CS5604 Information Storage and Retrieval

Term Project-Collection Management

Final Report

May 4, 2016

Virginia Tech

Blacksburg, VA 24061

Team Members:

Yufeng Ma

Dong Nan

Instructor:

Prof. Edward A. Fox

Executive Summary

The collection management portion of the information retrieval system has three major tasks. The first task is to perform incremental update of the new data flow from the tweet MySQL database to HDFS and then to HBase. Secondly, for the raw tweets coming into HBase, we are supposed to clean them. Duplicated URLs should be discarded. Also important is to conduct noise reduction. Finally, for the cleaned tweets and webpages, we are going to do Named Entity Recognition (NER), from which we extract out the information like person, organization, and location names.

First, based on existing data flow from tweet MySQL database to HBase in the IDEAL system, we developed Sqoop script to import new tweets from MySQL to HDFS. Then another Pig script is run to transfer them into HBase. Afterwards, for raw tweets in HBase, we ran noise reduction module to remove non-ASCII characters, extract hashtags, mentions and URLs from tweet text. Similar procedures were also performed for raw webpage records provided by the GRAs for this project. All the cleaned data for the 6 small collections have uploaded into HBase with pre-defined schemas documented in this report. Then all the other teams like classification and clustering can consume our cleaned data.

Besides done so far, we are also supposed to do NER, which tries to extract structured information such as person, organization and location from unstructured text. But due to time limitations, we will take this as part of our future work. Plus, we will focus our time on automating webpage crawling and cleaning process, which is essential after incremental update. It will expand URLs extracted from tweets in HBase first, and then crawl the corresponding webpages after invalid URL removal. Finally, extracted useful information in webpage will be stored into HBase.

Table of Contents

Table of Figures

Table of Tables

1. 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. CollectionManagementReport.pdf
    a. The PDF format of the term report that describes, in detail, our work for the project.
2. CollectionManagementReport.docx
    a. An editable Word format of the term report, CollectionManagementReport.pdf
3. CollectionManagementPresentation.pdf
    a. The PDF format of the presentation (slides) that provide an overview of our work for the project.
4. CollectionManagementPresentation.pptx
    a. An editable PowerPoint format of the presentation, CollectionManagementPresentation.pdf
5. CollectionMangementCode&Data.zip
    a. A compressed folder that contains the source code for our tweet and webpage cleanup implementation, and the corresponding dataset.
    b. Folder contains:
        I. Tweet cleaning
        II. Webpage cleaning

2. Overview

Our project for Collection Management can be divided into three parts. In the beginning, we plan to conduct incremental updates on the new data flow from the tweets database to HDFS, and then store them in HBase. Then, we are going to perform Noise Reduction on the raw tweets and webpages from HDFS to build new attributes and get the cleaned text data. These data can be provided to other teams for their work. At last, we will do Named Entity Recognition on the cleaned data to extract the structured information such as person, organization and location, and the extracted information will be stored in HBase as well.

The challenges we are facing now are to improve the performance of NER and noise-reducing with large datasets, and the integration of NER and noise-reducing modules with the whole system. The complete datasets include tweets datasets (about 1 TB) and webpages datasets (about 11 TB). The NER and noise-reducing module can perform well on some datasets while be unacceptable on others. So with the complete huge dataset like in our situation, the performance should be guaranteed on a reasonable level. Also, since we plan to take and improve the NER module and noise-reducing module developed from last year, there could be some problems when they are integrated into the current system. So to make sure the whole system will work well with these modules, we need to fix the possible conflicts when merging them.

The solution we plan to develop is to provide proper APIs to ensure our functionalities integrate well with the system. We plan to use Sqoop to process the new data flow from tweets database to HDFS, and merge the previous NER and noise-reducing modules into the system.

For further work, we are going to maintain the parts that we developed in the system, try more test cases like the big collections on their performance and improve their functionalities to make them work better. Moreover, we plan to refactor the modules so that it would be easier for adding or deleting features from the entire system in the future. Also, the automation of webpage crawling and cleaning is essential for us to implement later.

3. Literature Review

From the Information Retrieval textbook [3], we find that chapters 2, 13, 14 and 15 are related to the works of Named Entity Recognition. Chapter 2 introduces information preprocessing and retrieval, for example, tokenization, stop word removal and normalization. These can be the prerequisites of NER and will increase the accuracy of the work. Chapters 13 to 15 are related to classification, and introduce feature selection, document modeling and support vector machines. These parts enhance our understanding of working mechanisms of NER and other modules of the information retrieval system. We also found a few good papers about NER. A survey of existing NER technologies is presented by Nadeau and Sekine [4]. Zhou and Su [5] proposed a HMM-based chunk tagger for NER. Ratinov and Roth [6] investigated the common challenges and misconceptions in NER, and proposed corresponding solutions.

For the work of noise-reducing, we find chapters 2 and 10 are useful. In chapter 2, there are methods of word segmentation, true casing and coding or language detection in documents. Chapter 10 talks about information retrieval from structured documents, for instance, XML files and HTML files, which are exactly what we need in order to extract valuable features and decrease the document size.

Also, we find that the resources for text processing in Python are of great value in completing our work. There is an open source library called Natural Language Toolkit (NLTK) [7] which is developed for language processing in Python. And there is a reference book named “Natural Language Processing with Python” [8] that systematically introduces the working mechanisms of NLTK and how to clean up documents with it. Also in this book, chapter 3 provides us with raw text processing and chapter 6 introduces text classification, which we find might be useful to our work.

For incremental update, especially the incremental update from the relational database to HDFS, we mainly refer to the book named Apache Sqoop Cookbook [9]. Since we are using Sqoop to import data from the relational database to HDFS, we need to understand how it works on Hadoop. The previous Sqoop bash script provided by Sunshin is able to import all the data from the relational database to HDFS as a one-time operation. Our goal is to keep the data on Hadoop in sync with the source table on the relational database. However, if we use the previous script to simply reimport all the data every time there are new rows in the relational database, it takes a lot of time and create unnecessary burden on the database. In chapter 3 of this book, we get to know that Sqoop provides the ability to do incremental import, so that we modify the previous bash script and add the incremental feature which only append new rows from the relational database to HDFS to make the data import efficiently.

4. Requirements

As a collection management team, our final goal for the project is three-fold. First, we are supposed to extract and remove the "noise" in the tweets and associated web page data. Then based on these cleaned data, we will perform Named Entity Recognition to identify person, organization and location names. Also we need to implement the incremental update part from Relational Database to HDFS and finally into HBase. The detailed procedures can be summarized as follows:

1. For new tweets and webpages, we will have to do incremental update transfer from Relational Database to HDFS and then into HBase;

2. For each raw tweet stored in the HDFS system, we will identify and remove "noise" like the random emoticons, random characters, and stop words;

3. Process and standardize the cleaned or "sound" data to build multiple attributes like text column, language_name in HBase;

4. For short URLs in the tweets, we will extend them to long ones, and for all URLs we will remove duplicates when looking them up in WWW;

5. For the webpages associated with the valid URLs, we will try to remove advertising content, banners, and other such contents from the HTML page and only keep the actual "sound" text to be processed. Then we will store these "sound" webpages into HBase;

6. Finally, based on the cleaned data, Named Entity Recognition will be performed to recognize and build the person, organization and location name attributes for each tweet in HBase;

Functional Requirements

1. Reducing Noise function

   Given a document (tweet or webpage) in the HDFS system, the Reducing Noise module will produce structured output with multiple attributes or columns like language code, and cleaned texts without stop words.

2. NER function

   Given a document (web page or tweet) in HDFS, the NER module extracts the locations, person names, organization names, and any other special type of entities.

3. Incremental Update

   For new tweets and webpages in Relational Database, each record will have to be transferred to HDFS and finally into HBase.

Performance Requirements

1. For the Named Entity Recognition module, we would expect a good F1-measure like more than 80%.

2. For the Reducing Noise module, it is difficult to measure the performance quantitatively. Typically, we would measure it manually by looking at a small collection of tweets.

The following two tables shows our HBase schemas for cleaned tweets and webpages.

Table 1. Schema of cleaned tweet in HBase

Column Family/Columns / Rowkey	clean_tweet				
	clean_text	urls	mentions	hashtags	mappings
collection # - tweet id					

Table 2. Schema of cleaned webpage in HBase

Rowkey	Column Family										
	clean_web										
	collection	lang	domain	doc_id	text_clean	text_clean_profanity	title	urls	web_original	mappings	empty
URL											

Based on topic analysis team's new requirement, we add a new column under clean_web which is called empty. It indicates whether there are readable contents in this webpage.

5.Document Properties

Prior to cleaning the tweets and webpages, it is critical to understand and evaluate the structure of each document. Like in Figure 1, we colored and pointed out the useful features included in a typical tweet, which includes tweet ID, meaningful text itself, mentions, hashtags, etc. In Figure 2, as the webpage record is more unstructured, we only highlighted the most obvious features like tweet ID and URL. The visualized corresponding HTML code in a browser is shown in Figure 3.

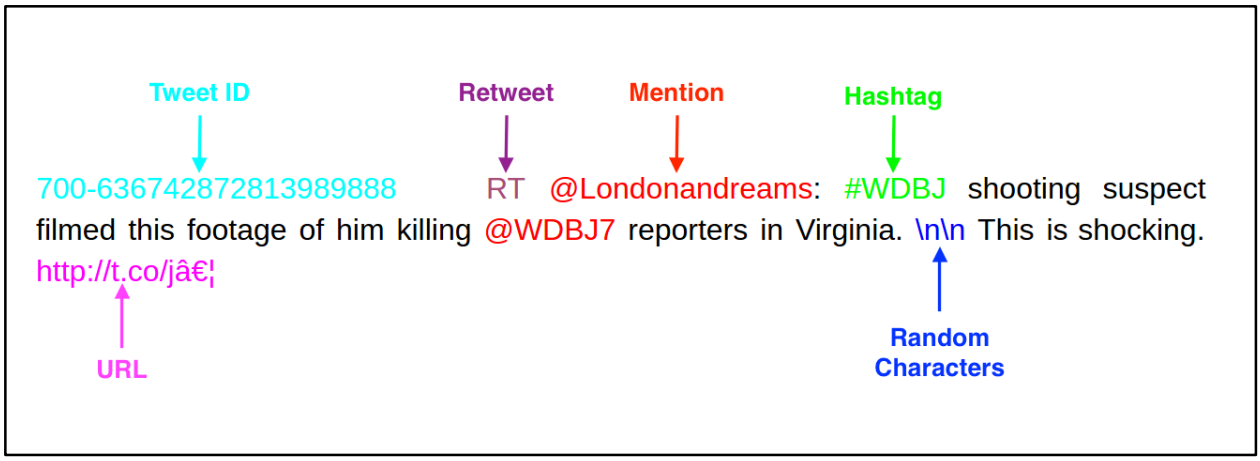

Figure 1. Example structure of a raw tweet record

Figure 2. Example structure of a webpage record

Figure 3. Example structure of a webpage in Brower

6. System Design

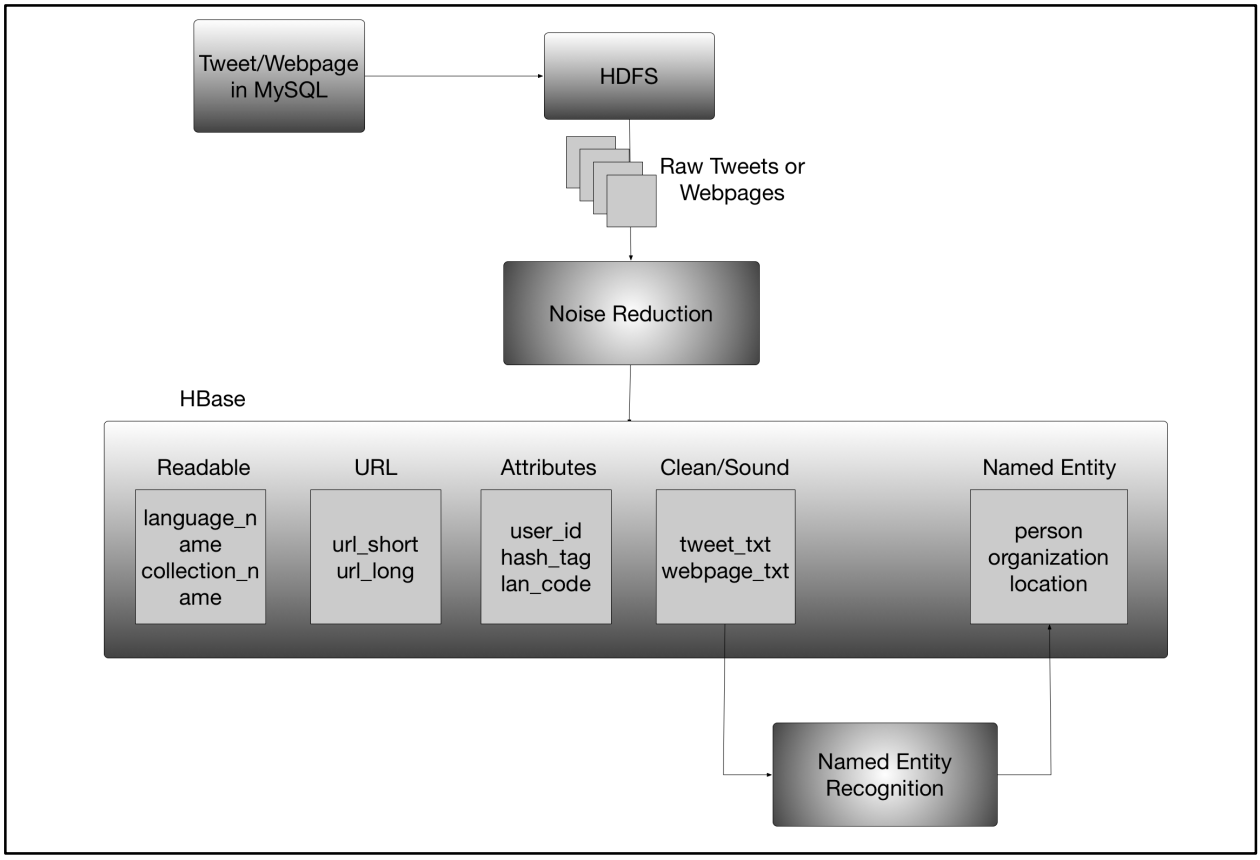

Figure 4. Design Diagram of Collection Management

The overall pipeline of processing is shown in Figure 4. First of all, incremental update is applied to transfer new data from a relational database to HDFS and finally into HBase. Then a Noise Reduction module is employed to retrieve tweets or webpages from the HDFS system, remove "noise" from these documents, and then store them in HBase. Thereafter based on the cleaned data, we will perform Named Entity Recognition to extract interesting named entities like person and organization names.

While designing our system, we are focusing on how the other components of the information retrieval system, especially the classification and cluster teams, consume the "cleaned or sound" data produced by our module. More specifically, our design seamlessly integrates with the frameworks designed by other teams.

In terms of this aspect, we formulate our output schema as follows.

For cleaned tweets and information extracted from the raw tweets, we append them in one table but with different columns.

Table 3. Schema of cleaned tweets

Field	Meaning

id	unique identifier for a raw tweet
text	a complete raw tweet
text_clean	a cleaned tweet without URLs, profanity words, stop words and non-characters
urls	URLs in the tweets
text_clean_stop	a cleaned tweet but with stop words
hashtags	hashtags in the tweet
mentions	mentions in the tweet

For webpages, we designed the schema in Figure 5 for our cleaned webpages.

```
{"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": "web_original", "type": ["null", "string"], "default": null},
  {"doc": "original", "name": "lang", "type": ["null", "string"], "default": null},
  {"doc": "original", "name": "urls", "type": ["null", "string"], "default": null},
  {"doc": "original", "name": "text_clean_profanity", "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": "url", "type": ["null", "string"], "default": null},
  {"doc": "original", "name": "tweet_id", "type": ["null", "string"], "default": null},
  {"doc": "original", "name": "domain", "type": ["null", "string"], "default": null}
]}
```

Figure 5. Cleaned Webpages Schema

You can see the detailed meanings of different fields for a cleaned webpage in Table 4.

Table 4. Meaning of the Fields in the AVRO schema

Field	Meaning
doc_id	unique identifier for the web page
text_clean	readable text in the web page
web_original	original HTML code for the web page
lang	language of the webpage; here we only keep webpages in English
urls	URLs in the webpage

text_clean_profanity	clean text without profanity words
collection	collection name for the web page, like shooting, disease
title	title of this web page
url	URL of this web page
tweet_id	tweet identifier of the tweet that contains this web page
domain	domain of the URL for this webpage

7. Implementation

7.1 Cleanup Implementation

7.1.1 Tweet cleanup

Instead of using Java or Python packages to do tweet cleaning, using Pig or Spark is more efficient and better to integrate with the existing system. Compared to Spark, Pig is even better in ETL (extract, transform, load) in HBase. So we decided to clean the tweets by Pig scripts.

Pig provides effective functions for us to extract information such as URLs, hash tags and mentions from raw tweets. Also, it is easy to remove contents like non-characters, stop words and profanities from the tweets in simple lines of code. Details can be found in the Appendix.

For the cleaned tweets to be stored into HBase, we designed the corresponding column family and columns as in Table 1. We ran the HBase shell command to load all the small collection tweets into it. You can see a clean_tweet record like in Figure 6.

```
hbase(main):004:0> get 'ideal-tweet-cs5604s16', '541-552940627672174595', {COLUMNS => 'clean_tweet'}
COLUMN                          CELL
 clean_tweet:clean_text         timestamp=1459454465065, value=FBI looks for motive in explosion near Colorado NAACP office  NAACPBombing
 clean_tweet:hashtags           timestamp=1459454578919, value=#NAACPBombing
 clean_tweet:mappings           timestamp=1459471464541, value=https://twitter.com/WMCActionNews5/status/552940627672174595/photo/1
 clean_tweet:mentions           timestamp=1459454624892, value=@NAACP
 clean_tweet:urls               timestamp=1459454530195, value=http://t.co/4OxJDhAENa
5 row(s) in 0.0090 seconds
```

Figure 6. Example of clean tweet in HBase

In some collections, there are many records where one URL is appended to another as shown in Figure 7. In previous stages, this kind of URLs are not extracted individually which makes it difficult for other teams to process the URL information.

```
700-637273438290341888  http://t.co/XIN1lqhtfbhttps://t.co/zj1u0FXXVm
700-637224833957408768  http://t.co/VtjW5OYopv
700-639164824564449280  http://t.co/X7QJJVNm9F
700-647150826331504641  http://t.co/GVw0MXFtzv
700-636677330698678272  http://t.co/0qYZdQWn5X
700-636677330698678272  http://t.co/T7JJ7wOlF7
700-637277165776732160  http://t.co/XIN1lqhtfbhttps://t.co/zj1u0FXXVm
```

Figure 7. Extracted URLs where one is appended to another

To solve this problem, the script for URL extraction is modified to detect these URLs that are close together. Using regular expression, different URLs are extracted individually and tabs are inserted between them as shown in Figure 8. In this way, other teams are able to read these URLs separately for further processing.

```
700-637289389790507009  http://t.co/XIN1lqhtfb  https://t.co/zj1u0FXXVm
700-637269734917054464  http://t.co/XIN1lqhtfb  https://t.co/zj1u0FXXVm
700-637269943487041537  http://t.co/XIN1lqhtfb  https://t.co/zj1u0FXXVm
700-637269083248029697  http://t.co/XIN1lqhtfb  https://t.co/zj1u0FXXVm
700-637314035524853760  http://t.co/XIN1lqhtfb  https://t.co/zj1u0FXXVm
700-637294604451102720  http://t.co/XIN1lqhtfb  https://t.co/zj1u0FXXVm
700-637352024271536128  http://t.co/K5tfBqSm9G  https:/...
```

Figure 8. Extracted URLs with tabs inserted to the appended ones

Table 5. Tweet cleaning statistics

	count	size	clean_count	clean_size	time
z_541	33869	5.1M	33865	4.3M	5.724s
z_602	92594	14M	90040	11M	7.163s
z_668	14858	2.2M	14832	1.8M	5.255s
z_686	184695	28M	183542	23M	9.288s
z_694	35246	5.1M	31969	3.5M	5.805s
z_700	11290	1.9M	10175	1.2M	4.761s

7.1.2 Webpage cleanup

Cleaning up a webpage is far more challenging because webpages are less formally structured (as compared to tweets). Webpages do come with the standard HTML tags. However, the HTML source also includes external content present in banners, pictures, headers, footers, etc., like in Figure 3. Therefore, it is difficult to identify patterns or properties of text within a web page. We try to clean the webpages following the pipeline in Figure 9.

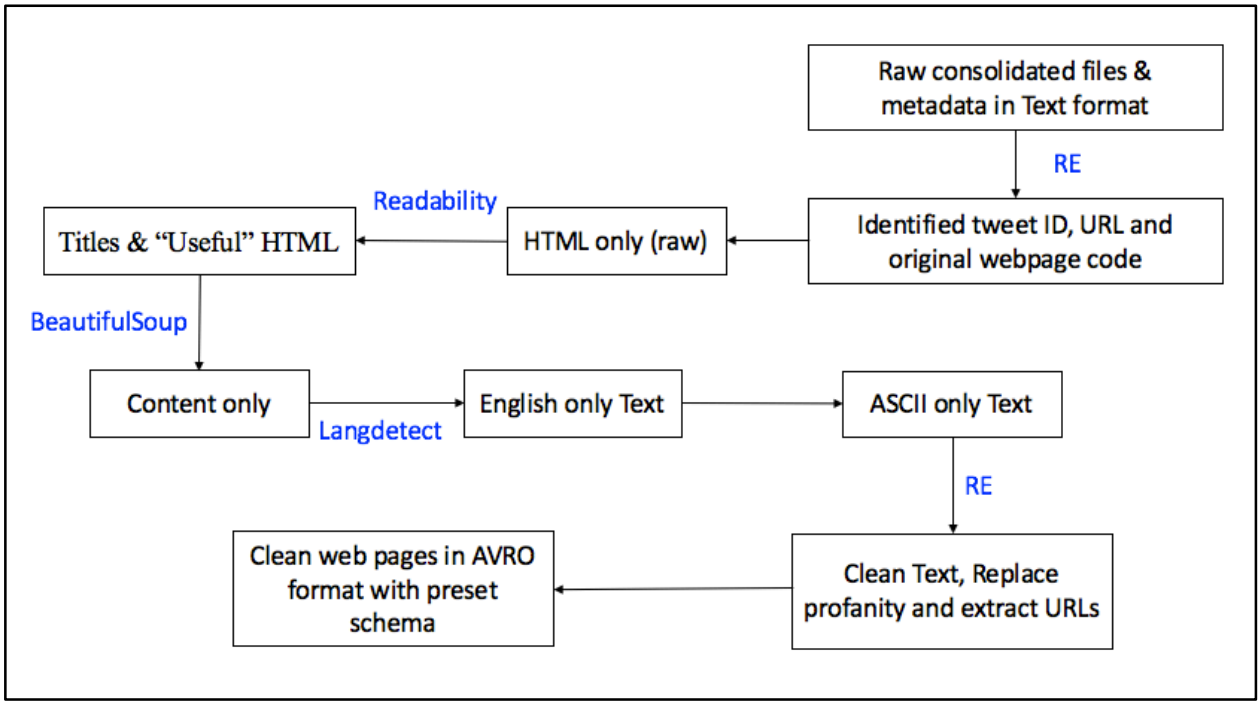

Figure 9. Overview of webpage cleanup implementation

Table 6. Steps and description for cleaning web pages

Step Number	Description
1	Identify tweet ID, URL for webpage and original webpage HTML code by regular expression.
2	Remove noise content in webpages like advertising content, banners, footers by using Python package: python-readability.
3	Cleanup HTML/XML tags, in the web page using package: BeautifulSoup [10].
4	Discard non-English webpages by langdetect package.
5	Remove all remaining non-alphanumeric text.
6	Replace curse or swear words with the exact word "profanity".

7	Extract URLs in the readable content by regular expression.
8	Generate a universally unique identifier (UUID) for each web page.

Like in tweets, cleaned webpages should also be loaded into HBase finally for other teams to consume. Based on the output schema in Table 4, we designed similar columns under the column family 'clean_web'. Details can be found in Table 2. What should be mentioned here is that duplicated URLs among the webpage data have been removed before loaded into HBase. Correspondingly, a new column 'mappings' is added, which tries to capture the relationship between webpage URL and tweet ID. Then we run the Pig scripts in the Appendix to load webpage features into HBase. The example cleaned webpage is shown in Figure 8.

Figure 10. Example of cleaned webpage in HBase

Table 7. Web page cleaning statistics

Collection	Input Size	Output Size
z541	918MB	239MB
z602	4.6GB	641MB
z668	840MB	371MB
z686	7.28GB	3.1GB
z694	2.14GB	817MB
z700	823MB	77MB

Due to the large size of collections, it's hard for us to count the number of records in the input and output manually. We will do statistics automatically in our script in our future work.

7.2 Incremental Update Implementation

7.2.1 Incremental update from relational DB to HDFS

Currently, data are imported from the relational database to HDFS using Sqoop, but the currently bash script used previously for importing more than 700 tables does not have the incremental update feature:

```
#! /bin/bash
# import all tables from archive DB using sqoop
echo "===== import all tables ====="
out=$(sqoop import-all-tables \
        --connect jdbc:mysql://10.0.0.101/twitter \
        --username dlrl \
        --password xxxxxxx \
        --warehouse-dir /collections/tweets-705/ \
        --as-avrodatafile \
        -m 1)
echo "$out"
```

So we need to periodically synchronize the tables' states from the relational database to Hadoop for further processing. The idea is to use --incremental parameter to activate this incremental import feature. Also, since a value in an existing row from a table can be changed after the table is imported into HDFS, we need to use lastmodified mode instead of append mode. The lastmodified mode requires a column storing a data value about when each row was last updated, so that Sqoop will only import rows that were updated after the specific date. The column should be set to the current time with every new inserted row and an update to an existing row, so a row that does not have a modified column will not be imported.

The Sqoop code that we are going to add to the current bash script is like:

```
        …
        --incremental lastmodified \
        --check-column last_update_date \
        --last value "2016-03-31 00:00:00"
        …
```

That will transfer rows whose value in column last_update_date is greater than 2016-03-31 00:00:00. We are still working on the implementation of this feature. We initialized a MySQL RDS instance on AWS and created some tables for testing. After implementation and testing of the feature on these tables with small data, we will apply the script on the relational database that contains real data of tweets and webpages.

In the relational database which contains the raw data, there are several columns including tweet_id, text, time, etc. Part of the schema is shown below in Figure 11. To import the raw data from the relational database to HDFS incrementally, a date of time is needed for the `lastmodified` mode of incremental import in Sqoop. There is one column called "time" in the relational database, and it includes the information about when the tweet is created. Using this information to set a reference of time, we are able to transfer rows whose value in column `time` is greater than a specific date of time.

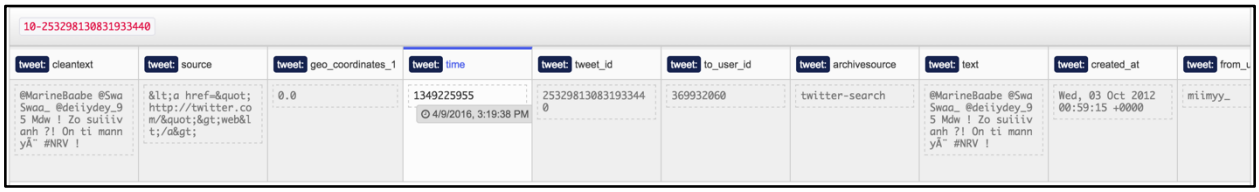

Figure 11. Part of the schema in the relational database which contains raw data

Upon this idea, the bash script can be modified accordingly with incremental import.

```
#! /bin/bash
# incremental import tables from archive DB using Sqoop
echo "=====incremental import tables====="
out=$(sqoop import \
        --connect jdbc:mysql://10.0.0.101/twitter \
        --username dlrl \
        --password xxxxxxx \
        --warehouse-dir /collection/tweets-705 \
        --incremental append \
        --check-column time \
        --last-value "1349225955"
        -m 1)
echo "$out"
```

As shown above, all the records with a time value greater than "1349225955" will be imported from the relational database to HDFS. Currently we are using local MySQL database to test this feature. Once confirmed, the real data from the relational database used by IDEAL will be imported incrementally to HDFS.

7.2.2 Incremental update from HDFS to HBase

After importing data from the relational database to HDFS successfully, we need to further import data to HBase. We wrote Pig script to load the newly imported data to HBase so that the values in HBase columns can get updated.

```
/* Load TSV file */

data = LOAD '/user/cs5604s16_cm/data' USING PigStorage('\t');

/* Store data into HBase */

STORE data 'hbase://ideal-cs5604s16' USING org.apache.pig.backend.hadoop.hbase.HBaseStorage('tweet:cleantext, tweet:source, tweet:geo_coordinat
es_1, tweet:time, tweet:tweet_id, tweet:to_user_id, tweet:archivesource, tweet:text, tweet:created_at, tweet:from_user, tweet:iso_language_code
, tweet:colnum, tweet:geo_coordinates_0, tweet:geo_type, tweet:profile_image_url, tweet:from_user_id');
```

As shown in the code above, first we load the data using Pig storage and then we store the newly imported data to HBase column family, which has the same schema as in the relational database. Thus, the new data can be successfully inserted into HBase from HDFS.

To make the incremental update process automatic, we applied job scheduler *Cron*, which can be run under Linux environment by editing the *crontab* file. By doing this, we are able to run the Pig script periodically, so that every time new data is imported to HDFS, it can be further imported to HBase in a specific period and we don't have to do it manually.

```
0 0 * * 0 pig /home/cs5604s16_cm/load-tsv-into-hbase-incremental.pig
```

Above is the command in the *crontab* file, which runs the Pig script for importing new data on a weekly basis. The first 5 fields specify the time that the command should be executed. In the case above, the Pig script will run once a week at midnight on Sunday morning. The period can be adjusted according to the frequency that new data is imported from the relational database to HDFS. We will talk more about this in the developer manual section.

7.3 Automated web page crawling and cleanup

Right now we are just working with ready-made data, which have been crawled by the GRAs. What if we have new tweet records after incremental update? Therefore, we need to automate the whole progress. For each new tweet record, we will have to extract its URLs and expand them to long ones. Then for such URL list, we will do web page crawling and the above cleanup. Finally, the new cleaned webpage will be loaded back into HBase. You can find the whole automated pipeline in Figure 12.

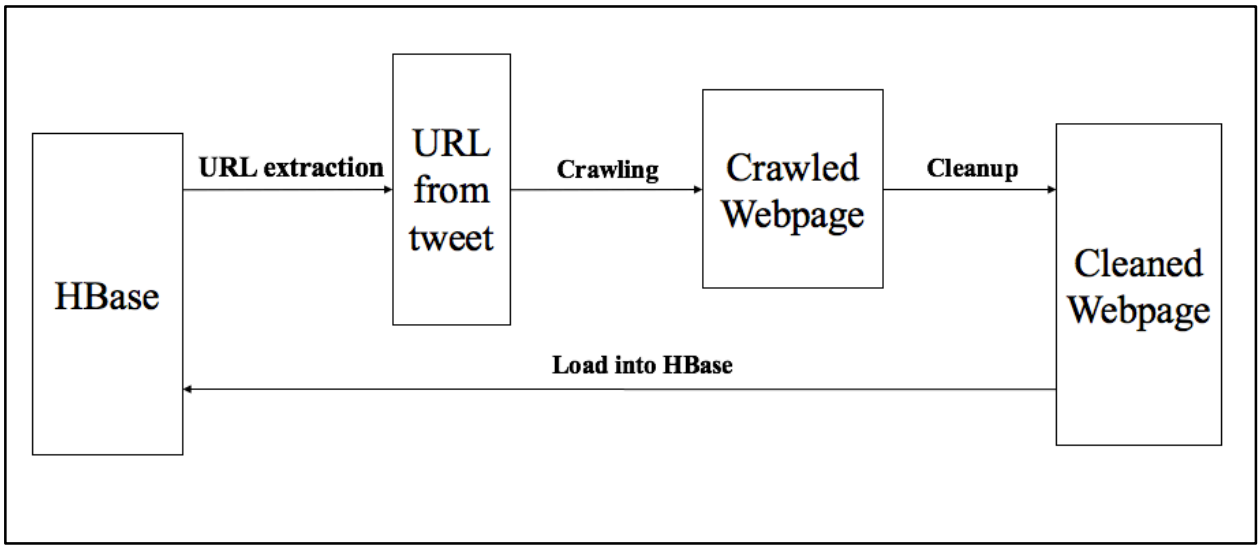

Figure 12. Automated web page crawling and cleanup

We are still working on this part. This will be our next step work.

7.4 Plan and Milestones

Table 8. Milestones and Work division

#	Task	Timeline	Status	Assigned To
1.	Setup Virtual Box and Virtual Machine on own laptops	1/28-2/5	Done	Both
2.	Connected to DLRL cluster and downloaded dataset.tgz and webpage-share	1/28-2/5	Done	Both
3.	Designed Collection Management overall architecture	2/5-2/10	Done	Both
4.	Interim Report 1	2/11-2/12	Done	Both
5.	Cleaning Tweets (Pig)	2/13-3/3	Done	Dong
6.	Cleaning Webpages (Python)	2/13-3/3	Done	Yufeng
7.	Interim Report 2	3/7-3/11	Done	Both
8.	Clean all tweets and webpages from the small collections	3/12-3/31	Done	Both
9.	Load all cleaned tweets and webpages into HBase	3/12-3/31	Done	Both
10.	Interim Report 3	3/29-3/31	Done	Both
11.	Cleaning Webpage (Pig)	4/1-4/15	Done	Yufeng
12.	Incremental update between	4/1-4/15	Done	Dong

	Relational DB and HDFS			
13.	Automated Web page crawling and cleanup	4/1-4/15	Future work	Yufeng
14.	Integrate Named Entity Recognition	4/1-4/15	Borrowed from GRA	GRA
15.	Interim Report 4	4/16-4/20	Done	Both
16.	The final executable code for Incremental Update, Noise Reduction and NER will be released	4/21-5/1	Done	Both
17.	Final project presentation	5/3	Done	Both
18.	Final project report	5/1-5/4	Done	Both

8. User Manual

8.1 Cleaning tweets on local node under HDFS

The code in Pig scripts for tweet cleaning can be found in the Appendix, and you can find guidance on how to use them below.

1. Connect to DLRL Hadoop Cluster and change directory to "cs5604s16_cm/Tweet_Clean".
2. Run Pig scripts to clean the tweets and extract necessary info.

$ **pig –x local** {script name}

Make sure to include "-x local" when you run the command, since the scripts are on a local node instead of HDFS.

3. Find the output files in different directories.

Extracted URLs can be found in "urls" and "urls_uniq". Duplicated URLs are removed and there are unique URLs in "urls_uniq". Hash tags and mentions can be found in "hashtags" and "mentions", and they are ordered by number of occurrence. Cleaned tweets can be found in "clean_tweets".
Currently we cleaned the tweets for the shooting collection – z700, which has been uploaded on **/home/cs5604s16_cm/Tweet_Clean**. If you want more cleaned data from other collections, you can change the path in Pig scripts and run the command as shown above.

8.2 Cleaning webpages on own machine

The Python script for webpage cleaning has been attached in the Appendix. The procedures are listed below:

1. Install Python libraries, including AVRO, readability-lxml, BeautifulSoup and langdetect by "pip install package-name". Save the AVRO schema in Figure 5. to "webpage.avsc", and save the Python code attached to "WebpageClean.py".
2. Run Python script to clean the webpages.

python WebpageClean.py part-00000 cleaned-part-00000 **webpage.avsc** wdbj7 shooting

The above command is of format: **python WebpageClean.py** {input file: part-xxxxx} {output file} {webpage output schema: **webpage.avsc}** {collection name: shooting or disease}

3. Load output into HDFS

Upload cleaned webpages from your own machine to local system on cluster

scp cleaned-part-00000 cs5604s16_cm@hadoop.dlib.vt.edu:/home/cs5604s16_cm/Webpage_Clean

Upload to HDFS from local system

hadoop fs –put webpage-clean/z700Webpages/part-00000 /user/cs5604s16_cm/

At the current stage, only one table of webpages (z700) has been cleaned, and they are available under the folder "/home/cs5604s16_cm/webpage-clean/z700Webpages/". Also we uploaded one cleaned webpage file "part-00000" to HDFS for testing. One can check it with the command "Hadoop fs –ls /user/cs5604s16_cm/", as shown below.

```
cs5604s16_cm@hadoop.dlib.vt.edu's password:
Last login: Thu Mar 10 20:38:45 2016 from 172.30.94.194
[cs5604s16_cm@node1 ~]$ hadoop fs -put webpage-clean/z700Webpages/part-00000 /user/cs5604s16_cm/
[cs5604s16_cm@node1 ~]$ hadoop fs -ls /user/cs5604s16_cm/
Found 5 items
drwx------   - cs5604s16_cm cs5604s16          0 2016-03-03 10:53 /user/cs5604s16_cm/.staging
drwxr-xr-x   - cs5604s16_cm cs5604s16          0 2016-02-28 22:55 /user/cs5604s16_cm/dataset
-rw-r--r--   3 cs5604s16_cm cs5604s16   18357762 2016-02-28 22:46 /user/cs5604s16_cm/dataset.tgz
-rw-r--r--   3 cs5604s16_cm cs5604s16    1047778 2016-03-10 21:59 /user/cs5604s16_cm/part-00000
drwxr-xr-x   - cs5604s16_cm cs5604s16          0 2016-02-09 20:37 /user/cs5604s16_cm/test
[cs5604s16_cm@node1 ~]$
```

Figure 13. Cleaned webpages on HDFS

9. Developer Manual

9.1 Adjusting the period of incremental update

As mentioned in the implementation section, the period to run the Pig script which is responsible for importing data from HDFS to HBase should be adjusted to match the frequency of new data imported from the relational database to HDFS. The developer may want to change this frequency according to the condition of data flow in the system.

```
[cs5604s16_cm@node1 ~]$ crontab -l
0 0 * * 0 pig /home/cs5604s16_cm/load-tsv-to-hbase-incremental.pig
```

The command above can list the current *Cron* task in the system. As shown, the Pig script will be run every Sunday midnight of the week.

```
[cs5604s16_cm@node1 ~]$ crontab -e
```

To change the frequency, one can edit the current *crontab* file by typing in the command above. The first 5 fields specify the details of when to execute the command. The syntax of the file is as follows: field 1 specifies the minute (0 – 59), field 2 is the hour (0 – 23), field 3 represents the day of month (1 – 31), field 4 is the month (1 – 12) and field 5 indicates the day of the week (0 – 6) (0 to 6 are Sunday to Saturday). If a field is left as *, it means "every"; for example, field 4 of the *crontab* command above is *, it means every month of the year.

With the statement above, developers are able to modify the *crontab* file and change the running period according to their need.

9.2 Tweet cleaning and information extraction strategy

Tweet cleaning and information extraction are based on Java regular expression (since functions in Pig Latin use Java regular expression form) and built-in functions provided by Pig Latin.

9.2.1 Java regular expression

Java regular expression is a language of string patterns built in to Java. The common syntax that could be used in tweet cleaning and information extraction is shown as below.

Table 9 Syntax of regular expression

\\d	digit: [0-9]
\\D	non-digit: [^0-9]
\\s	whitespace character
\\S	non-whitespace character
\\w	word character: [a-zA-Z0-9]
\\W	non-word character: [^\\w]
*	match 0 or more times
+	match 1 or more times
?	match 1 or 0 times
.	match any character
\	escape the next meta-character
^	match the beginning of the line
|	'or' statement
()	grouping

[]	custom character class

For example, for the URLs extraction in the raw tweet, the regular expression is "(http://|https://)\\S+". That means all the strings that begin with "http://" or "https://" till before a whitespace character, should be URLs to be extracted.

9.2.2 Built-in functions of Pig Latin

For tweet cleaning and information extraction, we use Pig Latin to process and store the data. Pig provides decent built-in functions for developers to use. The syntax is simple and it can be very efficient to process large amount of data.

For data loading and storing, Pig Latin has "`PigStorage()`" function for developers. Since the raw tweet data on HDFS is in TSV files, it is easy to specify "\t" which means a tab as input in the function as "`PigStorage('\t')`", which will separate tweet id and raw text for further processing.

To do cleaning on the tweet text, the function "`REPLACE`" is very useful. The function "`REPLACE`" replaces the existing characters in a string with new characters, where we can replace things like non-character and profanities with empty string to remove them. After finding the target strings in the text using regular expression, we are able to do "`REPLACE(text, 'regExp', '')`" to implement the cleaning. For example, by doing "`REPLACE(text, '([^a-zA-Z0-9\\s]+)', '')`", we will remove all the non-characters in the text.

For information extraction from the raw tweet, "`REGEX_EXTRACT`" and "`FILTER`" are essential for the work. "`REGEX_EXTRACT`" performs regular expression matching and extracts the matched group defined by an index parameter. For example, if a string named url which has the expression as "http://github.comhttps://www.wikipedia.org", we are able to extract the 2 URLs in it separately by doing "`REGEX_EXTRACT(url, '(http.*)(http\\S+)', 1) AS url1`" and "`REGEX_EXTRACT(url, '(http.*)(http\\S+)', 2) AS url2`" where url1 will be "http://github.com" and url2 will be "https://www.wikipedia.org". "`FILTER`" allows you to select which records will be retained in the data pipeline. Take the hash tags extraction as an example. By doing "`FILTER tweets BY text MATCHES '#\\s*(\\w+)'`", only the hash tags content will be reserved and other content in text will be filtered out.

There are other functions will could also be useful for the tweet cleaning and information extraction such as "`GROUP`", "`JOIN`", "`UNION`", etc. For detailed explanations and usages, developers can refer to this book: Programming Pig [11] by Alan Gates.

9.3 Webpage cleaning

To develop our source code for further webpage cleaning, first we will need to install the required packages like readability-lxml, BeautifulSoup and langdetect. The instructions are as follows.

pip install readability-lxml, beautifulsoup4, langdetect

While to extract the useful information in the webpage record, we will mostly turn to regular expression for help, which has been explained in detail in the tweet cleaning. We show the corresponding regular expressions here.

First to extract the whole webpage record, we need to identify the format of our webpage data as shown in Figure 3. Therefore, the following regular expression can be used, which start with a tweet ID and ends with the HTML code of a webpage.

'\d{1,3}-\d{18}\thttps?://[a-zA-Z0-9\./-]+:URL:<!DOCTYPE.*?</html>'

Then we will want to extract the tweet ID from the webpage record, we can just apply the prefix in the above regular expression. Since the tweet ID is formatted as "collection number – tweet number" followed by the URL for this webpage. We can apply similar regular expressions to extract other information like URLs in the webpage, the original HTML code for this webpage. Details can be found in our attached source code in Appendix.

10. Acknowledgement

This project is supported by the US National Science Foundation through grant IIS -1319578, Integrated Digital Event Archiving and Library (IDEAL). Also we would greatly appreciate the help from instructor Dr. Edward A. Fox, and the two GRAs Sunshin Lee and Mohamed Magdy Gharib Farag, of IDEAL project. We are also grateful for the efforts made by Classification, Clustering and Social Network and Solr teams to communicate with us on the requirements of data schemas needed to implement the whole system. Last but not the least, we thank the whole class for discussing the problems and learning about information storage and retrieval together.

11. References

[1] Qianzhou Du, Xuan Zhang, Named Entity Recognition for IDEAL, Spring 2015. https://vtechworks.lib.vt.edu/bitstream/handle/10919/52254/ReportNER.pdf

[2] Prashant Chandrasekar, Xiangwen Wang, Reducing Noise for IDEAL, Spring 2015. https://vtechworks.lib.vt.edu/bitstream/handle/10919/52340/ReportRN.pdf

[3] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze, Introduction to information retrieval. Vol. 1. Cambridge: Cambridge University Press, 2008.

[4] Nadeau, David, and Satoshi Sekine. "A survey of named entity recognition and classification." *Linguistic Investigations* 30.1 (2007): 3-26.

[5] Zhou, GuoDong, and Jian Su. "Named entity recognition using an HMM-based chunk tagger." *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics(ACL), Philadelphia, July 2002, pp. 473-480.

[6] Ratinov, Lev, and Dan Roth. "Design challenges and misconceptions in named entity recognition." *Proceedings of the Thirteenth Conference on Computational Natural Language Learning(CoNLL)*. Association for Computational Linguistics, Boulder, Colorado, June 2009, pp. 147-155.

[7] NLTK project, NLTK 3.0 documentation. http://www.nltk.org, accessed on 2/10/2016.

[8] Steven Bird, Ewan Klein, and Edward Loper, Natural language processing with Python. O'Reilly Media, 2009.

[9] Ting, Kathleen and Cecho, Jarek. *Apache Sqoop Cookbook*. Sebastopol: O'Reilly Media, Inc., 2013. Print.

[10] Leonard Richardson, Beautiful Soup Documentation. http://www.crummy.com/software/BeautifulSoup/bs4/doc/, accessed on 3/11/2016.

[11] Alan Gates. *Programming Pig*. Sebastopol: O'Reilly Media, Inc., 2011. Print.

Appendix: Source code of cleaning scripts

Pig scripts for tweets cleanup – clean_tweets.pig

```
/*
 * run by "$pig -x local clean_tweets.pig" if file in local node instead of HDFS.
 */

tweets = LOAD '/home/cs5604s16_cm/CS5604S16/small_data' USING PigStorage('\t') AS
(id:CHARARRAY, text:CHARARRAY);

-- Remove URLs
urls_clean = FOREACH tweets GENERATE id, REPLACE(text, '(http://\\S+)', '') AS url_clean;

-- Remove non-characters
nonchars_clean = FOREACH urls_clean GENERATE id, REPLACE(url_clean, '([^a-zA-Z0-9\\s]+)', '')
AS nonchar_clean;

-- Remove profanity words)
profanities_clean = FOREACH nonchars_clean GENERATE id,
        REPLACE(
        REPLACE(
        REPLACE(
        REPLACE(
        REPLACE(
        REPLACE(
        REPLACE(
        REPLACE(
        REPLACE(
        REPLACE(
        REPLACE(
        REPLACE(
        REPLACE(
        REPLACE(
        REPLACE(
        REPLACE(nonchar_clean, 'shit', ''),
        'fuck', '' ),
        'damn', ''),
        'bitch', ''),
        'crap', ''),
        'piss', ''),
        'dick', ''),
        'darn', ''),
        'cock', ''),
        'pussy', ''),
```

```
        'asshole', ''),
        'fag', ''),
        'bastard', ''),
        'slut', ''),
        'douche', ''
        ) AS tweet_clean;

-- Remove empty rows
empty_rows_clean = FILTER profanities_clean BY (id MATCHES '541-.*' OR id MATCHES '602-.*'
OR id MATCHES '668-.*' OR id MATCHES '686-.*' OR id MATCHES '694-.*' OR id MATCHES '700-
.*');
empty_rows_clean = DISTINCT empty_rows_clean;

STORE empty_rows_clean INTO 'cleaned_tweets';
```

Pig script for extracting URLs – extract_urls.pig

```
/*
 * run by "$pig -x local extract_urls.pig" if file in local file system.
 */

-- Data loading
tweets = LOAD '/home/cs5604s16_cm/CS5604S16/small_data' USING PigStorage('\t') AS
(id:CHARARRAY, text:CHARARRAY);

-- Remove invalid rows
tweets = FILTER tweets BY (id MATCHES '541-.*' OR id MATCHES '602-.*' OR id MATCHES '668-
.*' OR id MATCHES '686-.*' OR id MATCHES '694-.*' OR id MATCHES '700-.*');

-- Extract URL using regular expression
tweetwords = FOREACH tweets GENERATE id, FLATTEN( TOKENIZE(text) ) AS word;
urls = FILTER tweetwords BY word MATCHES '(http://|https://)\\S+';

-- Trim unnecessary characters
urls_clean = FOREACH urls GENERATE id, REPLACE(word, '([^a-zA-Z0-9_.#@&:/\\s]+)', '...') AS url;

-- Store clean URLs
STORE urls_clean INTO 'urls';

double_urls = FOREACH urls_clean GENERATE id, REGEX_EXTRACT(url, '(http.*)(http\\S+)', 1) AS
url1, REGEX_EXTRACT(url, '(http.*)(http\\S+)', 2) AS url2;
double_urls = FILTER double_urls BY ($1 IS NOT NULL);
```

```
-- Store double URLs
STORE double_urls INTO 'double_urls';
```

Pig script for extracting hashtag and mentions - extract_hm.pig:

```
-- Data loading
tweets = LOAD '/home/cs5604s16_cm/CS5604S16/small_data' USING PigStorage('\t') AS
(id:CHARARRAY, text:CHARARRAY);

-- Remove invalid rows
tweets = FILTER tweets BY (id MATCHES '541-.*' OR id MATCHES '602-.*' OR id MATCHES '668-
.*' OR id MATCHES '686-.*' OR id MATCHES '694-.*' OR id MATCHES '700-.*');

-- Remove non-characters except for # and @
tweets = FOREACH tweets GENERATE id, REPLACE(text, '([^a-zA-Z0-9#@\\s]+)', '') AS text;

-- First finding hashtags(#)
-- Tokenize each tweet message
tweetwords = FOREACH tweets GENERATE id, FLATTEN( TOKENIZE(text) ) AS word;

-- Search for only hashtags in the tweets
hashtags = FILTER tweetwords BY UPPER(word) MATCHES '#\\s*(\\w+)';

STORE hashtags INTO 'hashtags';

-- Same process as hashtags, finding mentions(@)
mentions = FILTER tweetwords BY UPPER(word) MATCHES '@\\s*(\\w+)';

STORE mentions INTO 'mentions';
```

Python script for webpage cleanup:

```
# 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 urlparse import urlparse
from langdetect import detect_langs
```

```
import uuid
import time

__undefined_collection__ = 'undefined collection'

# Load Profanity word list
with open('profanity_en.txt') as f:
    __profanity_words__ = f.read()[:-1].split('\n')

# Clean extra blank spaces in webpage text
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 ):
    URL_regexp = re.compile( r'https?://[a-zA-Z0-9\./-]+' )

    ProfanityRegexp = re.compile(
        r'(?<=^|(?<=[^a-zA-Z0-9-\.]))(' # Non word character starting
        + '|'.join(__profanity_words__) # Profanity word list
        + r')(?=$|\W)', re.IGNORECASE ) # End with string end or non-word character

    url_list = re.findall( URL_regexp, content )
    for url in url_list:
        content = content.replace( url , '' )

    clean_content_profanity = re.sub( ProfanityRegexp , '{"profanity"}', content )

    clean_content_profanity = cleanblankspaces( clean_content_profanity )

    clean_content = re.sub( r'([^\s\w]|_)+', '', content )
    clean_content = cleanblankspaces( clean_content )

    # Build URL list string split by '|'
    url_list_str = '|'.join(url_list)
    return ( clean_content, clean_content_profanity , url_list_str )

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

```
    return 0

if collection_name == None:
    collection_name = __undefined_collection__

# Load all text from the input file
raw_text_all = InFile.read().decode('utf8')
InFile.close()

schema = avro.schema.parse(SchemaFile.read())
writer = DataFileWriter(OutFile, DatumWriter(), schema)

# Each webpage record with tweet ID
regex_record_with_webpage = re.compile(
    '(?:\d{3}-\d{18}\t)?https?://[a-zA-Z0-9\./-]+(?::URL:|\t)<!DOCTYPE.*?</html>' , re.DOTALL )

# Each raw webpage record
regex_raw_webpage = re.compile( '<!DOCTYPE.*?</html>' , re.DOTALL )

# URL for this webpage
regex_url = re.compile( r'https?://[a-zA-Z0-9\./-]+(?=(?::URL:|\t)<!DOCTYPE)' )

# Tweet_id for this webpage
regex_tid = re.compile( r'\d{3}-\d{18}' )

webpage_records = re.findall( regex_record_with_webpage , raw_text_all )

clean_webpage_count = 0
languageAll = {}

for raw_record in webpage_records:
    # Get the tweet ID for this webpage
    if re.findall( regex_tid, raw_record ):
        tweet_id = re.findall( regex_tid, raw_record )[0]
    else:
        tweet_id = ""

    # Fetch URL for this webpage
    URL = re.findall( regex_url, raw_record )[0].strip()

    # Get raw webpage text
    raw_webpage = re.findall( regex_raw_webpage , raw_record )[0]

    # Get readable texts and title
    readable_article = Document( raw_webpage ).summary()
    readable_title = Document( raw_webpage ).short_title()

    # Replace non-ascii characters with white space in title
```

```
readable_title = ''.join( [ char if ord( char ) < 128 else ' ' for char in readable_title] )

URL = URL.decode( "utf8" )

# Replace multiple space with one space
readable_title = re.sub(' +', ' ', readable_title)

# Get all the readable texts in the webpage
soup = BeautifulSoup( readable_article, 'lxml' )
texts = soup.findAll( text=True )
all_text = ' '.join(texts).strip()

# Detect the language and continue if not English
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

# Delete extra whitespaces
all_text = all_text.replace('\r\n', ' ')
all_text = all_text.replace('\n', ' ')
all_text = all_text.replace('\t', ' ')

# Replace non-ASCII characters with whitespace
all_text = ''.join([ word if ord( word ) < 128 else ' ' for word in all_text])
all_text = re.sub(' +', ' ', all_text)

# Replace profanity words with "profanity" and get URL_list
( clean_content , clean_content_profanity , url_list_str ) = webcontentcleanup( all_text )

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"] = collection_name + '--webpage--' + webpage_id
webpage_json["text_clean"] = clean_content
webpage_json["web_original"] = raw_webpage
webpage_json["lang"] = 'en'
webpage_json["title"] = readable_title
webpage_json["text_clean_profanity"] = clean_content_profanity
webpage_json["collection"] = collection_name
webpage_json["urls"] = url_list_str
```

```
            webpage_json["domain"] = domain
            webpage_json["url"] = URL
            webpage_json["tweet_id"] = tweet_id

            writer.append( webpage_json )
            clean_webpage_count += 1

        SchemaFile.close()
        writer.close()
        print filename1 + ' has been cleaned up'
        print 'Total webpages: %d' % len( webpage_records )
        print 'Cleaned webpages: %d' % clean_webpage_count
        print 'Percentage cleaned: %.3f' % ( 100.0*clean_webpage_count / len( webpage_records ) )
        print 'Language Statitics: ', languageAll
        return 1

    # Main function to take parameters from terminal
    def main(argv):
        try:
            InputFile = argv[1]
            OutputFile = argv[2]
            SchemaFile = argv[3]
        except IndexError:
            print 'Please specify the input webpage filename, output avro filename and avro schema filename'
            return 0
        try:
            doc_collection = argv[4]
        except IndexError:
            doc_collection = __doc_id__

        return avrocleaning( InputFile , OutputFile, SchemaFile, doc_collection )

    if __name__ == '__main__':
        start_time = time.time()
        main(sys.argv)
        print("--- %s seconds ---\n\n\n" % ( time.time() - start_time) )
        sys.exit( 0 )
```

Pig script for loading cleaned webpages into HBase –

```
/* Load TSV file */

/* pig -f filename for HDFS */

/* pig -x local filename for local file system */
```

```
raw = LOAD '/user/cs5604s16_cm/web/' USING PigStorage('\t') AS (
    url,
    domain,
    collection,
    doc_id,
    lang,
    title,
    text_clean,
    text_clean_profanity,
    urls,
    web_original );


/* Store data into HBase */
STORE raw into 'hbase://ideal-tweet-cs5604s16' USING
org.apache.pig.backend.hadoop.hbase.HBaseStorage('
    clean_web:domain,
    clean_web:collection,
    clean_web:doc_id,
    clean_web:lang,
    clean_web:title,
    clean_web:text_clean,
    clean_web:text_clean_profanity,
    clean_web:urls,
    clean_web:web_original,
    ');
```

Pig script for loading both mappings into HBase - load-tsv-into-hbase-cf-tweetmapping.pig / load-tsv-into-hbase-cf-webmapping.pig

```
/* Load TSV file */

raw = LOAD '/user/cs5604s16_cm/tweetmapping/' USING PigStorage('\t') AS (

   tweet_id,

   urls);


STORE raw into 'hbase://ideal-tweet-cs5604s16' USING
org.apache.pig.backend.hadoop.hbase.HBaseStorage('

   clean_tweet:mappings

   ');

/* Load TSV file */

raw = LOAD '/user/cs5604s16_cm/webmapping/' USING PigStorage('\t') AS (

   url,

   tweet_ids );

STORE raw into 'hbase://ideal-tweet-cs5604s16' USING
org.apache.pig.backend.hadoop.hbase.HBaseStorage('

   clean_web:mappings

   ');
```