10-1-2011

Clustering of twitter technology tweets and the impact of stopwords on clusters

Surya Bhagvat

Follow this and additional works at: http://scholarworks.sjsu.edu/etd_projects

Recommended Citation
Clustering of twitter technology tweets and the impact of stopwords on clusters

A Writing Project Report

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By Surya Bhagvat

2011
Clustering of twitter technology tweets and the impact of stopwords on clusters

By
Surya
Bhagvat

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Teng Moh, Department of Computer Science

Dr. Mark Stamp, Department of Computer Science

Dr. Robert Chun, Department of Computer Science
ABSTRACT

Clustering of twitter technology tweets and the impact of stopwords on clusters

Year of 2010 could be termed as the year in which Twitter became completely mainstream. Twitter, which started as a means of communicating with friends, became much more than its beginning. Now Twitter is used by companies to promote their new products, used by movie industry to promote movies. A lot of advertising and branding is now tied to Twitter and most importantly any breaking news that happens, the first place one goes and tries to find is to search it on Twitter. Be it the Mumbai attacks that happened in 2008, or the minor earthquakes that happened in Bay Area in 2010 or the twitter revolution cause of the Iran elections, most of the tech and not so tech savvy viewers were following twitter rather than any main stream news channels. In fact most of the breaking news now comes on Twitter because of the huge number of user base rather than the traditional mainstream media.

The focus of this paper is clustering with the TF-IDF weighted mechanism of daily technology news tweets of prominent bloggers and news sites using Apache Mahout and to evaluate the effects of introducing and removing stop words on the quality of clustering. This project restricts itself to only tweets in the English language.
ACKNOWLEDGEMENTS

My sincere thanks to Dr. Teng Moh for guiding me throughout the entire project and providing suggestions which helped me immensely for completing the project. Also my thanks to committee members Dr. Mark Stamp and Dr. Robert Chun.
Table of Contents

1) Introduction 8
2) Related Work 9
3) Methodology for gathering and scrubbing the data 10
   3a. Gathering the data 11
       Identify Twitter Sources 11
       Programmatically collect tweets 13
       Twitter libraries/SDK 14
       Collecting the Tweets 14
       Challenges with using the Twitter API 15
       Using Amazon Cloud for collecting the Tweets 16
   3b. Scrubbing the data 18
   3c. Preparing the data 19
4) Background on TF-IDF 20
5) Apache Mahout 21
6) Running the Clustering
   6a. Convert the input tweets to a sequence file 22
   6b. Convert the sequence file to a vector 23
   6c. Custom analyzer for stop words 25
   6d. Running the clustering 26
7) Reading the cluster output 29
8) Evaluating the cluster quality 32
9) Conclusion 36
10) References 36
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Overall Process</td>
<td>10</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Process involved till data is gathered</td>
<td>11</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Different phases in gathering the data</td>
<td>11</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Amazon EC2</td>
<td>15</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Amazon cloud infrastructure</td>
<td>17</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Process involved till data is scrubbed</td>
<td>18</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Different phases in scrubbing the data</td>
<td>18</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Preparing the data</td>
<td>20</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Various steps in clustering</td>
<td>21</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Sequence file format</td>
<td>22</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Command to generate the sequence file</td>
<td>23</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Directory structure after generating the vectors</td>
<td>24</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Command to generate the vectors</td>
<td>25</td>
</tr>
<tr>
<td>Figures 14a-14d</td>
<td>Kmeans representation</td>
<td>27-28</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Command to run the kmeans clustering</td>
<td>29</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Command to run the cluster dumper</td>
<td>31</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Database table structure</td>
<td>32</td>
</tr>
</tbody>
</table>
1) Introduction

Twitter initially started as a service where users who were friends or have common interests could share things together. The idea was very similar to group Short Message Service. However, within a year or so after twitter’s launch it became very popular. The popularity of the twitter can be gauged by the fact that even the White House held a town hall meeting on July 6th, 2011 in which users had a chance to ask questions using the hashtag #AskObama[1]. Even our own SJSU has an official twitter account @SJSU [2]. Twitter uses hashtag mechanism as in the case of #AskObama so that users can simply search using the hashtag [6].

The impact of twitter on technology-related news and blogs is also tremendous. For example when it was found recently that CarrierIQ was installing root kit software on the mobile android and ios devices, the technology blogs that received most hits were those with Twitter accounts, specifically @Gizmodo[3], @Techcrunch[4], and @engadget[5]. These technology blogs would naturally post news on their respective blogs, but to entice readers and to reach a wider audience, the first place they posted news would be on their Twitter accounts.

Twitter, when it was first launched was given the moniker microblogging site and the reason being till that point all the blogs were one to two pages long but in the case of twitter that maximum you can post is 140 characters. The challenge would then be to convey the message in the most succinct manner as possible. To give the brief introduction, to tweet(post) something on twitter you create a twitter account on www.twitter.com and then to follow the other twitter users for example @BarackObama, @Gizmodo or @SJSU you simply use the follow button on the twitter website.

Being a technology and open source enthusiast and who uses twitter in a big way on a daily basis, for me the one thing that is of interest is to cluster the tweets to get related but different perspectives on the technology. For example if we take the case of HP touchpad fire sale we may also be interested in the tweets that have similar kind of fire sales going on in the electronics department, if we are following the CarrierIQ root kit fiasco we may be also interested in the tweets that have the security implications.
The related works done in this area and which serve as a reference point for this project include Tweets mining using WIKIPEDIA and impurity cluster measurement by Qing Chen et al.[8], Breaking News Detection and Tracking in Twitter by Swit Phuvipadawat et al.[7] and Using twitter to recommend real-time topical news by Owen Phelan et al.[9]. This project used the related work done by Qing Chen et al. [8] as a baseline and compared the results.

Clustering of the documents usually implies removing the stop words from the documents and then running the clustering algorithms on it. One cannot apply the same concept when it comes to the tweets, cause of the fact that each tweet is limited to 140 characters and applying the full set of English stopwords would lose the essence of the tweet completely resulting in meaningless clusters. Therefore, in this project there are two important things we need to consider, first to choose only the subset of the standard English stop words and second identifying the custom set of stop words applicable to our input set of tweets. The project used the open source Apache Mahout [12] as a technology platform for machine learning algorithms. To evaluate the quality of clustering, we compared the TF-IDF weights with our custom identified stop words.

The project also identified the API limitations with regards to Twitters API [13] and how this project used Amazon Cloud services [14] to overcome it. Also, Apache Mahout being relatively new, some of the features such as calculation of cluster quality and reading of clustered data were not user friendly. The paper discusses the extensions we wrote for reading the cluster data in a much more intuitive manner.

2) Related Work

Mining of Twitter data is relatively complex since tweets are free-flowing text and are limited to 140 characters each [16]. One of the interesting approaches taken by Qing Chen et al. [8] in their paper was to use Wikipedia [15] as the search engine.

Qing Chen et al. proposed to use Wikipedia search by identifying the important words in a tweet, which the authors called, “the trend,” and then do a Wikipedia search based on the trend name [8]. When a search is done with Wikipedia, it could potentially return hundreds of records. The top five to six search records are then selected, and the short snippet in the search result is added to the tweet, which is then used as an input record. We deviated from this approach by using random thousands of tweets, running them against the clustering algorithms and evaluating the TF-IDF weights of the top terms. In addition, we used custom stop words. Swit Phuvipadawat
et al. used the hashtag #breakingnews and collected all the tweets with that hashtag [7]. Once the tweets were collected, they used Stanford Named Entity Recognizer to classify the keywords into proper nouns. Once this step was completed, similar tweets were grouped together with a number and a group label so that every message would belong to a particular group [7].

### 3) Methodology

There were three main phases with respect to data collection: gathering the data, scrubbing the data and preparing the data. Once the data was available then the next two phases would be to apply the clustering algorithms and then evaluate the results. The overview of the process is shown in Figure 1.

![Fig 1. Overall Process](image)

#### 3a) Gathering the data

![Fig 2. Process involved till data is gathered](image)
The data gathering phase can be further divided into three steps: identifying the Twitter sources, programmatically collecting tweets and storing the data into the local disk. The overview of the process is shown in Figure 3.

![Figure 3. Different phases in gathering the data](image)

**Identify Twitter sources**

Data gathering plays a major role in any project especially the one which involves machine learning and this project is no different. For this project, the dataset would be the technology tweets. The first step in this process is to establish the prominent news sources/bloggers from whom we would collect the tweets on a daily basis.

The sources that we chose include popular bloggers who have a large number of followers on twitter, well known news sites, and news blogs, Apple and Microsoft related sites cause of their popularity among consumers and blogs, security related blogs and deals sites. This ensures that we have as much diversification as possible in our tweets. The sources, the grouping and the additional information is provided in the table below:

<table>
<thead>
<tr>
<th>Category</th>
<th>Twitter id</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual</td>
<td>KevinRose</td>
<td>KevinRose was the founder of Digg a popular crowd souring site, he has lot of inside information on the technology and has now started Milk a mobile application development company.</td>
</tr>
<tr>
<td>Individual</td>
<td>Jason</td>
<td>Jason Calicanis is an entrepreneur and blogger and has sold his company weblogs to AOL and is now the founder of Mahalo and the podcast This Week</td>
</tr>
<tr>
<td>Individual</td>
<td>LeoLaporte</td>
<td>LeoLaporte was involved in technology related broadcasting shows right from early 90’s. He currently runs TwiT.tv an online network which broadcasts shows on a daily basis.</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>News site</td>
<td>Wired</td>
<td>Covers technology news and how it impacts the culture, economy and politics.</td>
</tr>
<tr>
<td>News site</td>
<td>Cnet</td>
<td>Needs no introduction, if you want to know all about the technology then Cnet is the place to be.</td>
</tr>
<tr>
<td>News site</td>
<td>NyTimesTech</td>
<td>The technology stories of New York times</td>
</tr>
<tr>
<td>Gadget blog</td>
<td>Engadget</td>
<td>Covers the news about all the latest cool gadgets.</td>
</tr>
<tr>
<td>News blog</td>
<td>ArsTechnica</td>
<td>Covers the news and also publishes reviews, guides on the various technologies.</td>
</tr>
<tr>
<td>Controversial blog</td>
<td>Gizmodo</td>
<td>Controversial website which publishes all about digital culture and gadget news. They were the ones who caused a major furor when they leaked the lost iphone4 story.</td>
</tr>
<tr>
<td>Technology blog</td>
<td>GigaOM</td>
<td>Started by Om Malik, this covers the web 2.0 news as well as the related technology news.</td>
</tr>
<tr>
<td>Startup/Technology blog</td>
<td>Techcrunch</td>
<td>Needs no introduction, started by Michael Arrington this blog essentially covers all about technology startups. This is currently owned by AOL.</td>
</tr>
<tr>
<td>Social media news</td>
<td>Mashable</td>
<td>This site basically covers social media news.</td>
</tr>
<tr>
<td>Technology blog</td>
<td>ReadWriteWeb</td>
<td>Yet another technology blog.</td>
</tr>
<tr>
<td>Apple fan boys</td>
<td>MacWorld</td>
<td>The name itself suggests what it’s all about. Covers anything and everything related to Apple and its products.</td>
</tr>
<tr>
<td>Everything Microsoft</td>
<td>EverythingMS</td>
<td>If Apple has MacWorld then Microsoft has EverythingMS.</td>
</tr>
<tr>
<td>Emerging tech</td>
<td>Techreview</td>
<td>Owned and published by MIT, covers emerging technologies and their impact.</td>
</tr>
<tr>
<td>Gadgets</td>
<td>Techland</td>
<td>Owned by TIME, mainly covers gadgets, about web in general.</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>Tech news from India</td>
<td>tech2eets</td>
<td>They publish tweets about all the technology news happening in India and around the world.</td>
</tr>
<tr>
<td>Nonprofit</td>
<td>TechSoup</td>
<td>Their goal is to help other non-profits and with the technology resources.</td>
</tr>
<tr>
<td>Security related</td>
<td>SecurityNews</td>
<td>Provides all the latest security news regarding hacking, malware, technology and so on.</td>
</tr>
<tr>
<td>Security related</td>
<td>Itsecnews</td>
<td>Very similar so security news.</td>
</tr>
<tr>
<td>Deals</td>
<td>Woot</td>
<td>Very popular one day one deal web site.</td>
</tr>
<tr>
<td>Deals</td>
<td>Fatwalletdeals</td>
<td>Twitter feed of the user contributed fat wallet deals.</td>
</tr>
<tr>
<td>Mobile(ios) app store</td>
<td>148Apps</td>
<td>Reviews all about ios apps, the name comes from the fact that when iphone was initially launched you could only have 148 apps on your iphone.</td>
</tr>
<tr>
<td>Mobile(ios) app store</td>
<td>Appcraver</td>
<td>One more ios app review site.</td>
</tr>
</tbody>
</table>

**Programmatically collect tweets**

We used Twitter API as an asynchronous means of collecting tweets. One uses Twitter API by creating a developer account on their website, then submitting an application. Once approved, one can use Twitter API [13] to collect tweets. The challenge with using any external API is to understand its limitations with regards to the number of request per hour that an application can make. Twitter API has very strict restriction limits of 150 requests per hour for unauthenticated calls and 350 requests per hour for oauth authenticated calls [13].

As we can see the number is very low and once a developer id hits the limit, there is a chance that the application would be blacklisted and you would not be able to make the calls. Twitter provides REST API to make the calls or you can use open source Java libraries for using the Twitter API’s.

**Twitter libraries/SDK**

As mentioned in the Twitter developer documentation site [13], these are the available Twitter libraries for Java

I. Scribe by Pablo Fernandez – an OAuth library

II. Twitter4J by Yusuke Yamamoto – a Twitter API library (Java platform > v1.4.2, Android and GAE ready)
III. Choosing the Twitter library

For the project, the library we choose to use was Twitter4J. The reason that we choose this over the other two libraries is constant updates to the library and having much more robust api mechanism over the other two.

Collecting the Tweets

We need an automated mechanism where the program asynchronously collected Tweets from our chosen prominent bloggers and news sites on a daily basis. We use Twitter4j library and write a program called UserStream.java which collects the tweets on a daily basis. The algorithm for gathering the tweets is as follows

I. Algorithm 1. Collecting Tweets Using the Twitter4J Library
   Instantiate the TwitterStream class

II. Implement the StatusListener class and the following methods

- onStatus – Whenever the change of status occurs this method will get triggered.
- onDeletionNotice – Called upon deletionNotice notices [17]
- onTrackLimitationNotice – This notice will be sent each time a limited stream becomes unlimited. If this number is high and or rapidly increasing, it is an indication that your predicate is too broad, and you should consider a predicate with higher selectivity.[17]
- onScrubGeo – Called upon location deletion messages [17].
- onException – When the Exception object is thrown.

III. Add the listener

IV. Call the filter method of the TwitterStream passing in the id’s of the Twitter sources

Challenges with using the Twitter API

As mentioned before Twitter is very strict with regards to the usage of its API and has very strict restriction limits of 150 requests per hour for unauthenticated calls and 350 requests per hour for oauth authenticated calls. Initial setup of collecting the tweets included running the above Java program UserStream.java from my home computer which asynchronously collected the data.

The program ran without any issues for couple of days and then ran into the Twitter API limit. Once Twitter recognizes your userid and the Oauth credentials along with the IP address as the cause of the rate limit problem, it blocks it immediately. So definitely using the twenty-five
different resources and collecting the Twitter data over one home computer was not going to work. The only alternative then was to use the cloud based services like Rackspace or Amazon and use the virtual machines to collect the Tweets asynchronously on daily basis.

The choice of Amazon versus Rackspace basically boiled down to the cost of using the servers. Rackspace had a starting flat fee of $10.95/month or 1.5 cents/hr [18] compared to Amazon of $0.02 per hour for micro instances[19]. We chose Amazon EC2 for the project.

Using Amazon Cloud for collecting the Tweets

We used Amazon EC2 Cloud’s services [19] to overcome Twitter API’s limitations by using five virtual machines and dividing the Twitter sources into five sources for each virtual machine.

![Fig 4. Amazon EC2 instances](image)

Each of the virtual machine is using its unique twitter account, so for this project five separate Twitter accounts were used and each was running the same copy of UserStream. Twitter API also expects an application to be associated to a twitter account. The following is the account information for each box.

<table>
<thead>
<tr>
<th>Box name</th>
<th>Associated twitter account</th>
<th>Application name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ec2-107-20-94-65.compute-1.amazonaws.com</td>
<td>surya1902</td>
<td>TestApplicationST2</td>
</tr>
<tr>
<td>ec2-50-17-54-102.compute-1.amazonaws.com</td>
<td>Suryab</td>
<td>Tweets Clustering</td>
</tr>
<tr>
<td>ec2-107-20-94-171.compute-1.amazonaws.com</td>
<td>Suryasignup</td>
<td>tweets Clustering 1</td>
</tr>
<tr>
<td>ec2-184-73-120-69.compute-1.amazonaws.com</td>
<td>suryab_sjsu</td>
<td>Clustering of Tweets</td>
</tr>
</tbody>
</table>
The twenty-five twitter sources have been broken into five sources for each individual box and the data is asynchronously collected and it is pulled every week into the local disk as shown in the figure below.

![Amazon cloud infrastructure diagram]

Fig 5. Amazon cloud infrastructure

After the above steps and copying the files over to the local computer on a weekly basis we would have the consolidated tweets for each box the file would look something like this:

@chicpen75 - RT @mashable: The Last Words of Steve Jobs - http://t.co/OEgwovdV8
@mapostolou - RT @mashable: The Last Words of Steve Jobs - http://t.co/OEgwovdV8
@ShebaFamily - RT @mashable: 10 Creepy iPhone Photos for Halloween - http://t.co/mOwN1zFU
@sta2999 - @mashable always tweets right after me. i'm going to start reading all of it.

3b) Scrubbing the data
The data scrubbing phase can be further divided into the following steps as shown in the figure below:

1. Remove retweets
2. Remove shortened URLs
3. Remove error messages
4. Store scrubbed data

Twitter provides the streaming API for near real-time access to the twitter data [13]. For our project we will be using the User streams which provides the information about the user like his direct messages, mentions, followers and so on. We are more interested in getting the direct messages of a particular user or source. The streaming API needs the oauth credentials of the twitter account and we must provide them.

Since we are using the Twitter4j and we are interested in the user resources we need to make use of the twitter4j.StatusListener and twitter4j.TwitterStream classes. When we make use of the status listener we not only obtain the tweets made by a particular source but we also end up having the retweets as part of the stream. In this project we don’t necessarily make use of the retweets, so we filter out the retweets. Also most of the tweets use URL shorteners at the end of
the tweets which actually point to the original link of the article, this would affect the clustering
and as such we are going to strip away the shortened URL’s like `http://bit.ly`. The algorithm for
scrubbing the tweets is as follows

i. Go through each Tweet and if the Tweet begins with `@<source>` filter it out.

ii. Strip out the shortened URL’s from the Tweets.

iii. Write all the filtered tweets to a separate file.

As mentioned above even after breaking the sources into five different sources and having five
Amazon based virtual machines in the cloud we would still run into issues with the API limit and
the streamed source file would end up having all the information related to the connection as
shown in below

```
Stream closed.Relevant discussions can be on the Internet at:
http://www.google.co.jp/search?q=70971d2e or
http://www.google.co.jp/search?q=000687bf
TwitterException{exceptionCode=[70971d2e-000687bf 70971d2e-0006876e], statusCode=-1,
retryAfter=-1, rateLimitStatus=null, featureSpecificRateLimitStatus=null, version=2.2.5-
SNAPSHOT(build: daa2ef4273f09c48cfeddff8317ce12d6f22997b)}
at
twitter4j.AbstractStreamImplementation.handleNextElement(AbstractStreamImplementation.java:167)
at
twitter4j.StatusStreamImpl.next(StatusStreamImpl.java:67)
at
twitter4j.TwitterStreamImpl$TwitterStreamConsumer.run(TwitterStreamImpl.java:443)
Caused by: java.io.IOException: Premature EOF
at
at
at
at
java.io.FilterInputStream.read(FilterInputStream.java:116)
```

We need to clean up our tweets file by removing the irrelevant information and since the error
message is not standardized we do this step manually.

3c) **Preparing the data**
Apache mahout expects each input of the text to be in its own separate file. When we used the Twitter streaming API and scrubbed the data, we ended with one file with all the tweets (retweets removed). We needed to make sure that each tweet inside the file was contained in a file of its own.

i. **Algorithm 2.** Preparing the Input Data
   Take the filtered tweet file as the input.

ii. Read each line and effectively write it to a separate file, the file name begins with ‘Twitter_’ count of the file.

iii. This would effectively serve as the input to the clustering algorithms.

---

4) **Background on TF-IDF**

TF-IDF stands for Term Frequency-Inverse Document Frequency. Term Frequency represents the importance of a term in a specific document whereas Inverse Document Frequency represents the importance of a term relative to the entire corpus. Thus, under TF-IDF the stop words had smaller weight since they appeared frequently in the documents, and the non-stop words or the words that gave the gist of the topic had greater weight. One of the goals of the project was to see the effect of using the correct stop word.

5) **Apache Mahout**

For this project we would be using Apache Mahout which has scalable machine learning libraries including the clustering algorithms [12]. Mahout is very new but has become quite popular as an open source library for mining of large data sets.

Some of the challenges with using a technology as new as Mahout is the lack of documentation. To understand the usage of the libraries going through the source code was the only option. Recently Manning on Oct 14\textsuperscript{th}, 2011 released a book called Mahout in Action which was referenced in the project.
6) Running the clustering

Quite a lot of understanding and prep work needs to be done before running the actual clustering algorithms. The sequence is outlined and shown in the figure below:

i. Convert the input tweets to a sequence file.

ii. Converting the sequence file to a vector.

iii. Run the analyzer for the stop words.

iv. Running the actual clustering algorithm.

Fig 9. Various steps in clustering
6a & 6b) Convert the input tweets to a sequence file and then vector

Converting the input tweets into a vector would be the first step in clustering process. When we are trying to cluster text documents then we essentially represent document as a vector. We can think of a vector as something that contains the words as well as number of times each word occurs in the document. In Apache Mahout we need to first convert the input tweets into sequence file format and then from the sequence file format you create vectors.

There are two steps involved when using Apache Mahout in the generation of the vectors. The first step would be to generate the sequence file and the next would be to generate the vectors passing the sequence file as the input.

Step 1: Generate the Sequence File

First run the following command to generate the sequence file in the chunked manner [20]

```
$MAHOUT_HOME/bin/mahout seqdirectory \--input <PARENT DIR WHERE DOCS ARE LOCATED> --output <OUTPUT DIRECTORY> \--c <CHARSET NAME OF THE INPUT DOCUMENTS> {UTF-8|cp1252|ascii...>} \--chunk <MAX SIZE OF EACH CHUNK in Megabytes> 64 \--prefix <PREFIX TO ADD TO THE DOCUMENT ID>
```

The sequence file can be considered as sort of an intermediary file which essentially has the document id and the content of the text, in our case the tweet. It’s not an ascii text file, so the content shown here is essentially a screen shot of how the sequence file looks like

The screen shot to generate the sequence file is shown below
Fig 11. Command to generate the sequence file

Step 2: Generate the vectors from the Sequence File

When one tries to cluster text documents, one represents each document as a vector. One can think of a vector as an entity that contains words as well as number of times each word occurs in the document. Mahout provides three different classes for vectors namely DenseVector, RandomAccessSparseVector and SequentialAccessSparseVector. From the Java API [21] the following are the users of the three classes

**DenseVector** – Implements the vector as an array of doubles

**RandomAccessSparseVector** – This vector only stores non-zero doubles

**SequentialAccessSparseVector** – Similar to RandomAccessSparseVector wherein it stores non-zero doubles and the elements would be accessed in a sequential fashion.

In our project we are going to use SequentialAccessSparseVector. The class to generate SequentialAccessSparseVector from the sequential file is

```
java org.apache.mahout.vectorizer.SparseVectorsFromSequenceFiles --input <SequenceFile> --output <Output folder> --namedVector --minDF <int> --maxDFPercent <int> --weight TFIDF --analyzerName <CustomAnalyzerClass>
```
The required and important parameters are described below

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>Location of the sequence file.</td>
</tr>
<tr>
<td>output</td>
<td>Location of the output folder where the sparse vectors will be written.</td>
</tr>
<tr>
<td>namedVector</td>
<td>Output vectors should be NamedVectors.</td>
</tr>
<tr>
<td>minDF</td>
<td>The minimum document frequency, the minimum number of documents the term should occur. If it is less than the provided value then it won’t be the part of the dictionary.</td>
</tr>
<tr>
<td>maxDFPercent</td>
<td>The maximum document frequency, if the term occurs in more than the specified value then it is ignored.</td>
</tr>
<tr>
<td>Weight</td>
<td>In this case we use TF-IDF, we can also just use TF.</td>
</tr>
<tr>
<td>analyzerName</td>
<td>This should be a Java class that extends Lucene’s Analyzer where we would define our own custom stop words.</td>
</tr>
</tbody>
</table>

Once the above command is run the following would be the output

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>df-count</td>
<td></td>
<td>File Folder</td>
</tr>
<tr>
<td>tfidf-vectors</td>
<td></td>
<td>File Folder</td>
</tr>
<tr>
<td>tf-vectors</td>
<td></td>
<td>File Folder</td>
</tr>
<tr>
<td>tokenized-documents</td>
<td></td>
<td>File Folder</td>
</tr>
<tr>
<td>wordcount</td>
<td></td>
<td>File Folder</td>
</tr>
<tr>
<td>dictionary.file-0.crc</td>
<td>1 KB</td>
<td>CRC File</td>
</tr>
<tr>
<td>frequency.file-0.crc</td>
<td>1 KB</td>
<td>CRC File</td>
</tr>
<tr>
<td>dictionary.file-0</td>
<td>114 KB</td>
<td>FILE-0 File</td>
</tr>
<tr>
<td>frequency.file-0</td>
<td>115 KB</td>
<td>FILE-0 File</td>
</tr>
</tbody>
</table>

Fig 12. Directory structure after generating the vectors

The dictionary file contains the term to its ID, the tokenized-document folder is where the custom analyzer if specified is applied or the standard analyzer is used and the words are tokenized. The tfidf-vectors folder contains the actual vectors of the documents.

The screen shot to generate the vectors is shown below
6c) Custom analyzer for the stop words

Mahout provides the class org.apache.lucene.analysis.DefaultAnalyzer. This class extends
the org.apache.lucene.analysis.Analyzer class and provides a default constructor without any
arguments. This is the class where we define the stop words and choosing the right set of stop
words would play a crucial role in clustering.

By default Mahout’s DefaultAnalyzer doesn’t include any stop words. The goal was to
compare the weightiness of the top terms in three runs; the first run with no stop words defined,
the second with the chosen English set of stop words from the full set of MySQL stop words [22]
and the final run with the custom stop words that we identified.

The following are the stopwords that are used for the second run i.e., regular English
stopwords that we chose from the full set of MySQL stopwords [22]. Only a subset of the
stopwords from the MySQL stop words was chosen. These stop words were compiled by running
many rounds of the clustering algorithms and then comparing the results. If we include the full set
of MySQL stopwords [22] and compared the results, the clusters that were obtained were
especially meaningless because of the 140 character twitter limit. We would like to compare the
results of using the chosen regular English stopwords versus combining these with the custom
The custom stop words in addition to the above words that are specific to the twitter world are the following. These words were obtained by collecting the tweets on a weekly basis, then running the clustering algorithms, verify whether the top terms that were generated added value to the cluster or not. If the word didn’t really add value to that cluster it would be a stop word. This was a manual effort done on a weekly basis.

The way the regular and custom stop words were identified was by collecting the tweets on a regular basis, running the clustering algorithms on the input, evaluate the impact the stop words have on the weight of the prominent terms and then include or exclude it from the set. This was a manual process which could be improved by means of an algorithmic approach.

6d) Running the clustering

We would be using TF-IDF along with Kmeans clustering to cluster the tweets. The K means algorithm is as follows

i. Assume there are n points and k groups in which you want to cluster these n points.

ii. The first step would be to start with an initial set of k centroid points.

iii. Define the max number of iterations.

iv. The algorithm continues its processing and will keep refining the centroids until it comes to max number of iterations or the centroids converge.
An example follows (taken from Mahout in Action book)

Assume initially we have three centroids c1, c2 and c3

(Fig 14a)

The map stage assigns each point to the cluster nearest to it

(Fig 14b)

In the reduce stage, the associated points are averaged out. This would result in the centroid location shifting as shown in the figure and produces new centroids.

(Fig 14c)

Once the iteration completes, this configuration from the above figure becomes the starting point for the loop and continues till the centroids converge to their final positions
Selecting the distance measure

From distance measures, such as CosineDistanceMeasure, EuclideanDistanceMeasure and TanimotoDistanceMeasure, we chose CosineDistanceMeasure after doing some trail runs.

Command to run clustering

Mahout provides the class `org.apache.mahout.clustering.kmeans.KmeansDriver` to run the Kmeans clustering. The class takes the following parameters

```java org.apache.mahout.clustering.kmeans.KMeansDriver -i <input location> -c <initial clusters location> -o <output location> -cd <distance> -x <iterations> -k <Number of clusters> -cl -dm <Distance measure>
```

The required and important parameters are described below

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Path to the input directory.</td>
</tr>
<tr>
<td>c</td>
<td>Path to the initial clusters directory.</td>
</tr>
<tr>
<td>o</td>
<td>Output directory name.</td>
</tr>
<tr>
<td>cd</td>
<td>The convergence delta value.</td>
</tr>
<tr>
<td>x</td>
<td>The maximum number of iterations.</td>
</tr>
<tr>
<td>k</td>
<td>Number of clusters.</td>
</tr>
<tr>
<td>cl</td>
<td>Indicates that the clustering needs to be run after the iterations have taken place.</td>
</tr>
<tr>
<td>dm</td>
<td>The distance measure that needs to be used, in our case we use CosineDistanceMeasure.</td>
</tr>
</tbody>
</table>
The screen shot to run the kmeans clustering is shown below

![Command to run the Kmeans clustering](image)

After running the Kmeans we get the following directory structure

```
clustersPoints
    clusters-1
    clusters-2
    clusters-3-final
```

This would vary depending on each run, in our case we choose the number of clusters to be 50 and the number of iterations before the clusters converge was set as 20, the clusters may converge before as is the case shown above.

### 7) Reading the cluster output

Mahout provides a utility called `org.apache.mahout.utils.clustering.ClusterDumper` which provides all the clusters information. The way to run this program is

```
java org.apache.mahout.utils.clustering.ClusterDumper -s <Sequence files directory for the clusters> -p <Directory containing point sequence files> -n <Number of top terms that we want to print> -d <Location of the dictionary file> -dt <The dictionary file type> -o <Output directory>
```

The required and important parameters are described below.
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>This is the sequence file directory which is generated after running the Kmeans algorithm.</td>
</tr>
<tr>
<td>p</td>
<td>All the clustered points are stored in clusteredPoints directory, this would point the location of the directory.</td>
</tr>
<tr>
<td>n</td>
<td>The number of top terms (the terms with the highest weights that we want to print), in our case we are printing the top five terms.</td>
</tr>
<tr>
<td>d</td>
<td>This is the location of the dictionary file that was generated with the SequentialAccessSparseVector.</td>
</tr>
<tr>
<td>dt</td>
<td>The type of the dictionary file type whether it’s sequential or text.</td>
</tr>
<tr>
<td>o</td>
<td>The output directory.</td>
</tr>
</tbody>
</table>

Once we run the above program we would get the following file in the output directory clusteroutput. If we look at the content of the clusteroutput file it’s one huge file with all the clusters listed along with the weight for each term and the weights for the top five terms. A sample output is shown below (note this is just a sample, this file has lots of data)

As we can see the cluster dumper output nicely prints out the top terms in each cluster and their weights along with all the files that form part of the cluster.

The screenshot to run the ClusterDumper utility is shown below
Running the above program results in one file that contains all the cluster information and which is very difficult to read. Also, it is very difficult to associate the input tweet files in the cluster with the actual contents. To facilitate the viewing of the data in the clusters and to print the contents of the input tweets, we wrote a custom java program that employed the algorithm described below.

**Algorithm 3. Algorithm for Better Viewing of Clustering Data**

I. Opens the output file generated from ClusterDumper.
II. Reads each line of the file and looks for the following strings CL- and VL-, this basically indicates the start of the cluster.
III. From the position to the next CL- or VL- it reads each line, the following would be the lines Top Terms:
    - iphone => 0.1507526587055726
    - apple => 0.12412147780659481
    - mobile => 0.11814728919291656
    - jobs => 0.1115138864090512
    - samsung => 0.1009606923032927
IV. For each cluster we create a new file and first write the top terms.
The next line in the file generated from Mahout utility would be something along these lines as an example:

1.0 : [distance=0.9258744922524543]: /Tweet_1018.txt = [deutsche:7.992, mango:5.227, monday:7.673, sets:7.673, telekom:8.125]

The program parses this line, figures out what the file is in this case Tweet_1018.txt, then opens this file and writes the contents of it.

The end result would be a file for each cluster with the top terms and the actual content of the Tweet included as shown below.

| Top Terms: |
| changes => 4.999809711210189 |
| facebook => 2.0083499416228263 |
| reveal => 1.546768403822376 |
| consume => 1.0918245623188634 |
| mean => 0.7425660933217695 |

Tweet_1000.txt RT @mashable: COMIC: Facebook Changes Getting Out of Hand - Tweet_1039.txt Facebook changes getting to you? This #comic shows you how it could be worse Tweet_105.txt Sears Ultra Plus Powdered Laundry Detergent 1/2 price $11.99 Tweet_1082.txt Facebook Changes in a Nutshell [COMIC]

8) Evaluating the cluster quality
To evaluate the quality of the cluster, we basically take the approach of calculating the TF-IDF weights for prominent terms with the following analyzers:

- TweetsAnalyzer with no stop words.
- TweetsAnalyzer with common English stop words.
- TweetsAnalyzer with common English stop words + custom stop words.

Mahout doesn’t really provide any mechanism to store the top terms in any database, we extend Mahout and provide a custom program that essentially writes the top terms to the database for each run.

The database that we use for this purpose is MySQL and the database table that we have created for this purpose is named TermsAndWeights and the structure is in the following fashion.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Collation</th>
<th>Attributes</th>
<th>Null</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>RunType</td>
<td>varchar(200)</td>
<td>latin1_swedish_ci</td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Term</td>
<td>varchar(50)</td>
<td>latin1_swedish_ci</td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weight</td>
<td>varchar(50)</td>
<td>latin1_swedish_ci</td>
<td>No</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field_1</td>
<td>varchar(50)</td>
<td>latin1_swedish_ci</td>
<td>Yes</td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field_2</td>
<td>varchar(50)</td>
<td>latin1_swedish_ci</td>
<td>Yes</td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Fig 16)

RunType – Basically describes the run type of the whole process, this could be one of NoStopWords, DefaultStopWords or CustomStopWords.
Term – The actual term.

Weight – The value of the assigned weight by that particular RunType.

Field_1 – Additional field.

Field_2 – Additional field.

**Algorithm 4.** Algorithm to track the top terms and insert into the database

The algorithm to track the top terms and insert into the database is given below:

1. Opens the output file generated from ClusterDumper.
2. Reads each line of the file and looks for the following strings CL- and VL-, this basically indicates the start of the cluster.
3. Connect to the database.
4. From the position to the next CL- or VL- it reads each line, the following would be the lines:
   - Top Terms:
     - iphone => 0.15075426587055726
     - apple => 0.12412147780659481
     - mobile => 0.11814728919291656
     - jobs => 0.1115138864090512
     - samsung => 0.1009606923032927
5. Read each top term and insert into the table along with the run type.

We run the program with different analyzers:
- TweetsAnalyzer with no stop words.
- TweetsAnalyzer with common English stop words.
- TweetsAnalyzer with common English stop words + custom stop words.

We then look at the top terms and their weights for each run.

The results of the weights of the following five terms, siri, apple, iphone, augmented and with, for different runs, are shown in the tables below. These terms were chosen randomly:

**siri**

<table>
<thead>
<tr>
<th>RunType</th>
<th>Term</th>
<th>Weight</th>
<th>Field_1</th>
<th>Field_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>DefaultStopWords</td>
<td>siri</td>
<td>1.5073394045884874</td>
<td>CLUSTER_22</td>
<td>NULL</td>
</tr>
<tr>
<td>CustomStopWords</td>
<td>siri</td>
<td>3.51806756553362943</td>
<td>CLUSTER_16</td>
<td>NULL</td>
</tr>
<tr>
<td>CustomStopWords</td>
<td>siri</td>
<td>0.608281407211786</td>
<td>CLUSTER_37</td>
<td>NULL</td>
</tr>
</tbody>
</table>

We notice for **siri** that when we didn’t include any stop words it was not even part of the cluster but for default and custom stop words it carried the weights as shown above and for custom it appeared even twice.
For the term **apple** the weight is more when we have used the CustomStopWords when compared to no stop words and default stop words

<table>
<thead>
<tr>
<th>RunType</th>
<th>Term</th>
<th>Weight</th>
<th>Field_1</th>
<th>Field_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoStopWords</td>
<td>apple</td>
<td>0.53</td>
<td>CLUSTER_39</td>
<td>NULL</td>
</tr>
<tr>
<td>DefaultStopWords</td>
<td>apple</td>
<td>0.68</td>
<td>CLUSTER_20</td>
<td>NULL</td>
</tr>
<tr>
<td>CustomStopWords</td>
<td>apple</td>
<td>1.98</td>
<td>CLUSTER_22</td>
<td>NULL</td>
</tr>
<tr>
<td>CustomStopWords</td>
<td>apple</td>
<td>4.62</td>
<td>CLUSTER_37</td>
<td>NULL</td>
</tr>
</tbody>
</table>

**iphone**

<table>
<thead>
<tr>
<th>RunType</th>
<th>Term</th>
<th>Weight</th>
<th>Field_1</th>
<th>Field_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoStopWords</td>
<td>iphone</td>
<td>2.17</td>
<td>CLUSTER_38</td>
<td>NULL</td>
</tr>
<tr>
<td>DefaultStopWords</td>
<td>iphone</td>
<td>4.16</td>
<td>CLUSTER_20</td>
<td>NULL</td>
</tr>
<tr>
<td>CustomStopWords</td>
<td>iphone</td>
<td>3.07</td>
<td>CLUSTER_12</td>
<td>NULL</td>
</tr>
<tr>
<td>CustomStopWords</td>
<td>iphone</td>
<td>0.45</td>
<td>CLUSTER_16</td>
<td>NULL</td>
</tr>
<tr>
<td>CustomStopWords</td>
<td>iphone</td>
<td>0.80</td>
<td>CLUSTER_37</td>
<td>NULL</td>
</tr>
</tbody>
</table>

**iphone** is relatively common term in the tech industry and as such for custom the value is little bit less than that of default though it appeared in more times than default.

**augmented**

We see the term like augmented appears only when we have the custom stop words and if we look at the cluster to which it is assigned its relevance is immediately apparent

<table>
<thead>
<tr>
<th>RunType</th>
<th>Term</th>
<th>Weight</th>
<th>Field_1</th>
<th>Field_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustomStopWords</td>
<td>augmented</td>
<td>1.52</td>
<td>CLUSTER_36</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Tweet_11533.txt Scientists testing HUD contact lenses on rabbits, hope to bring **augmented** reality to your eyeballs

Tweet_1267.txt Lego's augmented reality at IDF, eyes-on (video)

Tweet_1269.txt 'What Was Here' project adds a pinch of history to augmented reality

Tweet_12749.txt Playing with Augmented Reality

**with**

We see that the meaningless terms like with which doesn’t add much value show up only for NoStopWords
The consolidated table is shown below for the following five terms siri, apple, iphone, augmented and with

<table>
<thead>
<tr>
<th>Term</th>
<th>No stop words</th>
<th>Default stop words</th>
<th>Custom stop words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Siri</td>
<td>Not part of cluster</td>
<td>1.5073364045884874</td>
<td>3.5180675863362043</td>
</tr>
<tr>
<td>apple</td>
<td>0.5385988283673778</td>
<td>0.6785155087709427</td>
<td>4.620405804027211</td>
</tr>
<tr>
<td>iphone</td>
<td>2.178985970975669</td>
<td>4.1664243917912245</td>
<td>3.071035173501861</td>
</tr>
<tr>
<td>augmented</td>
<td>Not part of cluster</td>
<td>Not part of cluster</td>
<td>1.5207760339464461</td>
</tr>
<tr>
<td>With</td>
<td>0.8422637692204228</td>
<td>Not part of cluster</td>
<td>Not part of cluster</td>
</tr>
</tbody>
</table>

Clearly, the important terms have more weightiness when we chose custom stop words compared to us using no and default stop words. Hence, choosing the correct set of stop words for microblogging sites such as Twitter is very important to get good clusters.

The second experiment that was conducted was to first collect sample of thousand random tweets and then use the approach taken by Qing Chen et al.[8] in their paper “Tweets mining using WIKIPEDIA and impurity cluster measurement”. As outlined in the paper we use wikipedia search by identifying the important words in the tweet which the authors call the trend and then do a wikipedia search based on the trend name. We combine the original text with the wikipedia text and use it as an input. Please note that since the tweets are collected on a daily basis there may not be any Wikipedia pages associated with the top terms. In this case we have used the original tweet as the input.

The results are shown for the top ten terms when using the above approach with default stop words versus the approach we took of using custom stop words and is shown in the table below.

The second column shows the weights of the top terms under the approach that we took. The third column shows the weights of the terms under the approach that Qing Chen et al. took [8]. We noticed that terms had the highest weight when we chose the right set of stop words compared to combining the tweets with Wikipedia search results, showing that choosing the right set of stop words plays a major role in getting good clusters.
<table>
<thead>
<tr>
<th>Term</th>
<th>Custom stop words</th>
<th>Wikipedia (Default stop words)</th>
</tr>
</thead>
<tbody>
<tr>
<td>hadoop</td>
<td>5.9972124099731445</td>
<td>3.5919069877037635</td>
</tr>
<tr>
<td>internet</td>
<td>4.997677008310954</td>
<td>1.4604256780524003</td>
</tr>
<tr>
<td>wired</td>
<td>4.031572866439819</td>
<td>2.644033670425415</td>
</tr>
<tr>
<td>nokia</td>
<td>3.9821564934470435</td>
<td>4.583253523882697</td>
</tr>
<tr>
<td>free</td>
<td>3.9537076155344644</td>
<td>2.5828651428222655</td>
</tr>
<tr>
<td>storage</td>
<td>3.895374298095703</td>
<td>2.7575849056243897</td>
</tr>
<tr>
<td>interface</td>
<td>3.8416065216064452</td>
<td>1.6496093273162842</td>
</tr>
<tr>
<td>engadget</td>
<td>3.836599276328087</td>
<td>2.8410439314665616</td>
</tr>
<tr>
<td>drop</td>
<td>3.7278315226236978</td>
<td>1.9779029173009537</td>
</tr>
<tr>
<td>million</td>
<td>3.5409763560575596</td>
<td>N/A</td>
</tr>
</tbody>
</table>

9) Conclusion

One of the main challenges in clustering Twitter data is that the length of each tweet is only 140 characters, and so, identifying the right set of custom stop words is very important.

This project was challenging because of the Twitter API limitations, evaluating and understanding cloud technologies such as Rackspace and Amazon and last but not least trying to understand Apache Mahout, which is a relatively new technology.

The work by no means is complete, one of the ideas that we can continue working on would be to use stop words and combine the twitter text with google news and then cluster the data. One other idea could be to see whether we can give a higher weightage to the tweet in the cluster based on how much the users are sharing it on social networking sites like Facebook and google plus.

10) References


doi: 10.1109/ISI.2010.5484758

http://doi.acm.org/10.1145/1639714.1639794

[10] Twitter API: Up and Running
Learn How to Build Applications with the Twitter API
By Kevin Makice
Publisher:O'Reilly Media
Released: March 2009

By Sean Owen, Robin Anil, Ted Dunning, Ellen Friedman
Publisher:Manning Publications
Released: October 14, 2011


