**Disclaimer:** Submitted to Dublin City University, School of Computing for module CA682: Data management and visualisation, 2014/2015. I hereby certify that the work presented and the material contained herein is my own except where explicitly stated references to other material are made.

CA682 Visualisation Report

*New York Times News Articles*

M**arcelo Grossi** ([marcelo.grossi2@mail.dcu.ie](mailto:marcelo.grossi2@mail.dcu.ie))

Student ID: 14211576

Introduction

The system has the objective of visualizing the most used words and word usage history, broken down into categories, of news articles posted to the New York Times website [www.nytimes.com]. The project aims to allow the user to visualise not only the most used words in a period of time and set of categories but also to view the seasonality of each individual word, also broken down into categories. The visualisation is used mostly as a tool to let the user choose the parameters he wants and analyse the data at his convenience, for this reason it is totally interactive and show not only one, but many charts; each with its own task of showing a piece of the data set.

Data gathering

All of the data was obtained by scrapping the news website of the New York Times, a world reaching American newspaper. The scrapping application was developed using Python script language [https://www.python.org/] and Scrapy [http://scrapy.org/], a very powerful web spider framework.

A much customized approach was used to traverse the collection of news articles available. The regular web crawling link finder technique (process web page 🡪 gather links from the web page 🡪 follow the links and repeat) did not yield very good results. It would stop after reaching around 15,000 news articles before needing human interaction to provide the crawler with another seed URL.

Fortunately, the New York Times website provides a very useful REST API for searching (used only within the website itself). By tapping into this API it was possible to assign a date range and a page number (the site automatically breaks up the response into 10 articles per page) and receive links to all the news articles in the date interval.

|  |
| --- |
| *http://query.nytimes.com/svc/cse/v2pp/sitesearch.json?sort\_order=a&date\_range\_upper={0}&date\_range\_lower={1}&page={2}* |

The web crawler needed only to parse two different kinds of pages; the search API results (that were already in JSON format) and the article itself. With this approach and by running several crawlers in parallel each responsible for one month’s interval, all the news articles from the year 2013 and 2014 (up to 19th of October – the date the web crawler was finalized) were scrapped totalling 148,120 news articles – what amounted to over 4Gb of raw textual data. They were categorized based on the information contained in the web article’s metadata extracted directly from the website. Each article was stored in one individual JSON file. The file format is of the form:

|  |
| --- |
| *{*  *"author": <author>,*  *"category": <category name>,*  *"date": <article date>,*  *"description": <news article short description>,*  *"text": <full text body of article>,*  *"textType": <type of article>,*  *"title": <news article title>,*  *"url": <news article full url>*  *}* |

Cleaning and initial processing

The objective of this stage was to clean the raw textual data of irrelevant terms (stop words), numerical only text, punctuation and any encoding tag that might exist (html tags mainly). For this a script written in Python was produced to output CSV files that would be loaded directly into a database. A very useful stop words library was used to avoid listing all possible stop words in the English language – Natural Language Toolkit [http://www.nltk.org/].

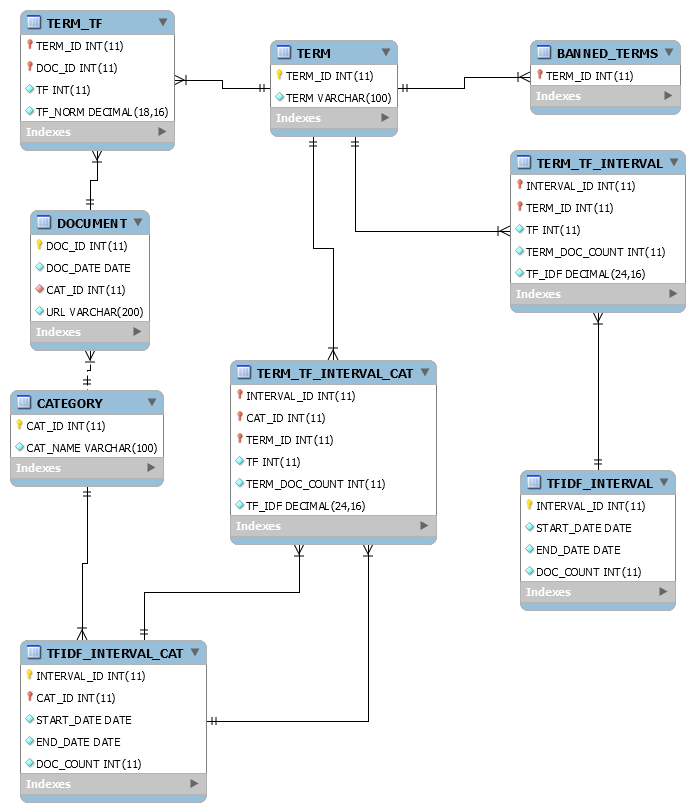
For each news article a list of words and word counts was also produced to support the TFIDF calculations. This file contained the word, the count of that word and the normalized count (the word count divided by the total words in the document; however, this information ended up not being used). The output of this data staging phase yielded four CSV files.

1. Category.csv: Grouping metadata extracted from the article itself. Total of 56 unique categories (less than 1Kb file size).
2. Documents.csv: High level information about the news article containing, the category to which it belonged, the date it was produced and the original URL. Total of 148,120 unique documents (17Mb file size).
3. Terms.csv: List of words. Total of 329,054 unique words (6Mb file size).
4. Termdoc\_TF.csv: List of words per document with count and normalized count. Total of 38,271,909 lines (1.34 GB file size).

Storing data

The CSV files were directly loaded into tables in a MySQL database. This was done due to very inefficient single line direct inserts. The command “load data local infile <filename> into table <table>” was used and data could be loaded in a timely manner.

Trying to calculate the TFIDF ranking for a given time period (the time variable used was the news article date) in this database had very poor performance. To calculate the 250 most important words via TFIDF took several minutes to calculate and thus could not be used in the final visualization. For optimizing the calculation, several support tables were created – by pre-processing the original tables that were initially loaded via the CSV files. Also a table was created especially to store irrelevant terms (after visually analysing the top used words, it was clear that a mechanism to filter out words such as “mr”, “ms”, “said”, amongst others was necessary.



The web server

In order to query the information from the database and feed it into the visualization a web server was constructed by using Python and Flask [http://flask.pocoo.org/]. A REST API was developed to support all the users’ requests for data in an asynchronous manner (AJAX requests on the web browser side). The API definition and output JSON file format can be found below.

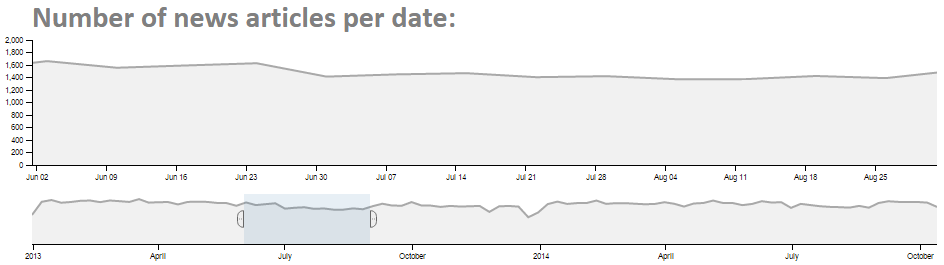
|  |  |  |
| --- | --- | --- |
| API (<server>/api/v1.0/) | Input parameters | JSON output |
| dates | <none> | { min\_date: <value>, max\_date: <value> } |
| CountPerDate | Start Date  End Date | [{ date: <value>, count: <value> }, {}, ..] |
| CountPerCategory | Start Date  End Date | [{ category: <value>, count: <value>}, ..] |
| CategoryCountPerTerm | Term  Start Date  End Date | [{ name: <value>, counts: [{date: <value>, count: <value>}, ..], ..] |
| TopWords | Start Date  End Date  Categories (optional) | [{ term: <value>, tf: <value>, num\_docs: <value>, tot\_docs: <value>, tf\_idf: <value> }, ..] |

The visualisation

The visualisation can be divided into three parts.

The time filter

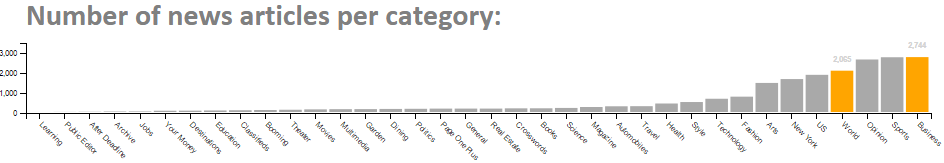
This section is comprised of two line charts stacked on top of each other, showing the number of news articles per date. The bottom chart shows the entire data interval available and is used to select the time interval and the top one represents only the selected time interval (like a zoom functionality). The D3.js brush component was used to perform the data selecting and a clip area was used on the top chart to hide the area of the chart that was not in the interval selected by the bottom chart. This way, the chart needed to be built only once.



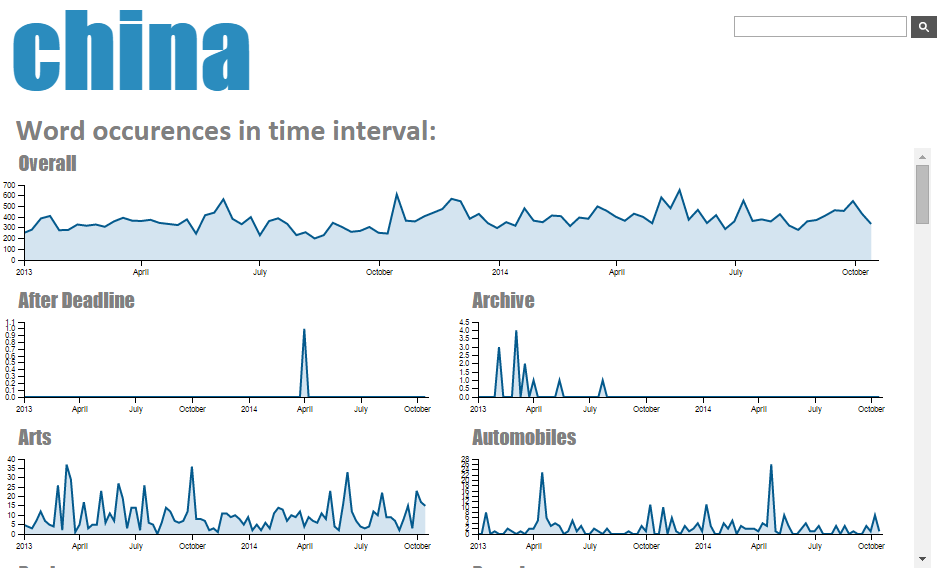
The world cloud and category filter

This section shows the top used words for the selected interval in a World Cloud visualisation (bigger font size equals more used word), based on the cloud layout by Jason Davies [www.jasondavies.com/wordcloud].

A category filter was also introduced to allow for filtering on one or many categories of news articles as well as display information of the number of news articles per category on the selected time interval.

The single word usage history

This section shows the overall usage of a word over time (number of appearances per date) and also the usage over time per each category (number of appearances on the particular category per date) in several line charts stacked on top of each other, with the overall chart always coming first. The word can be selected by clicking on a word from the word cloud chart or by searching directly on the search box.



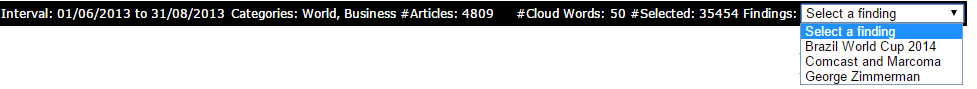
Pre-attentive features were used throughout the visualisation. All charts were displayed in grey tones as to not stand out from the rest. When the user hovered the mouse over an element that could be interacted upon (like categories and words in the word cloud) the element would turn bright orange to make it very easy for the user to see it.

The word cloud’s colours came from a six colour palette obtained from the Colour Brewer website [http://colorbrewer2.org/] with colour blindness safety in mind. The colour brightness of the word was inversely proportional with its relevance (font size) in the world cloud visualisation.

Pre-set findings

Three pre-set findings were also put available on the visualization in order to quickly show some examples of interesting information that this visualization can uncover.

1. Brazil world cup 2014: Even though soccer is not a popular sport in the USA the Brazilian world cup 2014 was a very hot topic.
2. Comcast and Marcoma: This was found by accident while playing with the visualization. Comcast was said to have been hacked in February 2014 and Marcoma was sentenced to nine years in prison for insider trading (this news made headlines because the actual trade amounted to over 200 million dollars in profit).
3. George Zimmerman: On February 2012, Zimmerman fatally shot 17-year-old African American high school student Trayvon Martin and was later acquitted of all crimes in 2013.



Challenges faced

There were many challenges faced during this assignment. The most notable was tweaking the MySQL database to perform in an acceptable speed, where the InnoDB engine’s buffer size increase was the most significant improvement to the system’s query response times. By setting this variable to 4 GB the database was able to perform the queries entirely in memory.

Python was also a big challenge, as I never had used it before and had to learn it from scratch. But this can be said of having been a sweet challenge as the scripting language was very easy to pick up and the speed in which usable programs could be fashioned were impressive. This is a programming language that I will certainly keep using going forward.

JavaScript was also a big challenge. I thought I already knew it before starting this assignment, but found out that my knowledge was very basic and a more detailed understanding was necessary. I cannot claim to be an expert in JavaScript now but I am certainly much better than two months ago.

Conclusion

This was one of the most enjoyable assignments I have ever made. From the thrills of scrapping websites to the breath taking emotion of seeing my very first world cloud pop up, passing by several hours of hair pulling trying to get MySQL to respond faster.

My work could be improved in the future by introducing word stemming before generating the database and also to maybe partitioning the database tables for increased speed. Some functionality of comparing documents could also be introduced for an even richer visualisation possibilities.