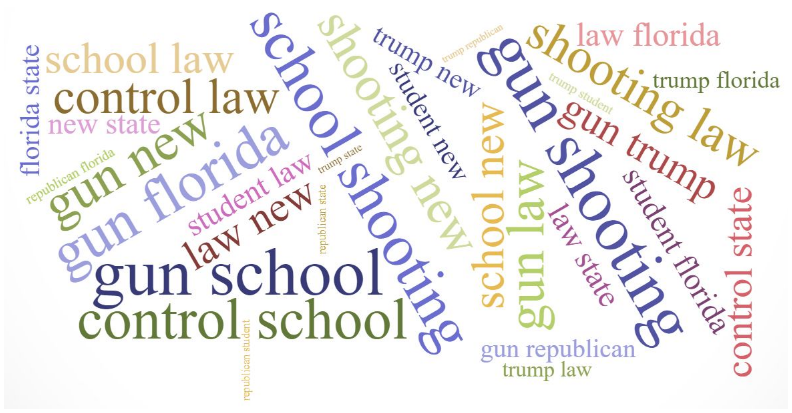
| | **Word** | **Twitter small** | **Twitter full** | **Word** | **NYTimes small** | **NYTimes full** |
|---|---|---|---|---|---|---|
| 1 | gun | 1021 | 53499 | gun | 982 | 3645 |
| 2 | control | 906 | 44700 | control | 263 | 935 |
| 3 | law | 156 | 5658 | school | 404 | 2360 |
| 4 | shooting | 105 | 5738 | trump | 120 | 1828 |
| 5 | president | 81 | 4755 | republican | 105 | 674 |
| 6 | hilary | 66 | 4029 | shooting | 209 | 1360 |
| 7 | school | 75 | 5669 | student | 269 | 1382 |
| 8 | nra | 56 | 5348 | law | 191 | 1059 |
| 9 | hogg | 76 | 3831 | florida | 144 | 628 |
| 10 | mass | 69 | 4838 | president | 144 | 1547 |
| 11 | right | 79 | 4865 | new | 171 | 2220 |
| 12 | | | | state | 177 | 1430 |

Co-occurrence MapReduce job were run on the above list of words, results visualized in d3, shown below:

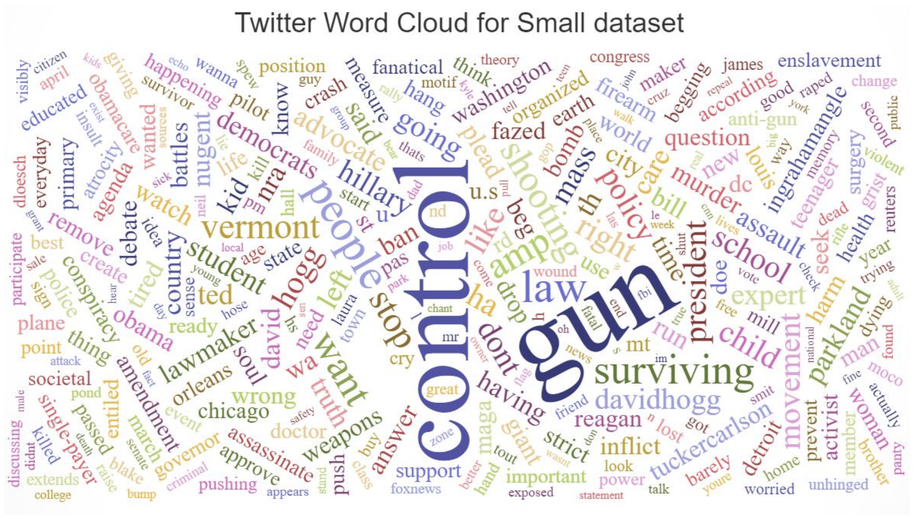
1. Word cloud for co-occurrence in Tweets.



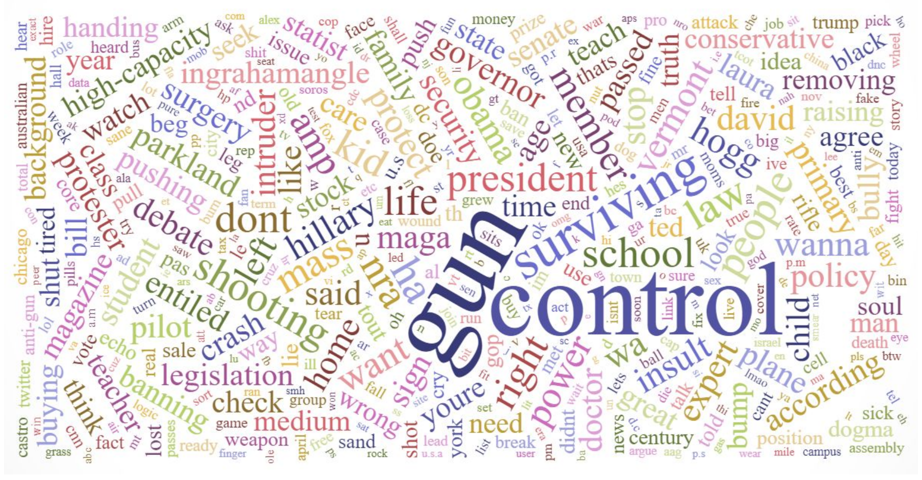
2. Word cloud for co-occurrence in NYTimes.



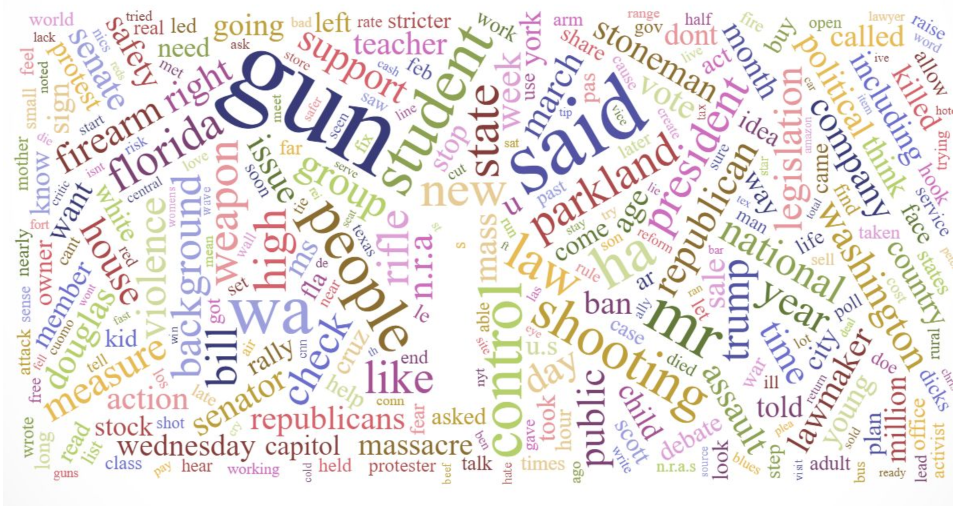
Remaining word clouds for small and full dataset:



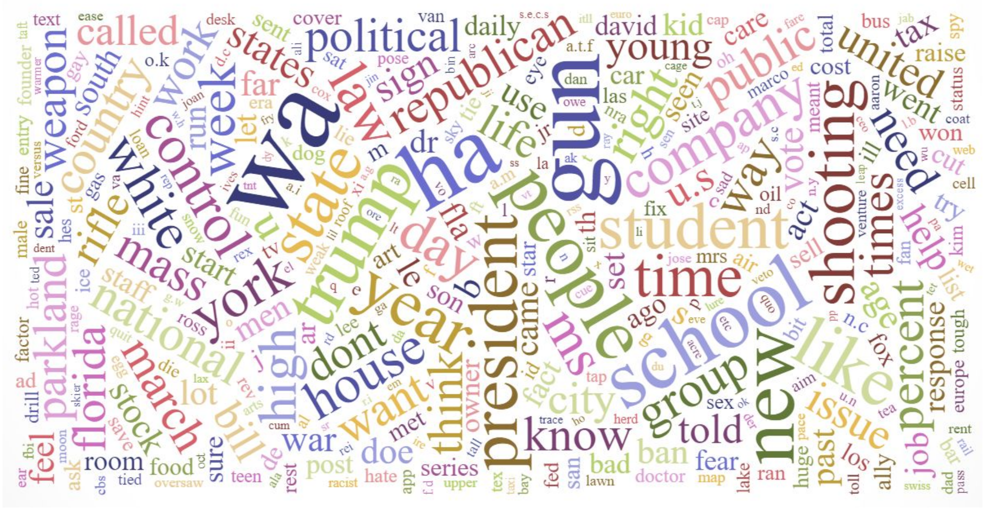
Twitter Word Cloud for Whole dataset



NY Times Word Cloud for Small dataset



NY Times Word Cloud for Whole dataset



## DIRECTORY STRUCTURE

Below are the files present in the tar folder:

- NairPart1.ipynb
- NairPart2.ipynb
- data/
    - part1/
        - GOOG.csv
        - lab2_part1_1.csv
    - part2/
        - nytimes_data/
        - guncontrol_nytimes.csv
        - Guncontrol_tweets.csv
- Word Count MapReduce/
    - nytimes_data/
        - nytimes_small_data/
        - nytimes_full_data/
    - tweets_data/
        - tweets_small_data/
        - tweets_full_data/
    - output/
        - tweets_small_data_output/
        - tweets_full_data_output/
        - nytimes_small_data_output/
        - nytimes_full_data_output/
        - tweets_small_data_cooccurrence_output/
        - nytimes_small_data_cooccurrence_output/
    - mapper.py
    - reducer.py
    - cooccurrenceMapper.py
    - cooccurrenceReducer.py
    - top10wordsTweets.txt
    - top10wordsNytimes.txt
    - Conda_env.yml

- Word Count Visualization/
    - data/
        - JS input files
    - lib/
        - Bootstrap and D3 libraries
    - results/
    - d3.wordcloud.js
    - lab2.html
    - style.css

## SOFTWARE/HARDWARE USED

- Sublime Text 3.
- Anaconda Python 3.
- Hadoop on XUbuntu VM.
- Python environment manager: conda.
- Python 2.7
- Python libraries: tweepy, nytimesarticles api, nltk, spacy
- D3.js, Bootstrap.js.

## REFERENCES

1. https://github.com/wvengen/d3-wordcloud
2. http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/