

# CS5604: Information Storage and Retrieval

## Term Project - Team Reducing-Noise

May 13, 2015

Virginia Tech

Blacksburg, VA

Team Members:

Prashant Chandrasekar

Xiangwen Wang

## Table of Contents

1	Executive Summary .....	5
2	Acknowledgement .....	7
3	VTechWorks Submission Inventory.....	8
4	Chapter/Section Summary .....	9
5	Project Requirements .....	10
6	Literature Review .....	11
7	System Design .....	13
8	Document Properties .....	15
8.1	Tweets .....	16
8.2	Web pages.....	16
9	Implementation Strategy .....	17
10	Tools and Frameworks .....	17
11	Implementation Methodology .....	18
11.1	Tweet Cleanup .....	18
11.2	Webpage Cleanup .....	19
11.3	Organize output in AVRO .....	20
12	Implementation.....	20
12.1	Cleaning Tweets .....	20
12.2	Cleaning Webpages.....	22
13	Implementation Output Statistics.....	25
13.1	Tweet Cleanup .....	25
13.2	Webpage cleanup.....	26
14	Evaluation Techniques .....	28
14.1	Manual evaluation .....	28
14.2	Feedback from teams.....	28
15	Project Timeline .....	29
16	Conclusion & Future Work .....	30
17	User Manual .....	31
17.1	Cleaning tweets for small collections on local machine.....	31

17.2	Cleaning tweets for small collections using Hadoop Streaming.....	31
17.3	Cleaning webpages.....	32
18	Developer Manual.....	34
18.1	Webpage crawling with Python scripts.....	34
18.2	Loading Web page Collection to HDFS.....	35
18.3	Solr Installation and data-importing .....	36
18.4	Hadoop and Mahout Installation .....	38
18.5	Setting up a 6-node Hadoop cluster.....	40
19	References.....	43
	Appendix A: AVRO Schema for Tweets and Web pages .....	45
	AVRO schema for tweet collections .....	45
	AVRO schema for web pages .....	45
	Appendix B: Source code of cleaning script .....	47
	Python script for tweet cleanup.....	47
	Python script for web page cleanup.....	52
	Python code for cleaning up tweets with MapReduce (Raw version) .....	56
	Appendix C: Code provided by TAs .....	59
	Crawl script for Nutch .....	59
	URL extraction from tweets .....	59

## List of Tables

Table 1: Steps and description for cleaning tweets .....	18
Table 2: Steps and description for cleaning web pages .....	19
Table 3: Tweet cleaning statistics .....	25
Table 4: Web page cleaning statistics .....	26
Table 5: Language statistics for web page collections .....	26
Table 6: Various file type for each web page collection .....	27

# List of Figures

- Figure 1: Framework of noise reduction within whole system context..... 13
- Figure 2: Example structure/format of a tweet ..... 16
- Figure 3: Example structure/format of a web page ..... 16
- Figure 4: Example of a cleaned tweet ..... 19
- Figure 5: Overview of tweet cleanup implementation ..... 21
- Figure 6: Overview of web page cleanup implementation ..... 22
- Figure 7: The size of cleaned tweet collections on HDFS ..... 31
- Figure 8: The size of cleaned web pages collection on HDFS..... 33
- Figure 9: Total amount of URLs in small shooting collection ..... 34
- Figure 10: Output from crawling web pages..... 35
- Figure 11: Result after running the Python script ..... 35
- Figure 12: Screenshot displaying “Shooting” collection..... 37
- Figure 13: Import the Hadoop virtual machine into Virtual Box..... 39
- Figure 14: Hadoop is running on the virtual machine ..... 40
- Figure 15: Hadoop installation ..... 42

# 1 Executive Summary

The corpora for which we are building an information retrieval system consists of tweets and web pages (extracted from URL links that might be included in the tweets) that have been selected based on rudimentary string matching provided by the Twitter API. As a result, the corpora are inherently noisy and contain a lot of irrelevant information. This includes documents that are non-English, off topic articles and other information within them such as: stop-words, whitespace characters, non-alphanumeric characters, icons, broken links, HTML/XML tags, scripting codes, CSS style sheets, etc.

In our attempt to build an efficient information retrieval system for events, through Solr, we are devising a matching system for the corpora by adding various facets and other properties to serve as dimensions for each document. These dimensions function as additional criteria that will enhance the matching and thereby the retrieval mechanism of Solr. They are metadata from classification, clustering, named-entities, topic modeling and social graph scores implemented by other teams in the class. It is of utmost importance that each of these initiatives is precise to ensure the enhancement of the matching and retrieval system. The quality of their work is dependent directly or indirectly on the quality of data that is provided to them. Noisy data will skew the results and each team would need to perform additional tasks to get rid of it prior to executing their core functionalities. It is our role and responsibility to remove irrelevant content or “noisy data” from the corpora.

For both tweets and web pages, we cleaned entries that were written in English and discarded the rest. For tweets, we first extracted user handle information, URLs, and hashtags. We cleaned up the tweet text by removing non-ASCII character sequences and standardized the text using case folding, stemming and stop word removal.

For the scope of this project, we considered cleaning only HTML formatted web pages and entries written in plain text file format. All other entries (or documents) such as videos, images, etc. were discarded. For the “valid” entries, we extracted the URLs within the web pages to enumerate the outgoing links. Using the Python package readability [19], we were able to clean advertisement, header and footer content. We were able to organize the remaining content and extract the article text using another Python package BeautifulSoup4 [5]. We completed the cleanup by standardizing the text by removing non-ASCII characters, stemming, stop word removal and case folding.

As a result, 14 tweet collections and 9 web pages collections were cleaned and indexed into Solr for retrieval. (The detailed list of the collection name can be found in Section 17 of the report.)

## 2 Acknowledgement

First, we want to thank the Integrated Digital Event Archiving and Library (IDEAL) [21] team for providing us with the wonderful opportunity to extend their current initiative, thereby providing us with various resources that serve as a platform for our project. This project is sponsored by NSF grant IIS - 1319578. Also we would like to thank the Digital Libraries Research Laboratory (DLRL) for sharing the cluster where we executed our analysis. We also want to convey a special thank you to Dr. Edward A. Fox, and the GTA/GRAs (Sunshin & Mohamed) for helping us throughout the project. Last but not the least, we want to thank all the other teams of CS5604 for their precious feedback and especially the Hadoop & Solr teams for their discussions on the AVRO schemas.



### 3 VTechWorks Submission Inventory

All our work for the semester will be uploaded to VTechWorks at <https://vtechworks.lib.vt.edu/handle/10919/19081>.

Please find below a brief description for each file that will be uploaded as a part of our submission:

1. ReportRN.pdf
  - a. The PDF format of the term report that describes, in detail, our work for the project.
2. ReportRN.docx
  - a. An editable Word format of the term report, ReportRN.pdf
3. PresentationRN.pdf
  - a. The PDF format of the presentation (slides) that provide an overview of our work for the project.
4. PresentationRN.pptx
  - a. An editable PowerPoint format of the presentation, PresentationRN.pdf
5. Code.zip
  - a. A compressed folder that contains the source code for our tweet and web page cleanup implementation.
  - b. Folder contains:
    - i. profanity\_en and profanity\_en\_2: Reference list of curse words or swear words.
    - ii. tweet.avsc: AVRO schema for tweet collections.
    - iii. webpage.avsc: AVRO schema for web page collections.
    - iv. tweet\_cleanup.py: Python script to clean tweets.
    - v. webpageclean.py: Python script to clean web pages.

## 4 Chapter/Section Summary

Chapter 1 introduces the goal of our project.

Chapter 5 discusses our project goal and outlines specific tasks within that goal.

Chapter 7 discusses the overall system architecture.

Chapter 8 provides an insight to the various structure and properties of the documents of our collection.

Chapter 9 and 10 provide details on the approach to our implementation and the tools we will be using to help us during the process.

Chapter 11 describes the step-by-step rules that we followed to clean the tweet collections and the web page collections.

Chapter 12 provides further detail on the process that we used for cleaning via a data flow diagram and an example output for the tweet cleanup implementation.

Chapter 13 describes our work in cleaning all of the collections. The chapter also includes statistics such as the various languages found in the collections along with the various file types that we encountered while cleaning the web page collection. These details are broken down for each collection.

Chapter 14 talks about the different ways in which we evaluated our work.

Chapter 15 provides a timeline breakdown of our tasks for the semester.

Chapter 17 and 18 provide step-by-step instructions to developers and researchers-alike who are interested in using our component and possibly extend its functionality.

Chapter 19 provides an exhaustive list of the references that we consulted for our work.

Appendix A, B and C are supplementary notes that provide the HBase schema, our code and the code provided to us by our TA.

## 5 Project Requirements

Our goal for the project, at a high level, can be described as the following:

1. Identify and remove the “noise” in the data
2. Process and standardize the “sound” in the data
3. Extract and organize the data into a usable format

As the Noise Reduction team, we are to clean up the collection of documents (tweets and web pages) by firstly identifying and removing character sequences that are irrelevant to various components of the IR system. After that, we standardize the remaining text using popular Natural Language Processing techniques to convert the text to a common structure/format. We then extract any information stored within the text that are valuable, such as twitter handles, URLs that are out links in web pages, etc. and store all of this information in a schema/format that aids the other teams that are responsible for building the remaining components of the IR system.

## 6 Literature Review

Before the data-cleaning procedures, several preprocessing steps are necessary. Chapter 2 of [7] introduces the method of word segmentation, true casing, and detecting the coding and language of a document. Also, Section 2.2.2 provides the basic ideas for removing the stop words from a document, which are the extremely common words with little value, by using a predefined stop words list.

One of our goals is to reduce the original HTML or XML documents to smaller text documents. Chapter 10 would be very useful since it provides the concepts and techniques about how to retrieve information from structured documents, such as XML files, HTML files, and other markup language documents.

There are a lot of existing resources in Python for text processing. Thus, we also want to explore how to reduce noise with Python. A very useful and well developed tool in Python, for language processing, is the open source library called *Natural Language Toolkit* (NLTK). There are two references for this tool. The main reference will be “Natural Language Processing with Python” [1], which systematically introduces how NLTK works and how it can be used to clean up documents. We can apply most of the text processing procedures based on this book. For example, Chapter 3 of the book presents the methods for raw text processing, and Chapter 6 introduces how to classify text, etc. In addition, the NLTK official website [2] provides an exhaustive introduction of the NLTK toolkit where we can find details for specific functions that we might use.

For the webpages cleaning-up, the first step is to extract text from the source files. *Beautiful Soup* is an open source Python library to extract text from HTML and XML files. *Beautiful Soup Document* [5] will be our main reference for web page text extraction. We can also find details for each of the functions in the beautifulsoup4 Python package along with some useful examples.

When we successfully extract the text from the HTML, we will find there’s still much visible text in the menus, headers and footers, which we might want to filter out as well. We can approach this problem under this concept: using information of text vs. HTML code to work out if a line of text (inside or outside of an HTML tag) is worth keeping or not. We can be motivated by the methods employed via the online tutorial “The Easy Way To Extract Useful Text From Arbitrary HTML” [8], which provides some ideas on how to fulfill this task. The analysis is based on the neural network method, by using the package open source C library FANN. Webpage [16] provides a general reference manual for the FANN

library. Also, the readability website [19] provides a powerful tool with the introduction for extract useful content from a HTML/XML file.

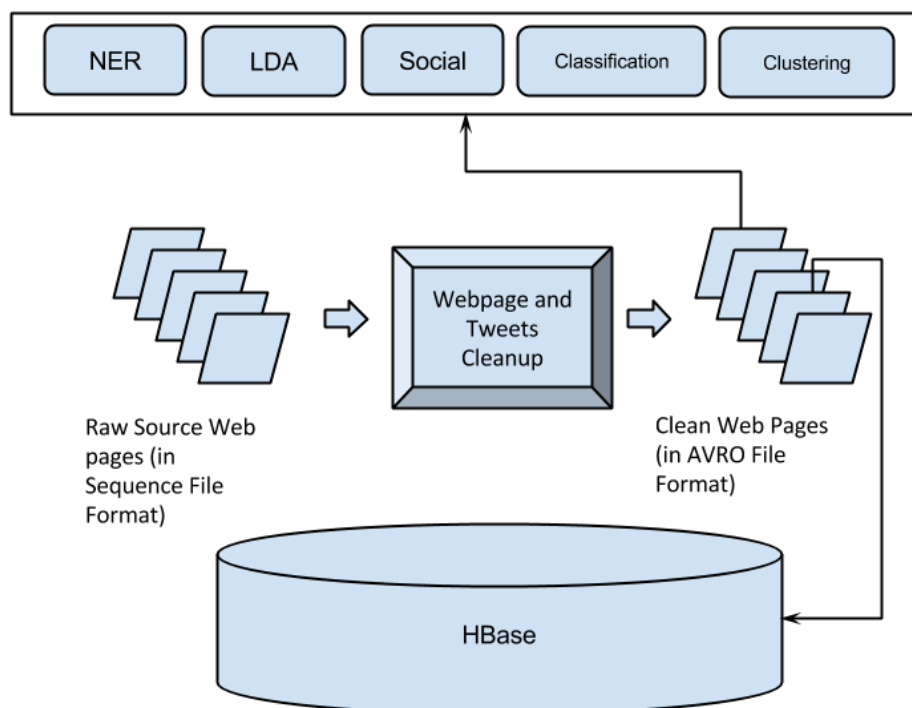
Since we are currently trying to build an English-based information retrieval system, it's important for us to filter out the non-English documents, which leads to a new problem, language detection of a document. The presentation [20] provides us with an idea and its algorithm for language detection based on Naive Bayes classifier. The presenter implements the algorithm with "Noise filter" and "Character normalization" to get a decent detection accuracy of 99.8%.

## 7 System Design

When designing our system, we focused on how the other components of the information retrieval system will consume “clean and relevant” data that is produced. More specifically, our design is focused on building a system that would seamlessly integrate with the frameworks and/or methodologies designed by other teams.

For the large collections, the tweets will be stored in the Hadoop cluster. We execute a Python script: `tweet_shortToLongURL_File.py` (that was provided to us) to extract all the URLs which are fed into Apache Nutch to extract the web pages, which are stored in SequenceFile Format.

We plan to develop a Python script that will run on the Hadoop Cluster via Hadoop’s Streaming API that will process the input files stored on HDFS and output files in AVRO file format. An AVRO file is essentially a JSON file that also contains the JSON schema (within the file) along with the data. The schema for the JSON is provided to us by the Hadoop team, which is responsible for uploading the AVRO files into HBase. Once our system is in place and fully functional, other teams could retrieve the data from the AVRO files or from the tables in HBase.



*Figure 1: Framework of noise reduction within whole system context*

The HBase schema for storing the cleaned tweets and webpages can be found below. The column family “original” contains information that we will be storing after cleaning the collection and as a result, providing to the other teams. Each row contains details for each individual document (tweet and/or web pages). The column family “analysis” contains information that is provided by the other teams as a result of their analysis or work on the cleaned collection. For further details on the individual columns, please refer to the report submitted by the Hadoop team in VTechWorks.

Tweets:

Column Family	Column Qualifier
original	doc_id
	tweet_id
	text_original
	text_clean
	created_at
	source
	user_screen_name
	user_id
	lang
	retweet_count
	favorite_count
	contributors_id
	coordinates
	urls
	hashtags
	user_mentions_id
	in_reply_to_user_id
	in_reply_to_status_id
	text_clean
	collection
analysis	ner_people
	ner_locations
	ner_dates
	ner_organizations
	cluster_id
	cluster_label
	class
	social_importance
	lda_topics
	lda_vectors

Webpages:

```
[rowkey: collection.uuid ]
```

```
Column Family Column Qualifier
```

```
=====
original          doc_id
                  title
                  domain
                  source
                  collection
                  text_original
                  text_clean
                  author
                  subtitle
                  created_at
                  accessed_at
                  section
                  lang
                  urls
                  coordinates
                  tweet_source
                  content_type
                  text_clean
                  appears_in_tweet_ids
```

```
analysis          ner_people
                  ner_locations
                  ner_dates
                  ner_organizations
                  cluster_id
                  cluster_label
                  class
                  social_importance
                  lda_topics
                  lda_vectors
```

The AVRO file format is a JSON schema with the fields (listed above) as keys with corresponding values. Using the schema, the Hadoop team will be writing a utility package that extracts the values within the AVRO file and inserts them as records into HBase.

## 8 Document Properties

Prior to cleaning the character sequences, it is critical to understand and evaluate the structure of each of the documents (tweets and web pages).



## 8.1 Tweets

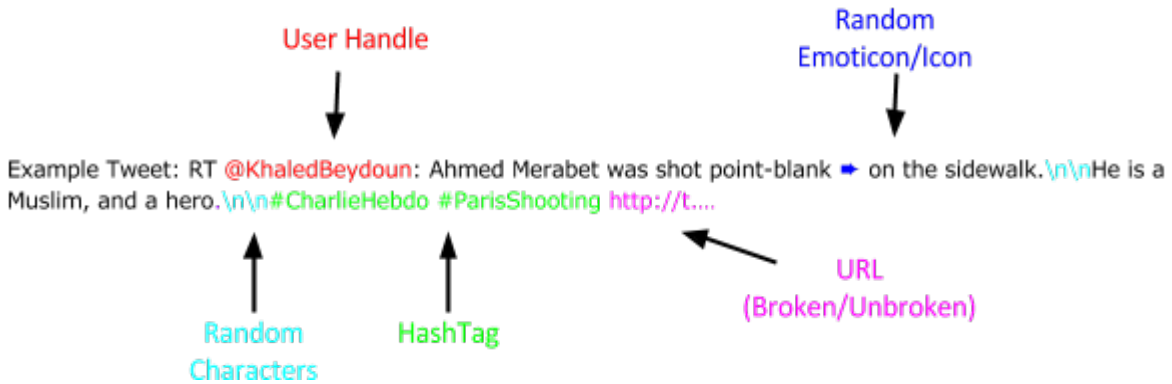


Figure 2: Example structure/format of a tweet

## 8.2 Web pages



Figure 3: Example structure/format of a web page

## 9 Implementation Strategy

There are various languages/frameworks that enable us to implement our design and more specifically, the aforementioned tasks of text processing and document cleanup. Based on the project requirements and our deliverables, we have identified the following constraints/considerations that our implementation strategy has to satisfy:

1. Coding language/framework should have libraries/packages for Natural Language Processing for text cleanup.
2. Coding language/framework should have libraries/packages that are specifically designed to cleanup HTML/XML content.
3. Coding language/framework should be extensible to run on Hadoop cluster with little modification.

Taking into consideration all of the above, we have decided that our implementation will be done in Python. We plan to develop a stand-alone Python script to clean the collections.

## 10 Tools and Frameworks

Following are the tools and frameworks that we plan to leverage as a part of our system/framework to achieve our goal:

1. Python Script To “Unshorten” shortened URLs (Provided by Mohamed)
  - a. There are many shortened URLs in the tweet collections, which would cost additional time for analysis. This script will unshorten and replace them with the original long URLs. The script can be found in Appendix C.
2. Packages in Python
  - a. Natural Language Toolkit (NLTK) is an open source Python library, which provides varieties of modules, datasets, and tutorials for research and development in natural language processing and related areas, such as information retrieval and machine learning. [2]
  - b. BeautifulSoup is an open source package that can be used to extract plain text from HTML/XML source files, which is our first procedure in webpage clean up. [5]
  - c. Langdetect is an open source Python package which can detect the dominant language from given text. [20]

- d. Readability is an open source Python package that extracts the main HTML body text and cleans it up. It helps with removing content from advertisers, banners, etc. [19]
- e. Re is an open source Python package which provides regular expression matching operations. [21]
- f. UUID is an open source Python package which generates UUIDs based on different algorithms (md5, SHA1, etc.). [22]

## 11 Implementation Methodology

We've designed a methodology for each of the intermediate tasks that we have identified for our system's functionality.

### 11.1 Tweet Cleanup

We inspected the collection of tweets that was provided to us. Contents within a tweet are textual in nature. Each tweet has a user handle and may have a hashtag or a URL. In Figure 2 we've highlighted some of the properties of a tweet that are common knowledge. We have additionally identified instances of "noisy" data that require our attention.

Please find below the table describing the intermediate steps for tweet cleanup.

*Table 1: Steps and description for cleaning tweets*

Step Number	Description
1	Discard non-English tweets
2	Remove Emoticons/Other Icons
3	Remove non-alphanumeric characters that aren't part of the User Handle, Hashtag, or URL. (This involves removing punctuation.)
4	Remove "RT" or "rt" that signifies that a tweet is a "retweet"
5	Replace curse words or swear words with the text "profanity"
6	Inspect format of a URL: Remove URL from tweet if the format is invalid (as in the example above)
7	Validate URL: Remove URL from tweet if URL isn't registered or is invalid (if it returns a 404)
8	Standardize text through stemming and stop word removal

9	Generate a universally unique identifier (UUID) for each tweet
10	Output the results in expected format (AVRO)



*Figure 4: Example of a cleaned tweet*

Figure 4 illustrates the output of the tweet cleanup functionality that has removed the invalid URL, non-alphanumeric characters (not part of the User Handle or Hashtag) and icons from the original tweet.

## 11.2 Webpage Cleanup

Cleaning up web pages is far more challenging because web pages are unstructured (as compared to tweets). Web pages do come with the standard HTML tags. However, the HTML source also includes external content present in advertisements, navigational elements, headers, footers, etc. Therefore, it is difficult to identify patterns or properties of text within a web page.

Please find below the table describing the intermediate steps for web page cleanup.

*Table 2: Steps and description for cleaning web pages*

Step Number	Description
1	Discard non-English web pages
2	Remove advertising content, banners, and other such content from the HTML page using Python package: python-readability (include link).
3	Remove text within <script> tags along with style sheets, links to images and other hyperlinks.
4	Cleanup HTML/XML tags, in the web page using package: BeautifulSoup [5]

5	Remove all remaining non-alphanumeric text using regular expressions
6	Standardize remaining text through stemming and stop word removal.
7	Replace curse words or swear words with the text “profanity”
8	Generate a universally unique identifier (UUID) for each web page.

### 11.3 Organize output in AVRO

As mentioned previously, the output from our cleaning module would be consumed by all of the other teams in HDFS. Additionally, the cleaned documents and any metadata that we had extracted, during the cleaning phase, was required to be stored in HBase so that the Solr team could extract the table data into their Solr engine. The Hadoop team was responsible in building the table structure in HBase.

We collaborated with the Solr and the Hadoop team in building the schema for the AVRO file. For details of the schema, please refer to Appendix A.

Since each team is working with two collections that are stored separately (in local filesystem and Hadoop cluster), we will be developing a mechanism for each of the scenarios processing files in Solr and another that will process the collection stored in HDFS.

Through our framework, each team will have access to “clean and relevant” content that would be available and accessible in HDFS (for large collections) and in their local file system (for small collections).

## 12 Implementation

### 12.1 Cleaning Tweets

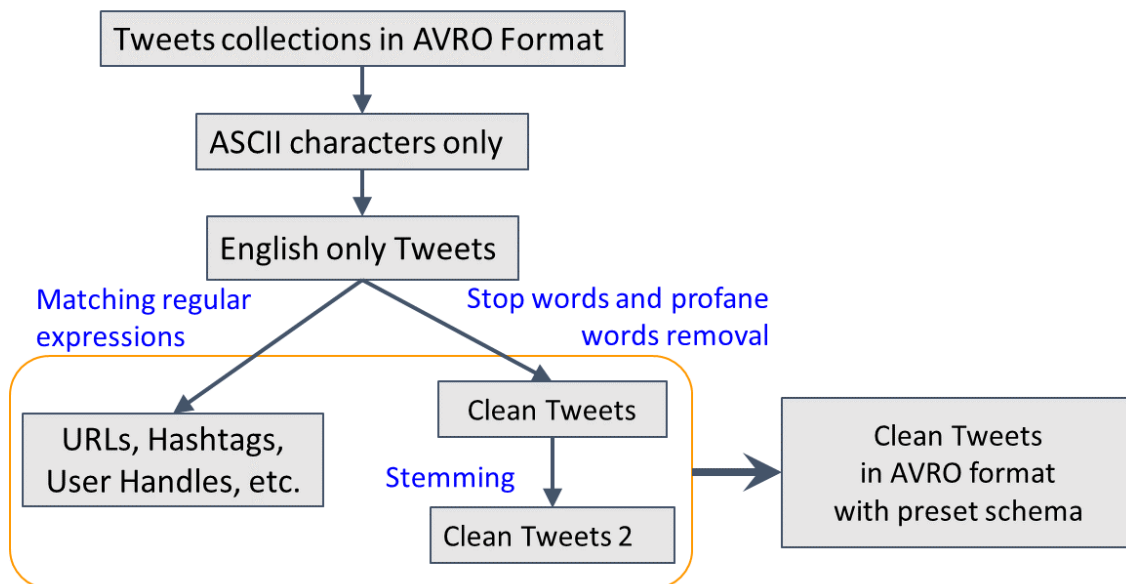


Figure 5: Overview of tweet cleanup implementation

The tweets collections are initially stored in AVRO file format. After dumping it as JSON format, we will only extract “useful” fields such as “text”, “time”, “id”, etc. An example of the tweet input is shown below as in JSON format, where the red “text” is the tweet content which we will clean up.

```

{
  'iso_language_code': 'en',
  'text': u"News: Ebola in Canada?: @CharlieHebdo #CharlieHebdo #Shooting Suspected
  patient returns from Nigeria to Ontario with symptoms http://t.co/KoOD8yweJd",
  'time': 1407706831,
  'from_user': 'toyeenb',
  'from_user_id': '81703236',
  'to_user_id': '',
  'id': '498584703352320001'
  . . .
}

```

We have written a Python script that takes the tweets in AVRO format from HDFS, cleans the tweets and stores the results as AVRO files in HDFS. The AVRO schema can be found in Appendix A.

During the cleanup process, we have extracted user handles, hashtags, usernames, tweet IDs, URLs and timestamps from the original tweets and cleaned up the tweet content. The “hashtags”, “urls”, “user\_mentions\_id”, etc. fields, in the AVRO schema, might be non-value or of multiple values. As of

now, different values are stored within the same string and separated by '|'. Also we generate a UUID for each tweet as the suffix of the “doc\_id”. Examples can be found below as marked red.

```

{
  'lang': 'English',
  'user_id': '81703236',
  'text_clean': 'News Ebola in Canada Suspected patient returns from Nigeria to Ontario
with symptoms',
  'text_clean2': 'new ebol canad suspect paty return niger ontario symptom',
  'created_at': '1407706831',
  'hashtags': 'CharlieHebdo|Shooting',
  'user_mentions_id': 'CharlieHebdo',
  'tweet_id': '498584703352320001',
  'urls': 'http://t.co/KoOD8yweJd',
  'collection': 'charlie_hebdo_S',
  'doc_id': 'charlie_hebdo_S--8515a3c7-1d97-3bfa-a264-93ddb159e58e',
  'user_screen_name': 'toyeenb',
  . . .
}

```

## 12.2 Cleaning Webpages

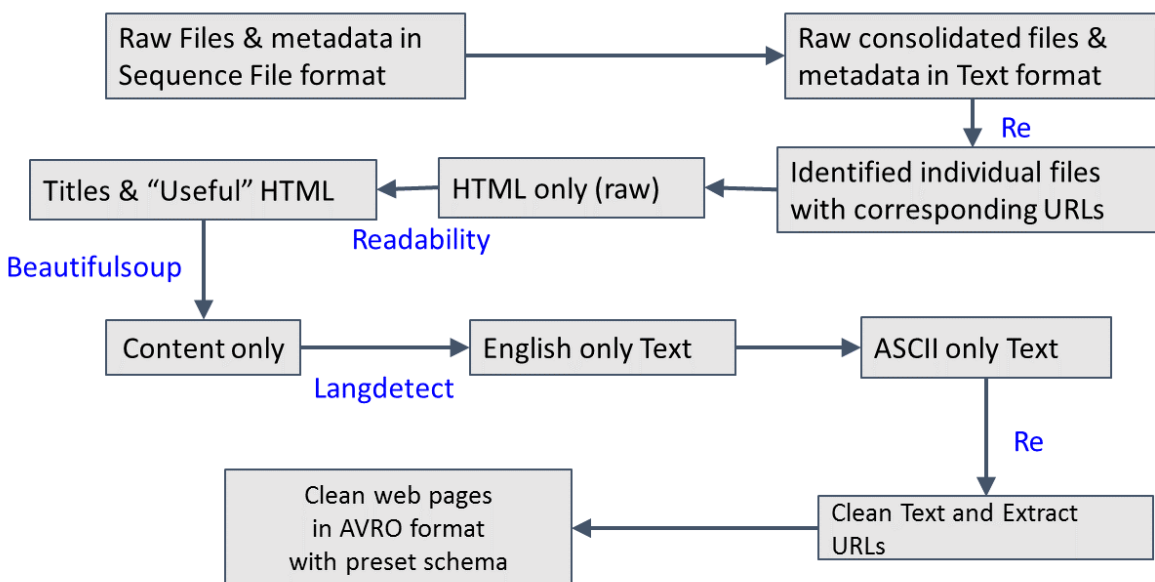


Figure 6: Overview of web page cleanup implementation

The input to our module are web page contents that are output from Apache Nutch. The files are written in SequenceFile format. To clean up the contents of the file, we first had to convert them into text files. We were able to convert the format of the file from SequenceFile format to text format using a script provided by the RA (Mohamed). The details can be found in the Developer Manual section of the report.

The format in which each web page content is presented in the file is:

```
url:http://21stcenturywire.com/2015/02/08/free-speech-british-police-hunt-down-buyers-of-charlie-hebdo/
base:http://21stcenturywire.com/2015/02/08/free-speech-british-police-hunt-down-buyers-of-charlie-hebdo/
contentType: application/xhtml+xml
metadata: X-Pingback=http://21stcenturywire.com/xmlrpc.php Expires=Fri, 20 Mar 2015 23:02:16 GMT _fst_=33 nutch.segment.name=20150320180211 Connection=close X-Powered-By=W3 Total Cache/0.9.4.1 Server=Apache Cache-Control=max-age=3600 Link=<http://wp.me/p3bwni-aFh>; rel=shortlink Date=Fri, 20 Mar 2015 22:02:16 GMT Vary=Accept-Encoding,User-Agent nutch.crawl.score=1.0 Content-Encoding=gzip Content-Type=text/html; charset=UTF-8
Content:
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en-US" xml:lang="en-US">
<head profile="http://gmpg.org/xfn/11">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

The output file contains web page content that is sequentially written into it. For each web page, you have the following information about it (highlighted in red):

1. Original URL
2. URL (same as 1.)
3. Base URL
4. Content Type of web page
5. Metadata of web page
6. Content

For each web page, we extracted the text under the “Content” field using regular expressions. After that; our module employed the Python libraries BeautifulSoup4 & Readability to clean the text and save the cleaned text in AVRO files.

Our biggest challenge was working with non-ASCII character sequences. For such instances, we used the Python library langdetect that parses the text and returns the language with most character sequence occurrences. For example, if a web page contained 70% English text and 30% non-English text, the



library would return “en” as the language of the text. We did not process/clean text for the web pages that didn’t return “en” or “English” as the language of the web page.

Additionally, the social networking team required all the outgoing links for each web page. They needed this information to build their social graph. Using regular expressions, we were able to extract the URLs within a web page document and store them as a list in the AVRO schema.

You can find the details of our code and the results of our execution in the User Manual section of the report.

## 13 Implementation Output Statistics

Please find below the details of our cleanup runs for all of the collections. Collections suffixed with “\_S” are small collections whereas collections suffixed with “\_B” are big collections.

### 13.1 Tweet Cleanup

*Table 3: Tweet cleaning statistics*

Collection Name	Number of Tweets	Number of Non-English Tweets	Percent of Tweets cleaned	Execution Time (seconds)	Size Of Input File (MB)	Size of Output File (MB)
suicide_bomb_attack_S	39258	2083	95% (37175)	32.18	15.14	15.8
Jan_25	911684	415873	55% (495811)	475.18	469.61	210.2
charlie_hebdo_S	520211	346989	34% (173222)	85.46	199.30	70.1
ebola_S	621099	240655	62% (380444)	422.66	226.76	136.3
election_S	931436	101659	90% (829777)	823.77	357.47	323.5
plane_crash_S	273595	7561	98% (266034)	237.96	99.22	100.0
winter_storm_S	493812	7772	99% (486040)	444.90	182.69	192.9
egypt_B	11747983	3797211	68% (7950772)	7271.36	5673.36	3313.2
Malaysia_Airlines_B	1239606	462651	63% (776955)	769.62	472.78	305.3
bomb_B	24676569	3955957	84% (20720612)	16506.19	9358.34	6537.7
diabetes_B	8382585	2845395	67% (5537190)	5452.62	3258.58	2073.2
shooting_B	26381867	3535187	87% (22846680)	19377.55	10288.89	8403.2
storm_B	27286337	4949268	82% (22337069)	18649.99	10658.21	7591.7
tunisia_B	6061926	3322288	46% (2739638)	3265.69	2359.02	1004.4

## 13.2 Webpage cleanup

*Table 4: Web page cleaning statistics*

Collection Name	Number of Web pages	Number of Non-English Web pages	Percent of Web pages cleaned	Execution Time (seconds)	Size of Input File (MB)	Size of Output File (MB)
plane_crash_S	575	170	70.2	30.525	35.09	23.3
ebola_S	1281	321	36.2	68.399	78.38	28.3
diabetes_B	12612	5449	55.1	607.176	572.10	293
election_S	268	74	72.4	17.635	14.42	11.2
tunisia_B	328	162	49.6	15.97	17.07	8.9
charlie_hebdo_S	341	44	87.1	22.927	18.41	16.5
winter_storm_S	1728	536	67.65	89.79	99.00	64
egypt_B	6846	3712	45.45	411.962	359.15	138
communities_B	47238	10539	76.6	2737.437	2300.27	1804

*Table 5: Language statistics for web page collections*

Collection Name	Top 3 Languages	Number of Languages
plane_crash_S	English, French, Tagalog	7
ebola_S	English, Spanish, French	17
diabetes_B	English, Spanish, French	28
tunisia_B	English, French, Indonesian	11
winter_storm_S	English, French, Japanese	17
egypt_B	English, Arabic, French	30
communities_B	English, French, Japanese	45
charlie_hebdo_S	English, French, Arabic	10

election_S	English, French, Hindi	5
------------	------------------------	---

*Table 6: Various file type for each web page collection*

Collection Name	Web page Type
plane_crash_S	application/xml, text/html, application/xhtml+xml
ebola_S	text/html, application/pdf, application/xhtml+xml
diabetes_B	audio/mpeg, application/octet-stream, application/pdf, text/html, application/xhtml+xml, application/x-shockwave-flash, video/mp4, text/plain, text/aspdotnet, text/x-php
tunisia_B	text/html, audio/mpeg, application/pdf, application/xhtml+xml
winter_storm_S	application/atom+xml, text/html, text/plain, application/octet-stream, application/pdf, application/xhtml+xml, application/xml
egypt_B	text/html, application/vnd.android.package-archive, application/xml, video/mp4, application/octet-stream, application/pdf, application/xhtml+xml
communities_B	application/vnd.google-earth.kmz, image/jpeg, audio/mpeg, video/x-ms-wmv, application/rss+xml, video/mp4, text/html, image/gif, application/octet-stream, video/x-m4v, application/pdf, application/xhtml+xml, application/x-rar-compressed, text/x-php, text/aspdotnet
election_S	text/html, application/xhtml+xml
charlie_hebdo_S	text/html, application/xhtml+xml

## 14 Evaluation Techniques

The evaluation of our collection was conducted in two-phases: 1) Manual Evaluation 2) Feedback from other teams.

### 14.1 Manual evaluation

As we were saving the cleaned text into AVRO format, we additionally, saved the plain text version for a subset of the entries, through random selection. We randomly selected 50 documents from each collection and scanned the text to check for any aberration in the output. We conducted the test for tweets and web pages.

It was through this validation step that we learnt that some documents had text with multiple languages. Until then, we were removing all non-ASCII text. We assumed that by doing so, we would get rid of non-English text in the document. However, this wasn't enough as we had web pages written in non-English, say Japanese, which included many instances of numbers and dates within the text. Only these numbers and dates would appear in the cleaned version of the document.

As a refinement, we chose to process the documents written in English. The Python package, langdetect [20], was included in our code to identify the prominent language of the document. We would filter out documents that didn't predominantly contain English text.

### 14.2 Feedback from teams

The cleaned collections were directly consumed by other teams. We asked the teams to let us know if their calculations or results were skewed (or affected) because of occurrences of any text that they considered as "noise". The common concern among all the teams was the occurrence of the term "RT" or "rt" when analyzing tweet content. The character sequence "RT" (or "rt") is a Twitter specific term that stands for "retweet". It is often found in tweets that are "re-tweets" of the original tweet.

Also, Dr. Fox noticed several instances of curse words or swear words in the cleaned text. He suggested that we replace these occurrences with the term "profanity", so as to standardize the text and preserve them for future text analysis, such as sentiment analysis. We used Google's list of curse words or offensive words for reference. [23]

Therefore, we cleaned the collections to remove the occurrence of the character sequence "RT" and "rt" and replaced all curse words and changed them to "profanity".

## 15 Project Timeline

Report 1: Solr installation on local machines. Documented the requirements for our team's work and identified the dependencies associated with other teams.

Report 2: Data imported into Solr. Outlined the requirement of our team's work. Finalized the References.

Report 3: Reorganized the structure of the report. Finalized the design part.

Report 4: Coding for text extraction from webpages.

Report 5: The code for cleanup tweet collections will be released along with the report.

Report 6: The preliminary executable code for cleanup of web pages will be released.

Report 7: Cleaned tweet collections ready.

Reports 8&9: The final executable code for web page cleanup released.

Report 10: Version 2 of the web page cleanup released.

## 16 Conclusion & Future Work

Through our work, we were able to successfully clean 14 English tweet collections and 9 English HTML formatted web page collections. These collections were indexed and loaded into Solr for retrieval.

We were able to employ industry standard packages/utilities to achieve our goals. Our work can be extended to extract more information about each collection, as a whole, as well as individual documents within them.

For example, our cleaning process removed all emoticons, which could be used along with other text processing tools to derive sentiment for tweets. This information would be useful for collections that include “disasters” such a hurricanes or fires as well as for the Egypt uprising collection.

Additionally, our work was restricted to processing entries that were in English. Researchers and developers alike could relax the constraint in our code and could achieve similar goals for the 30 or so languages that we detected in our collection.

Finally, our current scope was defined such that we only focus on cleaning HTML formatted or plain text formatted documents. Important documents that were represented in other formats could’ve been pruned as a result. There are many freely available tools that help convert and/or extract plain text from documents of various formats such as PDF files and Word files. Interested parties can extend our code to covert these documents to plain text format for further processing.

## 17 User Manual

### 17.1 Cleaning tweets for small collections on local machine

The code has been attached. The procedures are listed below:

1. Install Python library AVRO with the command “pip install Avro --user”. Save the AVRO schema file in Appendix to “tweets.arsc”, and save the Python code attached to “tweet\_cleanup.py”, and upload them onto the cluster.
2. Copy the large collections on HDFS to local using the command “hadoop fs -copyToLocal /class/CS5604S15/dataset/XXXXXX/part-m-00000.avro ./XXXXXX/part-m-00000-ori.avro”, where XXXXXX is the collection name.
3. To clean up the tweets, run the Python script with the command “nohup python tweet\_cleanup.py part-m-00000-ori.avro part-m-00000.avro tweet.avsc XXXXXX &”, where argument part-m-00000-ori.avro is the input AVRO file, argument part-m-00000.avro is the output AVRO file, and tweet.avsc is the AVRO schema file.
4. Copy the output AVRO file onto HDFS with the command “hadoop fs -copyFromLocal part-m-00000.avro YYYYYY”, where YYYYYY is the expected path of the output AVRO file on HDFS.

All the small and large tweet collections have been cleaned up, and they are available under the “/user/cs5604s15\_noise/TWEETS\_CLEAN/” folder. One can check the cleaned collections with the command “hadoop fs -du -h /user/cs5604s15\_noise/TWEETS\_CLEAN/”, as shown below.

```
[cs5604s15_noise@node1 ~]$ hadoop fs -du -h /user/cs5604s15_noise/TWEETS_CLEAN
204.3 M 612.8 M /user/cs5604s15_noise/TWEETS_CLEAN/Jan.25_S
258.4 M 775.3 M /user/cs5604s15_noise/TWEETS_CLEAN/Malaysia_Airlines_B
5.4 G 16.3 G /user/cs5604s15_noise/TWEETS_CLEAN/bomb_B
60.8 M 182.5 M /user/cs5604s15_noise/TWEETS_CLEAN/charlie_hebdo_S
1.7 G 5.1 G /user/cs5604s15_noise/TWEETS_CLEAN/diabetes_B
121.0 M 363.0 M /user/cs5604s15_noise/TWEETS_CLEAN/ebola_S
2.9 G 8.8 G /user/cs5604s15_noise/TWEETS_CLEAN/egypt_B
284.9 M 854.8 M /user/cs5604s15_noise/TWEETS_CLEAN/election_S
86.6 M 259.7 M /user/cs5604s15_noise/TWEETS_CLEAN/plane_crash_S
6.9 G 20.7 G /user/cs5604s15_noise/TWEETS_CLEAN/shooting_B
6.3 G 19.0 G /user/cs5604s15_noise/TWEETS_CLEAN/storm_B
13.1 M 39.4 M /user/cs5604s15_noise/TWEETS_CLEAN/suicide_bomb_attack_S
836.7 M 2.5 G /user/cs5604s15_noise/TWEETS_CLEAN/tunisia_B
166.2 M 498.6 M /user/cs5604s15_noise/TWEETS_CLEAN/winter_storm_S
```

*Figure 7: The size of cleaned tweet collections on HDFS*

### 17.2 Cleaning tweets for small collections using Hadoop Streaming



You can clean up the large tweets collection (in CSV format) using the Hadoop steaming and mapper.py (attached in the appendix). The only problem is that the output of this execution is NOT a custom AVRO schema. This was an experiment to extend our Python implementation to run on HDFS. Following are the steps to execute the script:

1. First, you need to download the avro-1.7.7.jar and avro-mapred-1.7.7-hadoop1.jar from <http://www.gtlib.gatech.edu/pub/apache/avro/avro-1.7.7/java/>, and upload them to the same directory with the mapper.py on cluster.
2. For the shell command attached below, replace the "MAPPER\_PATH" with the path of the mapper.py, for example, as the noise reduction team, our path is "/home/cs5604s15\_noise/testmapreduce/mapper.py".
3. Again, replace the "INPUT\_PATH" and "OUTPUT\_PATH" with the paths of your input and output directories. For example, our input path is "cs5604s15\_noise/input/", and our output path is "/user/cs5604s15\_noise/output\_tweet\_test\_20150328".
4. Then paste the command in remote terminal and run it. The output will be stored in AVRO format in the given output directory.

Shell command

```
-----  
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming.jar -D mapred.reduce.tasks=0 -files avro-1.7.7.jar,avro-mapred-1.7.7-hadoop1.jar -libjars avro-1.7.7.jar,avro-mapred-1.7.7-hadoop1.jar -file MAPPER_PATH -mapper MAPPER_PATH -input INPUT_PATH/*.csv -output OUTPUT_PATH -outputformat org.apache.avro.mapred.AvroTextOutputFormat  
-----
```

### 17.3 Cleaning webpages

The following process was following to clean the web pages for each and every collection:

1. The web pages (that were output from Apache Nutch) are in SequenceFile format. We first need to convert them to text file format.

```
hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-2.5.0-mr1-cdh5.3.1.jar -files apache-nutch-1.9.jar -libjars apache-nutch-1.9.jar -D mapred.reduce.tasks=0 -input /user/cs5604s15_cluster/ebola_S_webpages/ebola_S_Crawl/segments/20150413181432/content/part-000000/data -inputformat SequenceFileAsTextInputFormat -output /user/cs5604s15_noise/clustering_webpage_small -mapper org.apache.hadoop.mapred.lib.IdentityMapper
```

The above script takes as input the web pages for the clustering team's collection.

## 2. Run Python script to clean the text files.

```
python webpageclean.py
~/clustering_webpage_data/small_text_data/clustering_webpage_small/part-00000
WEBPAGES_CLEAN/clustering_small_00000_v2 webpage_new.avsc ebola_S
```

The above command is of format: python {Python script file} {input file} {output file} {AVRO schema for webpages} {collection name}

## 3. Load output into HDFS

```
hadoop fs -put /WEBPAGES_CLEAN/clustering_small_00000_v2 /user/cs5604s15_noise/WEBPAGES_CLEAN/
```

All the small and large web page collections have been cleaned up, and they are available under the “/user/cs5604s15\_noise/WEBPAGES\_CLEAN/” folder. One can check the cleaned collections with the command “hadoop fs -ls /user/cs5604s15\_noise/WEBPAGES\_CLEAN”, as shown below.

```
[cs5604s15_noise@node1 ~]$ hadoop fs -ls /user/cs5604s15_noise/WEBPAGES_CLEAN
Found 30 items
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 16686151 2015-05-02 20:30 /user/cs5604s15_noise/WEBPAGES_CLEAN/classification_small_00000_v2
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 8299499 2015-05-02 20:30 /user/cs5604s15_noise/WEBPAGES_CLEAN/classification_small_00001_v2
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 118997565 2015-05-02 22:15 /user/cs5604s15_noise/WEBPAGES_CLEAN/clustering_large_00000_v1
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 186977392 2015-05-02 22:15 /user/cs5604s15_noise/WEBPAGES_CLEAN/clustering_large_00001_v1
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 8454196 2015-04-23 16:42 /user/cs5604s15_noise/WEBPAGES_CLEAN/clustering_small_00000
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 8462670 2015-04-26 18:05 /user/cs5604s15_noise/WEBPAGES_CLEAN/clustering_small_00000_v2
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 19892415 2015-04-23 16:42 /user/cs5604s15_noise/WEBPAGES_CLEAN/clustering_small_00001
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 19917217 2015-04-26 18:05 /user/cs5604s15_noise/WEBPAGES_CLEAN/clustering_small_00001_v2
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 82503774 2015-04-23 16:42 /user/cs5604s15_noise/WEBPAGES_CLEAN/hadoop_small_00000
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 82716142 2015-04-26 18:05 /user/cs5604s15_noise/WEBPAGES_CLEAN/hadoop_small_00000_v2
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 61997309 2015-04-23 16:42 /user/cs5604s15_noise/WEBPAGES_CLEAN/hadoop_small_00001
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 61382017 2015-04-26 18:05 /user/cs5604s15_noise/WEBPAGES_CLEAN/hadoop_small_00001_v2
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 42629815 2015-04-23 16:42 /user/cs5604s15_noise/WEBPAGES_CLEAN/ner_small_00000
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 42569518 2015-04-26 18:05 /user/cs5604s15_noise/WEBPAGES_CLEAN/ner_small_00000_v2
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 22449398 2015-04-23 16:42 /user/cs5604s15_noise/WEBPAGES_CLEAN/ner_small_00001
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 23250237 2015-04-26 18:05 /user/cs5604s15_noise/WEBPAGES_CLEAN/ner_small_00001_v2
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 78575447 2015-05-02 22:15 /user/cs5604s15_noise/WEBPAGES_CLEAN/noise_large_00000_v1
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 299322794 2015-05-02 22:15 /user/cs5604s15_noise/WEBPAGES_CLEAN/noise_large_00001_v1
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 8757982 2015-04-23 16:42 /user/cs5604s15_noise/WEBPAGES_CLEAN/noise_small_00000
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 8771493 2015-04-26 12:26 /user/cs5604s15_noise/WEBPAGES_CLEAN/noise_small_00000_v2
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 8356503 2015-04-23 16:42 /user/cs5604s15_noise/WEBPAGES_CLEAN/noise_small_00001
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 8183090 2015-04-26 18:05 /user/cs5604s15_noise/WEBPAGES_CLEAN/noise_small_00001_v2
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 461710605 2015-04-23 16:42 /user/cs5604s15_noise/WEBPAGES_CLEAN/social_00000
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 462612082 2015-04-26 13:17 /user/cs5604s15_noise/WEBPAGES_CLEAN/social_00000_v2
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 3420783 2015-05-02 22:15 /user/cs5604s15_noise/WEBPAGES_CLEAN/solr_large_00000_v1
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 5405302 2015-05-02 22:15 /user/cs5604s15_noise/WEBPAGES_CLEAN/solr_large_00001_v1
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 7013519 2015-04-23 16:42 /user/cs5604s15_noise/WEBPAGES_CLEAN/solr_small_00000
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 7024837 2015-04-26 18:05 /user/cs5604s15_noise/WEBPAGES_CLEAN/solr_small_00000_v2
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 4274191 2015-04-23 16:42 /user/cs5604s15_noise/WEBPAGES_CLEAN/solr_small_00001
-rw-r--r-- 3 cs5604s15_noise cs5604s15_noise 4477940 2015-04-26 18:05 /user/cs5604s15_noise/WEBPAGES_CLEAN/solr_small_00001_v2
[cs5604s15_noise@node1 ~]$
```

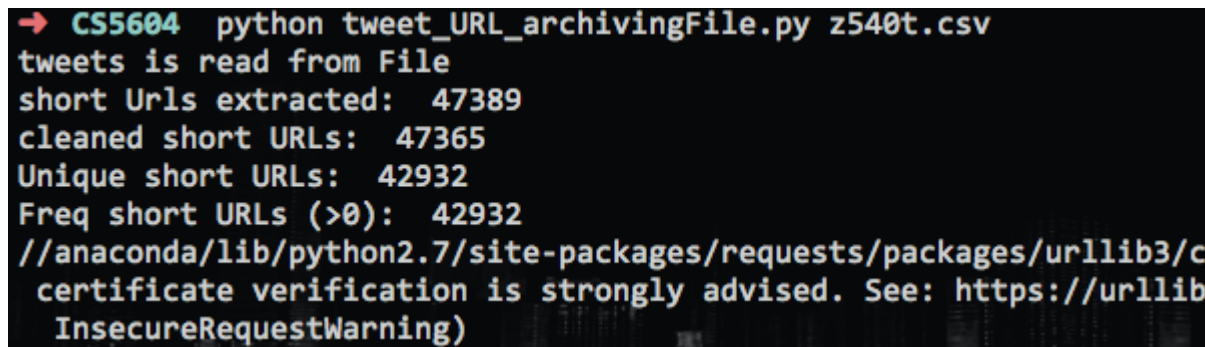
Figure 8: The size of cleaned web pages collection on HDFS

## 18 Developer Manual

### 18.1 Webpage crawling with Python scripts

As the noise reduction team, we have been given a small tweet collection about the event “shooting”. With the Python script provided by the RA, Mohamed Magdy, we are able to crawl each of the webpages that have a URL appearing in the set of tweets. We ran the script on a Macbook with Python 2.7.8 and Anaconda 2.1.0 installed. The procedures are listed below:

1. Download Mohamed’s script (please refer to Appendix C) and put it in the same directory as the small collection.
2. Modify the script, change line 17 from “thresh = 10” to “thresh = 0” to count the total number of unique URLs in the collection. Run the script by using the command “python tweet\_URL\_archivingFile.py z540t.csv”.
3. Terminate the program when it starts to send a connection request. See the figure below



```
→ CS5604 python tweet_URL_archivingFile.py z540t.csv
tweets is read from File
short Urls extracted: 47389
cleaned short URLs: 47365
Unique short URLs: 42932
Freq short URLs (>0): 42932
//anaconda/lib/python2.7/site-packages/requests/packages/urllib3/c
certificate verification is strongly advised. See: https://urllib
InsecureRequestWarning)
```

*Figure 9: Total amount of URLs in small shooting collection*

4. We find that there are in total 42932 unique URLs appearing in the small tweet collection. Due to the limitation on the performance of our computers and network bandwidth, we will only crawl the web pages that have a URL appearing more than 5 times in the set. However, on the cluster, this need not be the case.
5. Modify the script, change line 17 from “thresh = 0” to “thresh = 5”.
6. Rerun the script, using the same command “python tweet\_URL\_archivingFile.py z540t.csv”. Wait until the program finishes running, which takes about 5 minutes, as shown in the figure below:

```

→ CS5604 python tweet_URL_archivingFile.py z540t.csv
tweets is read from File
short Urls extracted: 47389
cleaned short URLs: 47365
Unique short URLs: 42932
Freq short URLs (>5): 196
//anaconda/lib/python2.7/site-packages/requests/packages/urllib3/connectionpool
certificate verification is strongly advised. See: https://urllib3.readthedocs
InsecureRequestWarning)
<class 'requests.exceptions.ReadTimeout'> http://t.co/OCTjv7kcK2
Unique Orig URLs expanded: 85
Bad URLs: 6
Webpages text saved

```

Figure 10: Output from crawling web pages

- In total 85 unique web pages are successfully collected and stored within the same directory of the script.

78.txt	Today 1:11 PM	17 KB	Plain...cument
79.txt	Today 1:11 PM	1 KB	Plain...cument
80.txt	Today 1:11 PM	35 KB	Plain...cument
81.txt	Today 1:11 PM	5 KB	Plain...cument
82.txt	Today 1:11 PM	47 KB	Plain...cument
83.txt	Today 1:11 PM	5 KB	Plain...cument
84.txt	Today 1:11 PM	8 KB	Plain...cument
85.txt	Today 1:11 PM	8 KB	Plain...cument
seedsURLs_z540t.txt	Today 1:11 PM	7 KB	Plain...cument
short_origURLsMapping_z540t.txt	Today 1:11 PM	12 KB	Plain...cument
shortURLs_z540t.txt	Today 1:07 PM	1.1 MB	Plain...cument
tweet_URL_archivingFile.py	Today 1:07 PM	4 KB	Python
z540t.csv	Feb 12, 2015 1:23 PM	8.3 MB	comm...values

Figure 11: Result after running the Python script

## 18.2 Loading Web page Collection to HDFS

Please execute the following instructions in order to load the web page collection into HDFS:

- Download (Shooting/Charlie Hebdo) collection called z540t.csv at <http://nick.dlib.vt.edu/data/CS5604S15/z540t.csv>
- Download Python script to extract URLs called tweet\_shortToLongURL\_File.py at [https://scholar.vt.edu/access/content/group/5508d3d6-c97d-437f-a09d-2cfd43828a9d/Tutorials/tweet\\_shortToLongURL\\_File.py](https://scholar.vt.edu/access/content/group/5508d3d6-c97d-437f-a09d-2cfd43828a9d/Tutorials/tweet_shortToLongURL_File.py)
- Execute Python script:
  - python tweet\_shortToLongURL\_File.py z540t.csv

- b. This script outputs a file with a list of URLs that we can then use as a seed file for Apache Nutch: seedsURLs\_z540t.txt
- 4. Download Apache Nutch (as per instruction provided in the Tutorial):  
<https://scholar.vt.edu/access/content/group/5508d3d6-c97d-437f-a09d-2cfd43828a9d/Tutorials/Nutch%20Tutorial.pdf>)
- 5. Modify crawl script (in directory bin/crawl)
  - a. Comment out section that indexes the web pages into Solr
- 6. Execute Nutch
  - a. `bin/crawl ../urls/ TestCrawl4/ http://preston.dlib.vt.edu:8980/solr/1 1`
    - i. `urls/` is the directory where the seed URL text file is.
    - ii. `TestCrawl4/` is the directory where the output is stored
    - iii. <http://preston.dlib.vt.edu:8980/solr> is the Solr instance that needs to be provided regardless of whether the output is indexed on Solr or not.
    - iv. `1` is the number of times you want to attempt to connect to the URLs
  - b. The web page file can be found in directory: `$nutch-1.9/TestCrawl4/segments/20150320180211/content/part-0000`
  - c. Upload File to Hadoop Cluster
    - i. `scp part-0000 cs5604s15_noise@hadoop.dlib.vt.edu:/home/cs5604s15_noise/`
  - d. Upload File to HDFS
    - i. `hadoop fs -copyFromLocal /home/cs5604s15_noise/* /user/cs5604s15_noise/input`

### 18.3 Solr Installation and data-importing

To get a better understanding of the IDEAL project and information retrieval systems, we want to install and practice with Solr on our local machines. We installed Solr on a PC with Windows 7 and indexed the collection (CSV file) by following procedures based on the *Solr Quickstart Tutorial* [6]:

1. Download Java version 1.7 64-bit installation file (jdk-7u75-windows-x64.exe) from Oracle website [3].
2. Install Java and modify the related environment variables (PATH, JAVA\_HOME and CLASSPATH).
3. Download Solr installation file (solr-4.10.3.zip) from Apache Solr website [4] and unzip it in "D:/Solr".
4. Under the command-line tool (cmd.exe), enter the Solr directory, and run the command "bin/solr.cmd start -e cloud -noprompt" to launch Solr
5. Check if the Solr is running by visiting "<http://localhost:8983/solr/>." with a web-browser.
6. Edit the environment variable CLASSPATH, add ";D:/Solr/dist/solr-core-4.10.3.jar".
7. Copy the collection CSV file, and the web pages just crawled into directory "D:/Solr/data".

8. Index the small collection by running the command “curl 'http://localhost:8983/solr/update?commit=true&separator=%09' -H 'Content-type:application/csv' --data-binary @z4t.csv” in command line tool.
9. Index the web pages by running the command “java -Dauto org.apache.solr.util.SimplePostTool data/\*.txt”.
10. Check if the collection has been indexed by visiting the link “<http://localhost:8983/solr/browse>”.

As seen in the figure below, the collection has been imported.

The screenshot shows the Apache Solr browse interface. At the top, there is a search bar with the text "Find:" and a "Boost by Price" checkbox. Below the search bar, there are several facets: "Field Facets", "Query Facets", "Range Facets", "Pivot Facets", and "Clusters". The "Field Facets" section lists various fields with their counts, such as "cat" (37001), "manu\_exact" (37001), "content\_type" (70), "author\_s" (36917), and "author\_s" (37001). The "Query Facets" section lists "ipod" (1) and "GB" (1). The "Range Facets" and "Pivot Facets" sections are currently empty. The "Clusters" section provides instructions on how to run Solr with clustering enabled. The main search results area displays four items, each with a title, a "More Like This" link, an ID, and fields for Price, Features, and In Stock. The items are: "7thWoman", "Lyndon\_RSA", "ChrystalDimitra", and "BtSPAK".

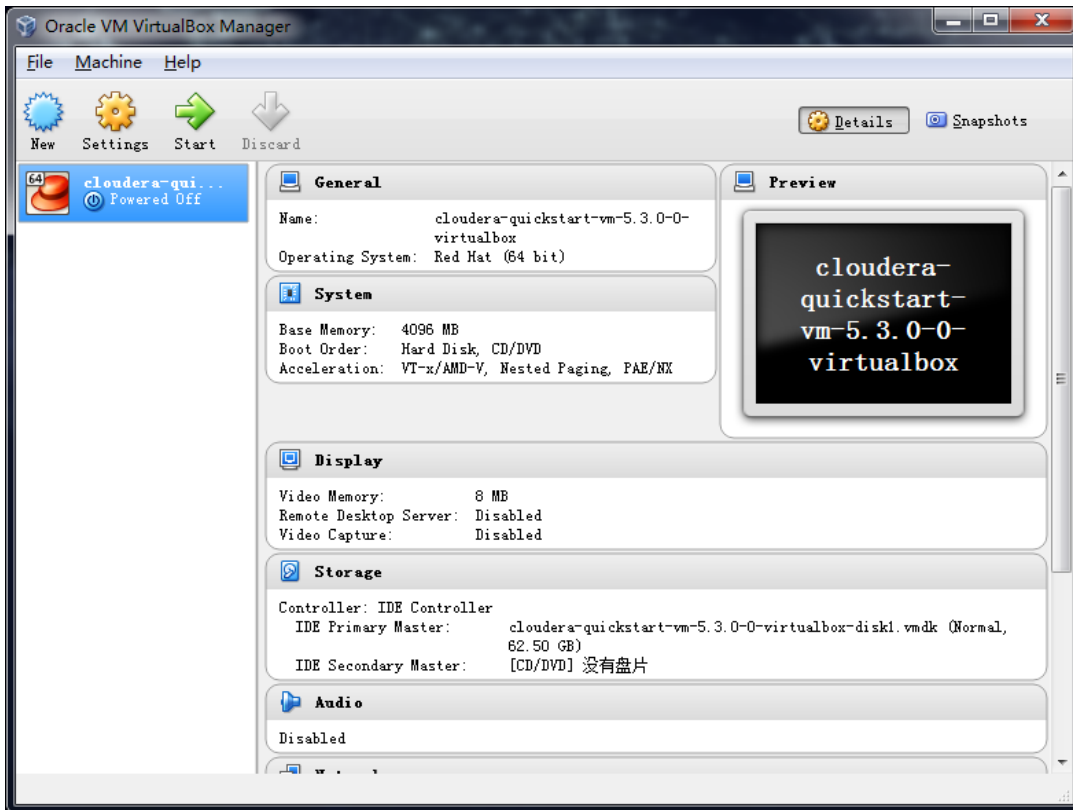
*Figure 12: Screenshot displaying “Shooting” collection*

This small collection contains 36917 tweets about the “CharlieHebdo attack”, so we can practice normalizing text, and classifying text using those tweets. Also, we can try to unshorten each tiny URL in those tweets.

## **18.4 Hadoop and Mahout Installation**

We initially installed Mahout to understand the input format for all of the machine learning implementations that were being used by other teams. We were able to experiment freely with our own installation. Following is a list of the procedures to build a Hadoop+Mahout environment on a virtual machine with a Windows 7 64-bit PC as the host.

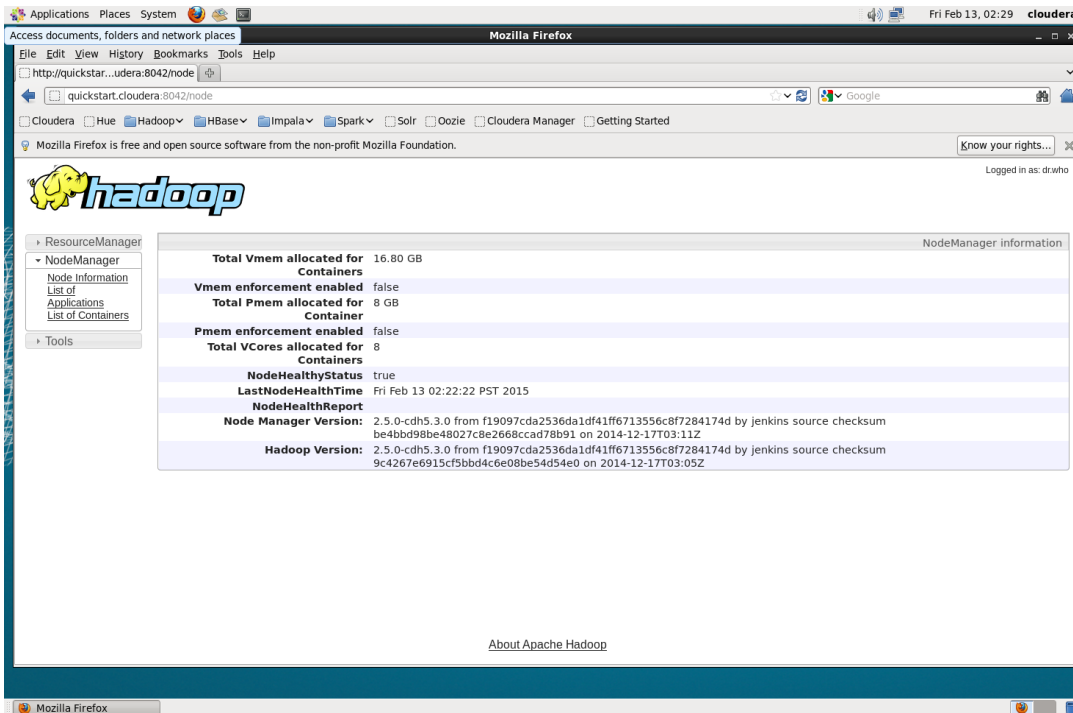
1. Download the virtualization software VirtualBox version 4.3.22 (VirtualBox-4.3.22-98236-Win.exe) from Oracle VM VirtualBox website, <https://www.virtualbox.org/>, and install it.
2. We will use the open source Cloudera Distribution Including Apache Hadoop (CDH) as our Hadoop platform. Download the CDH demo, a VirtualBox image which has CentOS 6.4 and CHD 5.3 already built in, from Cloudera Quick-start webpage, [http://www.cloudera.com/content/cloudera/en/downloads/quickstart\\_vms/cdh-5-3-x.html](http://www.cloudera.com/content/cloudera/en/downloads/quickstart_vms/cdh-5-3-x.html), and unzip it.
3. From the VirtualBox menu, import the OVF file which is just unzipped.



*Figure 13: Import the Hadoop virtual machine into Virtual Box*

4. Turn on the virtual machine, check the Node Manager, and see that the Hadoop is running.





*Figure 14: Hadoop is running on the virtual machine*

5. Then we install Mahout on CentOS, using the command “`sudo yum install mahout`”.
6. Use the executable `/usr/bin/mahout` to run our analysis.

Now we’ve installed the Hadoop and Mahout on our virtual machine. Further configurations will be included in future reports.

## 18.5 Setting up a 6-node Hadoop cluster

In order to practice coding on Hadoop, we tried to build a 6-node Hadoop cluster with ClouderaManager.

Hardware specifications:

- Number of nodes: 5 Hadoop Nodes and 1 Manager Node
- Quad-core CPU on each node, 24 cores in total.
- 48GB memory in total, 8GB on each node
- One 750GB enterprise-level hard drive on each node
- Nodes are connected by a 8-port gigabit ethernet switch.
- Master node has two network cards, one for public access, one for internal network.

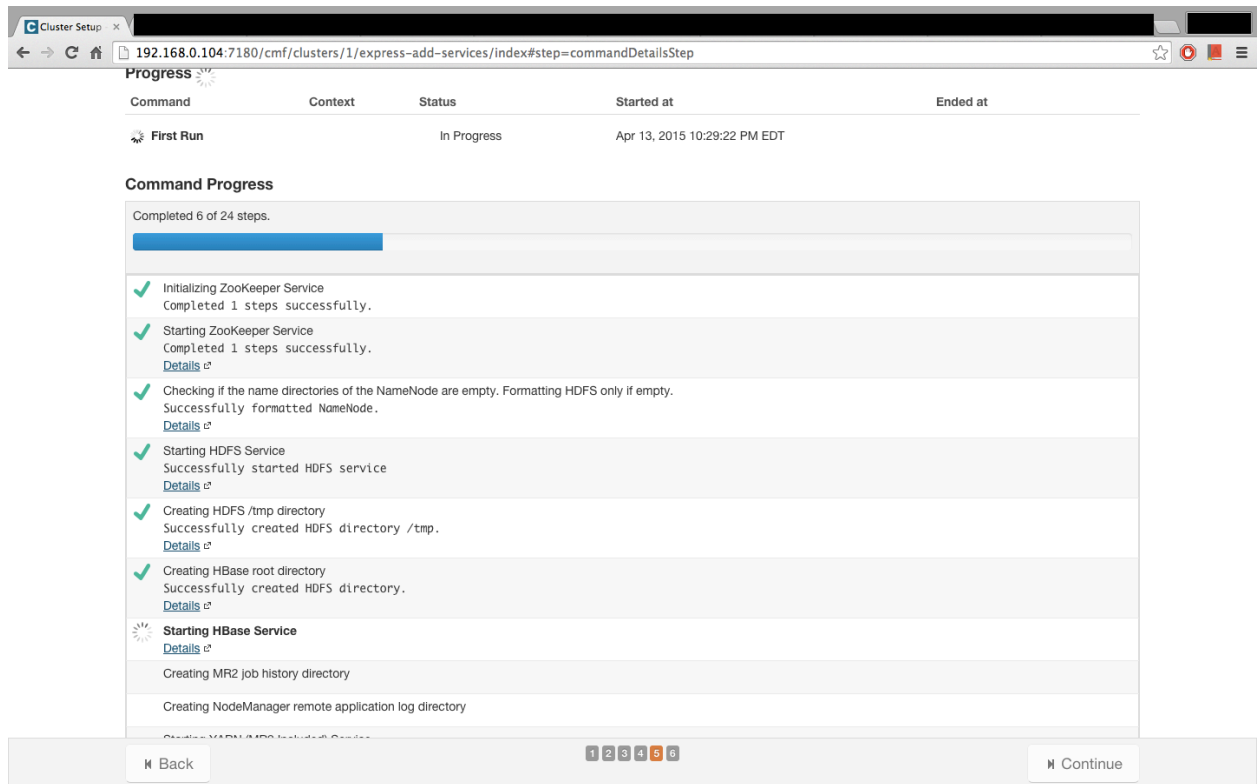
Procedures:

1. For each node, install CentOS 6.5 as the operating system.

2. In order to use ClouderaManager, selinux has to be disabled on manager node by editing `/etc/selinux/config`.
3. For each Hadoop (slave) node, the iptables firewall is disabled to avoid `NoRouteToHostException`.
4. The ntp service need to be enabled to synchronize time.
5. For all nodes, sshd server is enabled, as well as ssh root login.
6. For each node, the GUI had to be turned off by editing `/etc/inittab`.
7. For the manager (master) node, the iptables firewall is enabled, and all incoming traffic from public interface is denied except through Ports 22 and 50030, which corresponds to SSH service and Hadoop JobTracker service. Also, traffic through Ports 7180 should be accepted to allow controlling the services through cloudera on master node.
8. A forwarding rule has been set up with NAT on master node so that slave nodes can get access to internet.
9. For each Hadoop node, disable NetworkManager service and enable network service.
10. Download and install ClouderaManager with the command

```
wget http://archive.cloudera.com/cm5/installer/latest/cloudera-manager-installer.bin
chmod u+x cloudera-manager-installer.bin;sudo ./cloudera-manager-installer.bin
```

11. Manually assign an IP address, Gateways, etc. for each node.
12. Use a web browser to install CDH by visiting the manager node through port 7180.
13. In "License" section, select "Cloudera Express" option.
14. In "Specify hosts" section, type in the IP addresses of each node in the cluster. The installation program will automatically install Hadoop on each node.
15. Select the basic services (HDFS, MapReduce, YARN, Zookeeper, Oozie, Hive, Pigs, etc.) and the HBase service.
16. Assign the role for each node, here we will select the master node as the NameNode, SecondaryNameNode, and the HBase Master, and select the five slave nodes as the Data Nodes and HBase Region servers. Then manually select the corresponding services for each node.
17. Waiting for the installation to complete.



*Figure 15: Hadoop installation*

18. Here we skipped several tiny issues that may cause problems, since they can be easily solved by looking up ClouderaManager documents.
19. Run the PiTest and WordCount programs to examine if the installation is successful.

## 19 References

- [1] Steven Bird, Ewan Klein, and Edward Loper, *Natural language processing with Python*. O'Reilly Media, 2009.
- [2] NLTK project, NLTK 3.0 documentation. <http://www.nltk.org>, accessed on 02/05/2015.
- [3] Oracle, Java SE Development Kit 7 Downloads.  
<http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html>, accessed on 02/05/2015.
- [4] The Apache Software Foundation, Solr Download. <http://lucene.apache.org/solr/mirrors-solr-latest-redirect.html>, accessed on 02/05/2015.
- [5] Leonard Richardson, Beautiful Soup Documentation.  
<http://www.crummy.com/software/BeautifulSoup/bs4/doc/>, accessed on 02/05/2015.
- [6] The Apache Software Foundation, Solr Quick Start. <http://lucene.apache.org/solr/quickstart.html>, accessed on 02/05/2015.
- [7] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze, *Introduction to information retrieval*. Vol. 1. Cambridge: Cambridge University Press, 2008.
- [8] Alex J. Champandard, The Easy Way to Extract Useful Text from Arbitrary HTML. <http://ai-depot.com/articles/the-easy-way-to-extract-useful-text-from-arbitrary-html/>, accessed on 02/05/2015
- [9] HossMan, YonikSeeley, OtisGospodnetic, et al. Solr: Analyzers, Tokenizers, and Token Filters.  
<https://wiki.apache.org/solr/AnalyzersTokenizersTokenFilters> accessed on 02/03/2015
- [10] Apache Infrastructure Team, Solr: Solr Schema.  
[http://svn.apache.org/repos/asf/lucene/dev/branches/lucene\\_solr\\_3\\_6/solr/example/solr/conf/schema.xml](http://svn.apache.org/repos/asf/lucene/dev/branches/lucene_solr_3_6/solr/example/solr/conf/schema.xml) accessed on 02/03/2015
- [11] Joseph Acanfora, Stanislaw Antol, Souleiman Ayoub, et al. Vtechworks: CS4984 Computational Linguistics. <https://vtechworks.lib.vt.edu/handle/10919/50956> accessed on 02/05/2015
- [12] Arjun Chandrasekaran, Saurav Sharma, Peter Sulucz, and Jonathan Tran, Generating an Intelligent Human-Readable Summary of a Shooting Event from a Large Collection of Webpages Report for Course CS4984. <https://vtechworks.lib.vt.edu/handle/10919/51137> accessed on 02/05/2015
- [13] Apache Software Foundation, Mahout. <http://mahout.apache.org/>, accessed on 02/13/2015.
- [14] Edureka, Apache Mahout Tutorial,  
<https://www.youtube.com/watch?v=zvfKH9Yb0s0&list=PL9ooVrP1hQOGbSzlhdjb47SFMDC6bK6BW>, accessed on 02/13/2015.

- [15] Cloudera, Cloudera Installation and Upgrade. <http://www.cloudera.com/content/cloudera/en/documentation/core/latest/PDF/cloudera-installation.pdf>, accessed on 02/13/2015.
- [16] Steffen Nissen, Reference Manual for FANN 2.2.0. <http://leenissen.dk/fann/html/files/fann-h.html>, accessed on 02/13/2015.
- [17] Ari Pollak, Include OutputFormat for a specified AVRO schema that works with Streaming. <https://issues.apache.org/jira/browse/AVRO-1067>, accessed on 03/29/2015.
- [18] Michael G. Noll, Using AVRO in MapReduce Jobs With Hadoop, Pig, Hive. <http://www.michael-noll.com/blog/2013/07/04/using-avro-in-mapreduce-jobs-with-hadoop-pig-hive/>, accessed on 03/29/2015.
- [19] Arc90, Readability, <http://lab.arc90.com/2009/03/02/readability/>, accessed on 05/05/2015.
- [20] Nakatani Shuyo, Language Detection Library for Java, <http://www.slideshare.net/shuyo/language-detection-library-for-java>, accessed on 05/06/2015.
- [21] Python Software Foundation, Regular expressions, <https://docs.python.org/2/library/re.html>, accessed on 02/05/2015
- [22] Python Software Foundation, UUID, <https://docs.python.org/2/library/uuid.html>, accessed on 02/05/2015
- [23] Google, List of bad words, <http://fffff.at/googles-official-list-of-bad-words/>, accessed on 05/01/2015

## Appendix A: AVRO Schema for Tweets and Web pages

### AVRO schema for tweet collections

```
{"namespace": "cs5604.tweet.NoiseReduction",
  "type": "record",
  "name": "TweetNoiseReduction",
  "fields": [
    {"name": "doc_id", "type": "string"},
    {"doc": "original", "name": "tweet_id", "type": "string"},
    {"doc": "original", "name": "text_clean", "type": "string"},
    {"doc": "original", "name": "text_original", "type": "string"},
    {"doc": "original", "name": "created_at", "type": "string"},
    {"doc": "original", "name": "user_screen_name", "type": "string"},
    {"doc": "original", "name": "user_id", "type": ["string", "null"]},
    {"doc": "original", "name": "source", "type": ["string", "null"]},
    {"doc": "original", "name": "lang", "type": ["string", "null"]},
    {"doc": "original", "name": "favorite_count", "type": ["int", "null"]},
    {"doc": "original", "name": "retweet_count", "type": ["int", "null"]},
    {"doc": "original", "name": "contributors_id", "type": ["string", "null"]},
    {"doc": "original", "name": "coordinates", "type": ["string", "null"]},
    {"doc": "original", "name": "urls", "type": ["string", "null"]},
    {"doc": "original", "name": "hashtags", "type": ["string", "null"]},
    {"doc": "original", "name": "user_mentions_id", "type": ["string", "null"]},
    {"doc": "original", "name": "in_reply_to_user_id", "type": ["string", "null"]},
    {"doc": "original", "name": "in_reply_to_status_id", "type": ["string", "null"]}
  ]
}
```

### AVRO schema for web pages

```
{"namespace": "cs5604.webpage.NoiseReduction",
  "type": "record",
  "name": "WebpageNoiseReduction",
  "fields": [
    {"name": "doc_id", "type": "string"},
    {"doc": "original", "name": "text_clean", "type": ["null", "string"], "default": null},
    {"doc": "original", "name": "text_original", "type": ["null", "string"], "default":
null},
    {"doc": "original", "name": "created_at", "type": ["null", "string"], "default":
null},
    {"doc": "original", "name": "accessed_at", "type": ["null", "string"], "default":
null},
  ]
}
```

```

    {"doc": "original", "name": "author", "type": ["null", "string"], "default": null},
    {"doc": "original", "name": "subtitle", "type": ["null", "string"], "default": null},
    {"doc": "original", "name": "section", "type": ["null", "string"], "default": null},
    {"doc": "original", "name": "lang", "type": ["null", "string"], "default": null},
    {"doc": "original", "name": "coordinates", "type": ["null", "string"], "default":
null},
    {"doc": "original", "name": "urls", "type": ["null", "string"], "default": null},
    {"doc": "original", "name": "content_type", "type": ["null", "string"], "default":
null},

    {"doc": "original", "name": "text_clean2", "type": ["null", "string"], "default":
null},
    {"doc": "original", "name": "collection", "type": ["null", "string"], "default": null},
    {"doc": "original", "name": "title", "type": ["null", "string"], "default": null},
    {"doc": "original", "name": "domain", "type": ["null", "string"], "default": null},
    {"doc": "original", "name": "url", "type": ["null", "string"], "default": null},
4/22/2015

    {"doc": "original", "name": "appears_in_tweet_ids", "type": ["null", "string"],
"default": null} 4/21/2015
  ]}

```

## Appendix B: Source code of cleaning script

### Python script for tweet cleanup

```
#!/usr/bin/env python
# coding: utf-8
import sys
import re
import avro
import avro.schema
from avro.datafile import DataFileReader, DataFileWriter
from avro.io import DatumReader, DatumWriter
import json
from nltk.stem.lancaster import LancasterStemmer
import time
import uuid

st = LancasterStemmer()
__doc_id__ = 'charlie_hebdo_S'
__stop_word__ = [u'i', u'me', u'my', u'myself', u'we', u'our', u'ours',
                 u'ourselves', u'you', u'your', u'yours', u'yourself', u'yourself', u'he',
                 u'him', u'his', u'himself', u'she', u'her', u'hers', u'herself', u'it',
                 u'its',
                 u'itself', u'they', u'them', u'their', u'theirs', u'themselves', u'what',
                 u'which',
                 u'who', u'whom', u'this', u'that', u'these', u'those', u'am', u'is',
                 u'are', u'was',
                 u'were', u'be', u'been', u'being', u'have', u'has', u'had', u'having',
                 u'do', u'does',
                 u'did', u'doing', u'a', u'an', u'the', u'and', u'but', u'if', u'or',
                 u'because', u'as',
                 u'until', u'while', u'of', u'at', u'by', u'for', u'with', u'about',
                 u'against', u'between',
                 u'into', u'through', u'during', u'before', u'after', u'above', u'below',
                 u'to', u'from',
                 u'up', u'down', u'in', u'out', u'on', u'off', u'over', u'under', u'again',
                 u'further',
                 u'then', u'once', u'here', u'there', u'when', u'where', u'why', u'how',
                 u'all', u'any',
                 u'both', u'each', u'few', u'more', u'most', u'other', u'some', u'such',
                 u'no', u'nor',
                 u'not', u'only', u'own', u'same', u'so', u'than', u'too', u'very', u's',
                 u't', u'can',
                 u'will', u'just', u'don', u'should', u'now']

with open('profanity_en.txt') as f:
    __profanity_words__ = f.read()[:-1].split('\n')
```



```

f.close()

def validate(url): # task 4
    return 1

def cleanup(tweet):
    # task 1
    # remove emoticon
    try: # Wide UCS-4
        EmoRegex = re.compile(u'[
            u'\U0001F300-\U0001F64F'
            u'\U0001F680-\U0001F6FF'
            u'\u2600-\u26FF\u2700-\u27BF]+',
            re.UNICODE)
    except re.error: # Narrow UCS-2
        EmoRegex = re.compile(u>('
            u'\ud83c[\udf00-\udfff]|'
            u'\ud83d[\udc00-\ude4f\ude80-\udeff]|'
            u'[\u2600-\u26FF\u2700-\u27BF]+',
            re.UNICODE)

    tweet = EmoRegex.sub(' ', tweet)
    tweet = re.sub(r'\s', ' ', tweet)
    tweet = re.sub(r'\s', '', tweet)
    # task 2
    # Remove non-alphanumeric characters
    HashtagRegex = r'(?<=^|(?<=[^a-zA-Z0-9-\.]#)([A-Za-z_]+[A-Za-z0-9_]+)'
    UserhandleRegex = r'(?<=^|(?<=[^a-zA-Z0-9-\.]#)([A-Za-z_]+[A-Za-z0-9_]+)'
    UrlRegex = r'(?P<url>https?://[a-zA-Z0-9\./-]+)'
    Hashtaglist = re.findall(HashtagRegex, tweet)
    for hashtag in Hashtaglist:
        tweet = tweet.replace('#' + hashtag, '')
    Userhandlelist = re.findall(UserhandleRegex, tweet)
    for userhandle in Userhandlelist:
        tweet = tweet.replace('@' + userhandle, '')
    url_list = re.findall(UrlRegex, tweet)
    for url in url_list:
        tweet = tweet.replace(url, '')
    tweet = re.sub(r'([\s\w]|_)+', '', tweet) # task 5 included
    clean_tweet_only = tweet
    for hashtag in Hashtaglist:
        tweet = tweet + ' #' + hashtag
    for userhandle in Userhandlelist:
        tweet = tweet + ' @' + userhandle
    # task 3
    # validating url
    ValidUrlRegex = re.compile(r'^(?:http|ftp)s?:/' # http:// or https://
        # domain...

```

```

r'(?:(?:[A-Z0-9](?:[A-Z0-9-]{0,61}[A-Z0-9])?\.)+(?:[A-Z]{2,6}\.?[A-Z0-9-]{2,}\.?)|'
r'localhost|' # localhost...
# ...or ip
r'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}'
r'(?::\d+)?' # optional port
r'(?:/?[/?]\S+)$', re.IGNORECASE)

for url in url_list:
    if ValidUrlRegexp.match(url) and validate(url):
        tweet = tweet + ' ' + url
tweet = re.sub(' +', ' ', tweet)
clean_tweet_only = ''.join(
    [w if len(w) > 0 else '' for w in clean_tweet_only])
if len(clean_tweet_only) == 0:
    return (None, None, None, None, None, None)
clean_tweet_only = ''.join(
    [w if ord(w) < 128 else '' for w in clean_tweet_only])
clean_tweet_only = ' '.join(
    [word for word in clean_tweet_only.split(' ') if word.lower() not in __stop_word__])
ProfanityRegexp = re.compile(
    r'(?<=^|(?<=[^a-zA-Z0-9-\.])(? + '|'.join(__profanity_words__ + r')(?=$|\W)',
re.IGNORECASE)
clean_tweet_only = re.sub(' +', ' ', clean_tweet_only)
clean_tweet_only = re.sub(r'\n', ' ', clean_tweet_only)
clean_tweet_only = re.sub('^RT |^ |$', '', clean_tweet_only)
clean_tweet_2 = re.sub(ProfanityRegexp, '', clean_tweet_only)
clean_tweet_only = re.sub(
    ProfanityRegexp, '{"profanity"}', clean_tweet_only)
clean_tweet_2 = ' '.join([st.stem(word.lower())
    for word in clean_tweet_2.split(' ')])
clean_tweet_2 = re.sub(' +', ' ', clean_tweet_2)
clean_tweet_2 = re.sub('^ |$', '', clean_tweet_2)
return (tweet, clean_tweet_2, clean_tweet_only, Hashtaglist, Userhandlelist, url_list)

def checknone(str): # convert empty string to None
    if str == '' or str == u'':
        return None
    else:
        return str

def avrocleaning(filename1, filename2, filename3, doc_id):
    try:
        InFile = open(filename1, 'r')
        OutFile = open(filename2, 'w')
        SchemaFile = open(filename3, 'r')
    except IOError:
        print 'please check the filenames in arguments'

```

```

    return 0
reader = DataFileReader(InFile, DatumReader())
schema = avro.schema.parse(ShemaFile.read())
writer = DataFileWriter(OutFile, DatumWriter(), schema)
tweet_count = 0
clean_tweet_count = 0
for full_tweet_json in reader:
    tweet_count += 1
    if tweet_count % 25000 == 0:
        continue
    # if tweet_count > 100: break
    # remove leading and trailing whitespace
    try:
        # print full_tweet_json
        full_tweet = json.loads(json.dumps(full_tweet_json))
    except:
        continue
    # only select tweets in English
    if full_tweet[u'iso_language_code'] != u'en':
        # print 'not English'
        continue
    rawtweet = full_tweet[u'text'] # original tweet
    (clean_tweet, clean_tweet_2, clean_tweet_only, Hashtaglist,
     Userhandlelist, url_list) = cleanup(rawtweet)
    if clean_tweet is None:
        continue
    clean_tweet_count += 1
    full_clean_tweet = {}
    username = full_tweet[u'from_user']
    tweetID = full_tweet[u'id']
    user_id = full_tweet[u'from_user_id']
    timestamp = str(full_tweet[u'time']) # original 'time' is of type int
    source = checknone(full_tweet[u'archivesource'])
    in_reply_to_user_id = checknone(full_tweet[u'to_user_id'])
    geocord1, geocord2 = full_tweet[
        u'geo_coordinates_0'], full_tweet[u'geo_coordinates_1']
    full_clean_tweet['tweet_id'] = tweetID.encode('ascii', 'ignore')
    unique_id = uuid.uuid3(uuid.NAMESPACE_DNS, user_id.encode('ascii', 'ignore') +
        '_' + tweetID.encode('ascii', 'ignore'))
    full_clean_tweet['doc_id'] = doc_id + '--' + str(unique_id)
    full_clean_tweet['text_clean'] = clean_tweet_only.encode(
        'ascii', 'ignore')
    full_clean_tweet['text_clean2'] = clean_tweet_2.encode(
        'ascii', 'ignore')
    full_clean_tweet['text_original'] = rawtweet
    full_clean_tweet['created_at'] = timestamp.encode('ascii', 'ignore')
    full_clean_tweet['user_screen_name'] = username
    full_clean_tweet['user_id'] = user_id.encode('ascii', 'ignore')
    full_clean_tweet['lang'] = 'English'
    full_clean_tweet['collection'] = doc_id

```

```

    if float(geocord1) != 0.0 or float(geocord2) != 0.0:
        coordinate = '%s,%s' % (geocord1, geocord2)
        full_clean_tweet['coordinates'] = coordinate.encode(
            'ascii', 'ignore')
    if url_list != []:
        full_clean_tweet['urls'] = '|'.join(
            url_list).encode('ascii', 'ignore')
    if Userhandlelist != []:
        full_clean_tweet['user_mentions_id'] = '|'.join(
            Userhandlelist).encode('ascii', 'ignore')
    if Hashtaglist != []:
        full_clean_tweet['hashtags'] = '|'.join(
            Hashtaglist).encode('ascii', 'ignore')
    if source is not None:
        full_clean_tweet['source'] = source.encode('ascii', 'ignore')
    if in_reply_to_user_id is not None:
        full_clean_tweet['in_reply_to_user_id'] = in_reply_to_user_id.encode(
            'ascii', 'ignore')
    print full_clean_tweet
    writer.append(full_clean_tweet)
reader.close()
SchemaFile.close()
writer.close()
print filename1 + ' has been cleaned up'
print 'total tweets: %d' % tweet_count
print 'cleaned tweets: %d' % clean_tweet_count
return 1

def main(argv):
    try:
        InputFile = argv[1]
        OutputFile = argv[2]
        SchemaFile = argv[3]
    except IndexError:
        print 'Please specify the tweets input avro filename, output avro filename and avro
schema filename'
        return 0
    try:
        doc_id = argv[4]
    except IndexError:
        doc_id = __doc_id__
    return avrocleaning(InputFile, OutputFile, SchemaFile, doc_id)

if __name__ == '__main__':
    start_time = time.time()
    main(sys.argv)
    print("--- %s seconds ---\n\n" % (time.time() - start_time))

```

```
sys.exit(1)
```

## Python script for web page cleanup

```
#!/usr/bin/env python
# coding: utf-8

import sys
from readability.readability import Document
import re
from bs4 import BeautifulSoup
import avro
import avro.schema
from avro.datafile import DataFileWriter
from avro.io import DatumWriter
from nltk.stem.lancaster import LancasterStemmer
from urlparse import urlparse
from langdetect import detect_langs
import uuid
import time

st = LancasterStemmer()
__doc_id__ = 'charlie_hebdo_S'
__stop_word__ = [u'i', u'me', u'my', u'myself', u'we', u'our', u'ours',
                 u'ourselves', u'you', u'your', u'yours', u'yourself', u'yourself', u'he',
                 u'him', u'his', u'himself', u'she', u'her', u'hers', u'herself', u'it',
                 u'its',
                 u'itself', u'they', u'them', u'their', u'theirs', u'themselves', u'what',
                 u'which',
                 u'who', u'whom', u'this', u'that', u'these', u'those', u'am', u'is',
                 u'are', u'was',
                 u'were', u'be', u'been', u'being', u'have', u'has', u'had', u'having',
                 u'do', u'does',
                 u'did', u'doing', u'a', u'an', u'the', u'and', u'but', u'if', u'or',
                 u'because', u'as',
                 u'until', u'while', u'of', u'at', u'by', u'for', u'with', u'about',
                 u'against', u'between',
                 u'into', u'through', u'during', u'before', u'after', u'above', u'below',
                 u'to', u'from',
                 u'up', u'down', u'in', u'out', u'on', u'off', u'over', u'under', u'again',
                 u'further',
                 u'then', u'once', u'here', u'there', u'when', u'where', u'why', u'how',
                 u'all', u'any',
                 u'both', u'each', u'few', u'more', u'most', u'other', u'some', u'such',
                 u'no', u'nor',
```

```

        u'not', u'only', u'own', u'same', u'so', u'than', u'too', u'very', u's',
u't', u'can',
        u'will', u'just', u'don', u'should', u'now']

with open('profanity_en.txt') as f:
    __profanity_words__ = f.read()[::-1].split('\n')
f.close()

def cleanblankspaces(clean_content0):
    clean_content = clean_content0
    clean_content = re.sub(' . ', '. ', clean_content)
    clean_content = re.sub(' (,|;) ', ', ', clean_content)
    clean_content = re.sub(' +', ' ', clean_content)
    clean_content = re.sub('^ | $', '', clean_content)
    return clean_content

def webcontentcleanup(content):
    UrlRegexp = r'(?P<url>https?://[a-zA-Z0-9\./-]+)'
    ProfanityRegexp = re.compile(
        r'(?<=^|(?<=[^a-zA-Z0-9-\.])(? + '|' + '.join(__profanity_words__) + r')(?=$|\W)',
re.IGNORECASE)
    url_list = re.findall(UrlRegexp, content)
    for url in url_list:
        content = content.replace(url, '')
    clean_content_only = ' '.join(
        [word for word in content.split(' ') if word.lower() not in __stop_word__])
    clean_content_2 = re.sub(ProfanityRegexp, '', clean_content_only)
    clean_content_only = re.sub(ProfanityRegexp, '{"profanity"}', clean_content_2)
    clean_content_2 = ' '.join([st.stem(word)
                                for word in clean_content_2.split(' ')])
    clean_content_only = cleanblankspaces(clean_content_2)
    clean_content_2 = re.sub(r'([\s\w]|_)+', '', clean_content_2)
    clean_content_2 = cleanblankspaces(clean_content_2)
    url_list_str = '|'.join(url_list)
    return (clean_content_only, clean_content_2, url_list_str)

def avrocleaning(filename1, filename2, filename3, doc_id):
    try:
        InFile = open(filename1, 'r')
        OutFile = open(filename2, 'w')
        SchemaFile = open(filename3, 'r')
        fp = open('test.dat', 'w')
    except IOError:
        print 'please check the filenames in arguments'
        return 0
    raw_text_all = InFile.read().decode('utf8')

```

```

InFile.close()
schema = avro.schema.parse(SchemaFile.read())
writer = DataFileWriter(OutFile, DatumWriter(), schema)
regex_raw_webpage = re.compile(
    'url:.*?contentType:.*?Content:.*?Version: -1', re.DOTALL)
regex_webpage = re.compile('Content:.*Version: -1', re.DOTALL)
regex_url = re.compile(r'(?<=url: )http.*')
regex_contentType = re.compile(r'(?<=contentType: ).*')
regex_rubbish = re.compile('http.*Version: -1')
webpages = re.findall(
    regex_raw_webpage, raw_text_all + '\nhttp:TO_FIND_THE_LAST_WBBPAGE_Version: -1')
clean_webpage_count = 0
html_file_count = 0
contentTypeAll = {}
languageAll = {}
for raw_text in webpages:
    url = re.findall(regex_url, raw_text)[0].strip()
    contentType = re.findall(regex_contentType, raw_text)[0].strip()
    if contentType not in contentTypeAll:
        contentTypeAll[contentType] = 1
    else:
        contentTypeAll[contentType] += 1
    if contentType.find('html') < 0:
        continue
    html_file_count += 1
    raw_text = re.findall(regex_webpage, raw_text)[0]
    raw_text = re.sub(regex_rubbish, '', raw_text)
    readable_article = Document(raw_text).summary()
    readable_title = Document(raw_text).short_title()
    readable_title = ''.join(
        [i if ord(i) < 128 else ' ' for i in readable_title])
    # url = ''.join([i if ord(i) < 128 for i in url])
    url = url.decode("utf8")
    readable_title = re.sub(' +', ' ', readable_title)
    soup = BeautifulSoup(readable_article)
    texts = soup.findAll(text=True)
    all_text = ' '.join(texts).strip()
    try:
        lan = str(detect_langs(all_text)[0]).split(':')[0]
    except:
        continue
    if lan not in languageAll:
        languageAll[lan] = 1
    else:
        languageAll[lan] += 1
    if lan != 'en':
        continue
    all_text = all_text.replace('\r\n', ' ')
    all_text = all_text.replace('\n', ' ')
    all_text = all_text.replace('\t', ' ')

```

```

    all_text = ''.join([i if ord(i) < 128 else ' ' for i in all_text])
    all_text = re.sub(' +', ' ', all_text)
    (clean_content_only, clean_content_2, url_list_str) = webcontentcleanup(all_text)
    # print clean_content_2
    domain = '{uri.netloc}'.format(uri=urlparse(url))
    webpage_id = str(uuid.uuid3(uuid.NAMESPACE_DNS, url.encode('ascii', 'ignore')))
    webpage_json = {}
    webpage_json["doc_id"] = doc_id + '--webpage--' + webpage_id
    webpage_json["text_clean"] = clean_content_only
    webpage_json["text_original"] = raw_text
    webpage_json["title"] = readable_title
    webpage_json["text_clean2"] = clean_content_2
    webpage_json["collection"] = doc_id
    webpage_json["content_type"] = 'html'
    webpage_json["urls"] = url_list_str
    webpage_json["domain"] = domain
    webpage_json["url"] = url
    writer.append(webpage_json)
    clean_webpage_count += 1
    fp.write("%s\n%s\n\n%s\n\n\n\n\n\n" %
            (doc_id + '--webpage--' + webpage_id, url, clean_content_only))

fp.close()
SchemaFile.close()
writer.close()
print filename1 + ' has been cleaned up'
print 'Total webpages: %d' % len(webpages)
print 'Cleaned webpages: %d' % clean_webpage_count
print 'Percentage cleaned: %.3f' % (100.0*clean_webpage_count/len(webpages))
print 'HTML webpages: %d' % html_file_count
print 'Non-English webpages: %d' % (html_file_count-clean_webpage_count)
print 'Content Type Statistics: ', contentTypeAll
print 'Language Statitics: ', languageAll
return 1

def main(argv):
    try:
        InputFile = argv[1]
        OutputFile = argv[2]
        SchemaFile = argv[3]
    except IndexError:
        print 'Please specify the webpage input avro filename, output avro filename and avro
schema filename'
        return 0
    try:
        doc_id = argv[4]
    except IndexError:
        doc_id = __doc_id__
    return avrocleaning(InputFile, OutputFile, SchemaFile, doc_id)

```



```

if __name__ == '__main__':
    start_time = time.time()
    main(sys.argv)
    print("--- %s seconds ---\n\n\n" % (time.time() - start_time))
    sys.exit(1)

```

## Python code for cleaning up tweets with MapReduce (Raw version)

```

#!/usr/bin/env python
import sys
import re

doc_id = 'charlie_hebdo_S'

def cleanup(tweet):
    # task 1
    # remove emoticon
    try: # Wide UCS-4
        EmoRegex = re.compile(u'[
            u'\U0001F300-\U0001F64F'
            u'\U0001F680-\U0001F6FF'
            u'\u2600-\u26FF\u2700-\u27BF]+',
            re.UNICODE)
    except re.error: # Narrow UCS-2
        EmoRegex = re.compile(u'[
            u'\ud83c[\udf00-\udfff]|'
            u'\ud83d[\udc00-\ude4f\ude80-\udeff]|'
            u'[\u2600-\u26FF\u2700-\u27BF]+',
            re.UNICODE)

    tweet = EmoRegex.sub(' ', tweet)
    tweet = re.sub(r'\s', ' ', tweet)

    # task 2
    # Remove non-alphanumeric characters
    HashtagRegex = r'(?<=^|(?<=[^a-zA-Z0-9-\.]#)([A-Za-z_]+[A-Za-z0-9_]+)'
    UserhandleRegex = r'(?<=^|(?<=[^a-zA-Z0-9-\.]@)([A-Za-z_]+[A-Za-z0-9_]+)'
    UrlRegex = r'(?P<url>https?://[a-zA-Z0-9\./-]+)'
    Hashtaglist = re.findall(HashtagRegex, tweet)
    for hashtag in Hashtaglist:
        tweet = tweet.replace('#' + hashtag, '')
    Userhandlelist = re.findall(UserhandleRegex, tweet)
    for userhandle in Userhandlelist:
        tweet = tweet.replace('@' + userhandle, '')
    url_list = re.findall(UrlRegex, tweet)
    for url in url_list:
        tweet = tweet.replace(url, '')
    tweet = re.sub(r'([\s\w]|_)+', ' ', tweet) # task 5 included
    clean_tweet_only = tweet
    for hashtag in Hashtaglist:
        tweet = tweet + ' #' + hashtag

```

```

for userhandle in Userhandlelist:
    tweet = tweet + ' @' + userhandle
# task 3
# validating url
ValidUrlRegexp = re.compile(r'^(?:http|ftp)s?:/' # http:// or https://
                             # domain...
                             r'(?:(?:[A-Z0-9](?:[A-Z0-9-]{0,61}[A-Z0-9])?\.)+(?:[A-
Z]{2,6}\.\.?|[A-Z0-9-]{2,}\.?)|'
                             r'localhost|' # localhost...
                             # ...or ip
                             r'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})'
                             r'(?::\d+)?' # optional port
                             r'(?:/|?[/?]\S+)$', re.IGNORECASE)

for url in url_list:
    if ValidUrlRegexp.match(url):
        tweet = tweet + ' ' + url
tweet = re.sub(' +', ' ', tweet)
return (tweet, clean_tweet_only, Hashtaglist, Userhandlelist, url_list)

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line
    content = line.split(',')
    if len(content) < 14:
        continue
    if content[6] != 'en': # only select tweets in English
        continue
    source = content[0]
    rawtweet = content[1]
    (cleanedtweet, clean_tweet_only, Hashtaglist,
     Userhandlelist, url_list) = cleanup(content[1])
    hashtags = ','.join(Hashtaglist)
    urls = ','.join(url_list)
    user_mentions_id = ','.join(Userhandlelist)
    username = content[3]
    tweetID = content[4]
    user_id = content[5]
    if content[10] == 0.0 and content[11] == 0.0:
        coordinate = ''
    else:
        coordinate = '%s,%s' % (content[10], content[11])
    timestamp = content[13]
    json_string = {}
    json_string["doc_id"] = doc_id+'_'+tweetID
    json_string["text_clean"] = clean_tweet_only
    json_string["tweet_id"] = tweetID
    json_string["text_original"] = line
    json_string["user_screen_name"] = username
    json_string["user_id"] = user_id
    json_string["created_at"] = timestamp
    json_string["source"] = source
    if hashtags != '':
        json_string["hashtags"] = hashtags
    if urls != '':

```

```
    json_string["urls"] = urls
if username != '':
    json_string["user_mentions_id"] = username
if coordinate != '':
    json_string["coordinates"] = coordinate
print '%s' % json_string
```

## Appendix C: Code provided by TAs

### Crawl script for Nutch

```
if false
then
# note that the link inversion - indexing routine can be done within the main loop
# on a per segment basis
echo "Link inversion"
"$bin/nutch" invertlinks "$CRAWL_PATH"/linkdb "$CRAWL_PATH"/segments/$SEGMENT
if [ $? -ne 0 ]
then exit $?
fi
echo "Dedup on crawlddb"
$bin/nutch dedup $CRAWL_PATH/crawlddb
if [ $? -ne 0 ]
then exit $?
fi
echo "Indexing $SEGMENT on SOLR index -> $SOLRURL"
"$bin/nutch" index -D solr.server.url=$SOLRURL "$CRAWL_PATH"/crawlddb -linkdb
"$CRAWL_PATH"/linkdb "$CRAWL_PATH"/segments/$SEGMENT
if [ $? -ne 0 ]
then exit $?
fi
echo "Cleanup on SOLR index -> $SOLRURL"
"$bin/nutch" clean -D solr.server.url=$SOLRURL "$CRAWL_PATH"/crawlddb

if [ $? -ne 0 ]
then exit $?
fi
fi
done
```

### URL extraction from tweets

```
import sys
import requests
#import hashlib
from bs4 import BeautifulSoup, Comment
import re
#import sunburnt
#import pymysql
```

```

from operator import itemgetter
from contextlib import closing
requests.packages.urllib3.disable_warnings()

headers = {'User-Agent': 'Digital Library Research Laboratory (DLRL)'}

def visible(element):
    if element.parent.name in ['style', 'script', '[document]', 'head']:
        return False
    return True

thresh =1
archiveID = sys.argv[1].split(".")[0]

tweetFile = sys.argv[1]

tweets = []
f = open(tweetFile,"r")
us = f.readlines()
f.close()
for l in us[1:]:
    l = l.strip()
    p = l.split(",")
    t = p[0]
    tweets.append(t)

docs=[]

print "tweets is read from File"

# Extract short URLs from Tweets
#-----
shortURLsList =[]
#for row in cursor.fetchall():
for line in tweets:
    #line = row[1]
    regexp = "(?P<url>https?://[a-zA-Z0-9\.-]+)"
    url_li = re.findall(regExp, line) # find all short urls in a single tweet
    while (len(url_li) > 0):
        shortURLsList.append(url_li.pop())
print "short Urls extracted: ", len(shortURLsList)
surls = []
for url in shortURLsList:
    i = url.rfind("/")
    if i+1 >= len(url):
        continue
    p = url[i+1:]
    if len(p) < 10:
        continue
    while url.endswith("."):

```

```

        url = url[:-1]
        surls.append(url)
print "cleaned short URLs: ", len(surls)
surlsDic ={}
for url in surls:
    if url in surlsDic:
        surlsDic[url] = surlsDic[url] + 1
    else:
        surlsDic[url] = 1
print "Unique short URLs: ", len(surlsDic)
sorted_list = sorted(surlsDic.iteritems(), key=itemgetter(1), reverse=True)

freqShortURLs =[]

for surl,v in sorted_list:
    if v > thresh:
        freqShortURLs.append(surl)
print "Freq short URLs (>"+str(thresh)+"): ",len(freqShortURLs)

fs = open("shortURLs_" + archiveID + ".txt", "w")
for surl,v in sorted_list:
    fs.write(surl + "," + str(v)+"\n")
fs.close()
# Expand Short URLs
#-----
expanded_url_dict = {}
i=0
e=0
webpages=[]
for url in freqShortURLs:
    try:
        with closing(requests.get(url,timeout=10, stream=True,
verify=False,headers=headers)) as r:
            #page = r.text or r.content
            if r.status_code == requests.codes.ok:

                ori_url =r.url

                if ori_url != "":
                    # add the expanded original urls to a python dictionary with their count
                    if ori_url in expanded_url_dict:
                        expanded_url_dict[ori_url].append(url)
                    else:
                        expanded_url_dict[ori_url] = [url]
                        i+=1
                        ...

                    page = r.content or r.text
                    soup = BeautifulSoup(page)
                    title = ""
                    text = ""

```

```

        if soup.title:
            if soup.title.string:
                title = soup.title.string

        comments = soup.findAll(text=lambda text:isinstance(text,Comment))
        [comment.extract() for comment in comments]
        text_nodes = soup.findAll(text=True)

        visible_text = filter(visible, text_nodes)
        text = ''.join(visible_text)
        #text = title + " " + text
        webpages.append((url,title, text))
        ...

    else:
        e = e+1
except :
    print sys.exc_info()[0],url
    e = e +1
print "Unique Orig URLs expanded: ", i
print "Bad URLs: ",e

#print "Unique Orig URLs: ", len(expanded_url_dict)
fo = open('seedsURLs_'+archiveID+'.txt','w')
fs = open("short_origURLsMapping_" + archiveID + ".txt","w")
for ourl,surls in expanded_url_dict.items():
    fs.write(ourl +":--"+",".join(surls)+"\n")
    fo.write(ourl+'\n')
fs.close()
fo.close()
...

#Saving Webpages text to file
i=1
for wp in webpages:
    f = open(str(i)+'.txt','w')
    cont = wp[1]+ ' ' + wp[2]
    f.write(cont.encode('utf8'))
    f.close()
    i+=1

print "Webpages text saved"
...

```