TrumpFacts

Eric Norgren ericnor@kth.se, Johannes Olsson johanneo@kth.se,
Anna Veselova veselova@kth.se, Henrik Rönnholm henron@kth.se.

May 18, 2017

Abstract This work describes TrumpFacts, a search-based application with Elasticsearch as its backbone that allows users to generate wordclouds from U.S. President Donald Trump's Twitter feed. The purpose of TrumpFacts is to serve as a tool for analyzing how the words that seem important to Trump change over time. The results show that TrumpFacts allows for some interesting observations regarding Trump's changing political dispositions, but concludes that wordclouds are an imprecise tool better suited for leisurely use, with deeper analytics requiring more sophisticated visualization methods.

Contents

1 Introduction

With current advancements in technology, the world is faced with previously unseen and exponentially increasing volumes of digital information. Identified as two major contributors are social-mobile-cloud and Internet of Things [1], and IBM estimates that 2.5 quintillion bytes of data is created every day - so much that 90% of the world's data was generated in the past two years alone [2].

As most data today is semi- or unstructured [1], the field of information retrieval (IR) is passing a turning point as many enterprise applications leave behind a 30-year tradition of being database-centered to become search engine-based and utilize data which is diverse in content, structure and origin [3]. In this shift, the frontier between databases and search engines is blurred: modern database systems such as NoSQL databases allow for much greater flexibility, while search engines enable data analysis very similar to that of database applications without having to compromise on scalability [4]. In this space, with the advantages of each technology within reach, a new software model - Search Based Application (SBA) - has emerged, which can be defined as "any software application built on a search engine . . . whose purpose is not classic IR, but rather mission-oriented information access, analysis or discovery" [3].

1.1 Trump & Twitter

This work was done as a part of the course Search Engines and Information Retrieval Systems at Royal Institute of Technology (KTH). TrumpFacts was selected from a list of problems to be realized as an instance of project-based learning. The challenge was to build an SBA, analyzing current U.S. President Donald Trump's Twitter account by crawling and parsing Trump's Twitter page, storing its contents using a search engine, and through a graphical interface enabling users to generate and visualize content statistics.

Donald J. Trump, a successful businessman and now president of the USA, is a man that can be considered to have controversial views. Here are a few examples:

- Trump has proposed that a wall should be built along the USA border with Mexico, and that this wall should be funded by the Mexican government [5].
- Trump wants to immediately deport millions of undocumented immigrants who are in the USA on an expired visa [5].
- Trump has called global warming "an expensive hoax"[1], and claims that it was created by the Chinese[2].

President Trump has used Twitter[3], a social medium and a big contributer to the vast volumes of unstructured data on the Web, to post more than 30 000 tweets. The political content and the size of his feed makes it an interesting corpus to explore the often overlooked analytical capabilities of search engines [3]. This report investigates how these tweets can be visualized, specifically by visualizing them as a wordcloud.

[1] `https://twitter.com/realDonaldTrump/status/428414113463955457`
[2] `https://twitter.com/realDonaldTrump/status/265895292191248385`
[3] `https://twitter.com/`

1.2 Problem statement

This work aims to solve the problem of developing TrumpFacts, an SBA which enables users to visualize statistics of words used by Trump on his Twitter feed during different time periods.

1.3 Purpose

The purpose of TrumpFacts is to make possible analysis of when Trump made certain claims, and the changing frequency with which he used certain terms over time. In the context of the course, it is also hoped that TrumpFacts give insight to what search engines can be used for in extension to traditional IR.

1.4 Related works

Word clouds have been used in many previous works to visualize various documents. Vafa [6] visualized the inauguration speeches of Trump, Obama and Bush using tf-idf as the scoring mechanism. He found that there are a few trends that stand out: Trump, for example, used words such as "jobs", "workers", "factories", "winning" and "politicians." Prominent words in Obama's speech, on the other hand, were "healthcare", "warming", "women" and "father". Finally, prominent words in Bush's speech were "tyranny," "defended," and "freedom," highlighting the mood of the post 9/11 America.

Atenstaedt [7] created a wordcloud to visualize the content of the British Journal of General Practice (BJGP) over the years 2011 and 2016. He found, for example, that in 2011 the medical conditions that appeared most prominently were "cancer" and "depression". In 2016 the most prominent medical conditions were "cancer" and "diabetes", highlighting the issue that diabetes is reaching epidemic proportions. Another discovery Atenstaedt made is that the word "pressure" did not appear in the 2011 wordcloud but did in the 2016 one. The meaning here is ambiguous - since BJGP is a medical journal, the term pressure could be used in the context of "blood pressure" or "peer pressure". But it could also be referring to the fact that doctors in England are facing an ever increasing pressure of a growing and older population with a higher demand for healthcare [7].

1.5 Contribution

Word clouds are often used to create quick and intuitive visualizations of a document. This work however, explores the possibility of using the word cloud for more dynamic and interactive analysis by visualizing specified subsets of a large collection of data.

Additionally, to make the word cloud work well for the particular use case of analyzing Donald Trump's Twitter feed, TrumpFacts required a scoring function more advanced than simple term frequency. TrumpFacts uses a trend-based scoring (see section 4.2) to make key political terms more prominent in visual representation.

2 Method

To solve the task at hand, the TrumpFacts application was created, which generates word clouds from searches on Trump's collection of tweets. First, Trump's tweets were retrieved from a third-party database. (The initial idea of crawling his Twitter was discarded due to the Twitter API's limitations.) Elasticsearch was then used to index and parse these tweets. Via a Java backend, TrumpFacts performs a search on this Elasticsearch index to retrieve a working set of documents. Statistics on tweets and their terms are retrieved from Elasticsearch during the search. These statistics are then used within a trend-based scoring function to score each term that appears within the working set. Finally, each term and its corresponding score is passed to a third-party word cloud generator called Kumo, which generates the word cloud for that search. A graphical user interface serves as a wrapper for the application, allowing a user to specify search terms and displaying the resulting wordcloud. These various stages of the process are discussed further in their respective sections.

2.1 Retrieving Tweets

Initially, the Twitter API[4] was used to query Twitter for Trump's tweets. A shell script was written to fetch these tweets using cURL[5] and save them locally.

The Twitter API, however, has a limitation that prevents users from fetching more than the most recent 3200 tweets made by any user. Considering the fact that Trump has tweeted more than 30,000 times, another method was investigated. A website called Trump Twitter Archive[6] hosts various publicly collected data on Trump's tweets. The raw tweet data is hosted publicly on Github[7], updated hourly. This data was used instead of the data fetched with the shell script mentioned above, as it contains all tweets made from Trump's twitter account.

2.2 Elasticsearch

The backbone of TrumpFacts was Elasticsearch, an open source query and analytics tool. The version used was Elasticsearch 5.3.1[8]. Elasticsearch stored our database of tweets and was used to search for relevant documents and retrieve term statistics.

2.2.1 Storing

TrumpFacts stores tweets in Elasticsearch using its RESTful interface which accepts documents (tweets) in JSON format. Prior to running our application, a shell script was run, which used cURL commands to create an Elasticsearch index, specify necessary settings, and then feed in the JSON file of retrieved tweets. As Elasticsearch runs independently of the rest of the application, and maintains the created index even when the application is not running, this can be seen as an initialization step that was only needed to be performed once. New tweets were later passed into the index as they appeared over time.

[4] `https://dev.twitter.com/docs`
[5] `https://curl.haxx.se/`
[6] `http://www.trumptwitterarchive.com/`
[7] `https://github.com/bpb27/trump_tweet_data_archive`
[8] `https://www.elastic.co/`

The tweets were retrieved from the archive in a JSON format, with various tweet information (text, date, number of retweets, etc) listed as different fields for each tweet. For Elasticsearch, it was necessary upon index creation to specify how to parse the different fields, for instance what format the date was in and how to process the text field. Elasticsearch then handled treating dates as numerical data and performing textual analysis on the text of each tweet.

2.2.2 Linguistic preprocessing

Elasticsearch calls linguistic preprocessing *analysis* and offers a number of built in text analysis tools. For TrumpFacts, a custom analyzer was constructed, consisting of: a character filter to exclude URLs, another filter to replace non-letter characters with spaces, Elasticsearch's *standard* tokenizer, and several token filters, including a stop-word filter which removes the *_english_* list of common words such as "the" and "to," a filter which reformats words which contain an apostrophe (e.g. replacing "Iraq's" with "Iraq" but keeping "shouldn't" and "don't" unchanged), and a filter which makes all words uppercase.

2.2.3 Querying

Queries are performed on Elasticsearch from the Java backend, using the Java API[9]. From the GUI, the user specifies a date range and an optional search term. A query is then evaluated on the index, consisting of two parts: a date-range filter, which selects only tweets within the specified date range, and a union query using the search term. This query serves only as a filter to retrieve a relevant subset of documents – no scoring is performed at this stage.

2.2.4 Term statistics

Per-term statistics are retrieved using Elasticsearch's TermVectors API[10], which generates statistics on each term in a given document (or tweet, in this case). To retrieve a list of unique terms occurring in the working set, TrumpFacts iterates over the documents in the working set, calling the API for each and manually aggregating data for each term into a map-like structure. Of particular interest are the cumulative term frequency of a term over the working set of tweets, the document frequency of a term across all documents, and the document frequency of a term within the working set. Due to the limitations of Elasticsearch's precision, the main benefit of term vectors is to provide a list of terms; much of the actual per-term data must be calculated manually - df over the working set, for instance, is simply incremented for each term every time it is come across within a document in the working set.

2.3 Scoring Techniques

To create a word cloud, it is necessary to assign a score to each term in the working set WS ($WS \subseteq D$ being the result of our Elasticsearch query). A number of scoring

[9] `https://www.elastic.co/guide/en/elasticsearch/client/java-api/current/index.html`

[10] `https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-termvectors.html`

techniques were investigated and implemented before a final scoring function was settled on.

2.3.1 Term Frequency

The standard method of scoring terms for a wordcloud is simply to calculate the frequency of each term. In our case, this means treating the entire working set as a single document, and calculating the term-frequency ($\text{tf}_{t,WS}$) over it for each term.

Downsides to this approach are the resulting high prevalence of generically common terms (in our case, "America," "we," etc) in any given subset of tweets. Towards the goal of TrumpFacts, it was desirable to have a stronger emphasis on locally prevalent terms, unique to a given working set.

2.3.2 TF-IDF

tf-idf, a classic scoring method, was then investigated. tf-idf is a scoring technique commonly used in search engines to give a ranking to a document-term tuple. It is typically defined as the product of two components: $\text{tf}_{t,d}$, the term frequency of the term t in some document d, and idf_t, the inverse document frequency of the term, usually defined as

$$\text{idf}_t = ln\left(\frac{\text{N}}{\text{df}_t}\right)$$

where df_t is the document frequency of t (the number of documents d that contains t) and $N = |D|$, the number of documents in D. tf-idf is then defined as:

$$\text{tf-idf}_{t,d} = \frac{\text{tf}_{t,d} * \text{idf}_t}{|d|}$$

A wordcloud using tf-idf would have to aggregate over all documents in the working set in some way, for example by summing $\text{tf}_{t,d}/|d|$ over each document $d \in WS$:

$$\text{tf-idf}_{t,WS} = \left(\sum_{d \in WS} \frac{\text{tf}_{t,d}}{|d|}\right) * \text{idf}_t$$

The downside to this scoring function is that it does not work well in practice, giving a result not very different from that of $\text{tf}_{t,WS}$ as a scoring function – although some more uncommon words are given a higher score, common words are weighed too heavily, so results for small time ranges are similar to results for broader ranges. The goal of TrumpFacts, rather, is to highlight locally prevalent terms.

2.3.3 Trend-Based Measurement

Finally, a custom measurement called a "trend-measure" was derived, intended to score terms that were unusually prevalent in a given set of documents higher. The trend-

measure for a given term t is defined as follows:

$$\begin{aligned}\text{trend}(t) &= \frac{\text{proportion of docs in } WS \text{ containing } t}{\text{proportion of docs in } D \text{ containing } t} \\ &= \frac{\text{df}_{t,WS}/|WS|}{\text{df}_{t,D}/|D|} \\ &= \frac{\text{df}_{t,WS} * |D|}{\text{df}_{t,D} * |WS|}\end{aligned}$$

where $\text{df}_{t,WS}$ is the document frequency of the term in the working set, $\text{df}_{t,D}$ is the document frequency for the term across all documents, and $|WS|$ and $|D|$ are the number of documents in the working set and the entire set of documents, respectively.

If a term is more prevalent in WS than in D, $\text{trend}(t) > 1$. Otherwise, $\text{trend}(t)$ would be < 1. Thus, the trend measure serves to weigh the score up or down depending on how "trendy" a term is.

2.3.4 Final Scoring Function

Finally, the score for a term t is calculated as follows:

$$\text{score}(t) = \text{tf}_{t,WS} * ln(1 + \text{idf}_t * \text{trend}(t))$$

$\text{tf}_{t,WS}$ for this function is calculated by treating the working set WS as a single document. To manually calculate this value, TrumpFacts iterates over every document in the working set, summing up the tf of each term, so $\text{tf}_{t,WS} = (\sum_{d \in WS} \text{tf}_{t,d})$. idf_t for this function is calculated using the document frequency of a term over the entire set D. The natural logarithm is applied to the trend measure and idf, in order to prevent tf from becoming insignificant. 1 is added to prevent negative results.

This scoring function serves the goal of TrumpFacts by giving high scores to words which are not only frequent in a working set, but also unusually common within that set compared to the entire dataset, with the idea that these are the most interesting to the user. Term frequency is present to prioritize more frequently-occurring words, idf balances it by not allowing too-common (and thus, irrelevant) words to have the highest scores, and the trend factor weighs in favor of "trendy" terms.

2.4 GUI

The graphical user interface was constructed in Java AWT/Swing using standard library components. The user modifies the range of dates, types in an optional search query, selects a scoring option, and presses "submit." A settings menu is also available, containing options to edit wordcloud features such as size, wordcount, colorscheme, and shape, as the wordcloud can have custom shapes using loaded PNG files as background templates. These parameters are passed to the Java backend, a search is performed, and a wordcloud is generated within seconds. When a wordcloud has been generated, the image can be saved as a PNG image file.

2.5 Wordcloud

TrumpFacts utilizes the API of a free wordcloud generator called Kumo [8]. The API takes an array of (word, score) tuples, and generates an image in which the relative

font size of a word corresponds to its score. Although Kumo expects the score to be a simple tf of each term, and even contains built-in tools to calculate frequency from a text file, TrumpFacts passes in the custom scores generated by the trend-based function, resulting in a wordcloud that highlights locally prevalent terms found within the subset of documents being analyzed.

3 Results and analysis

The following figures show the TrumpFacts GUI (visible are the various options available to the user), as well as wordclouds used to compare Trump's words use in different time periods, utilizing features such as various scoring methods and term filtering.

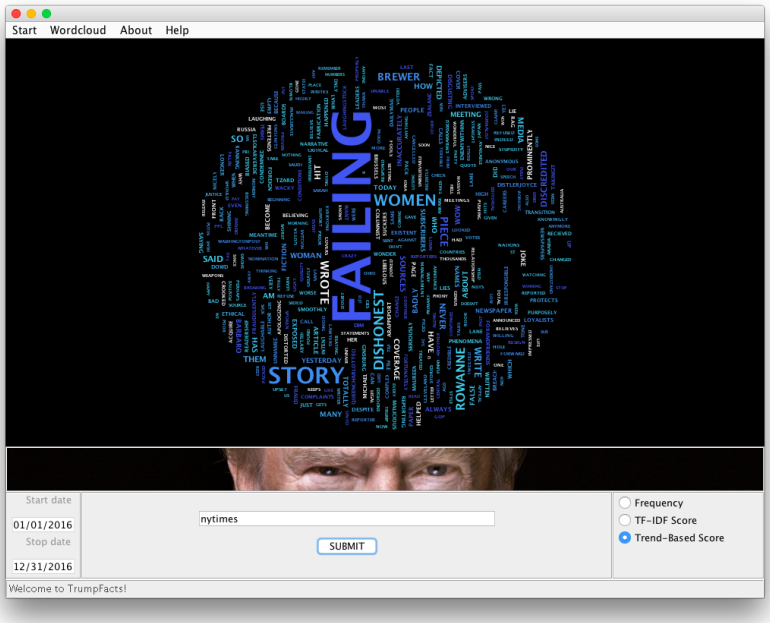
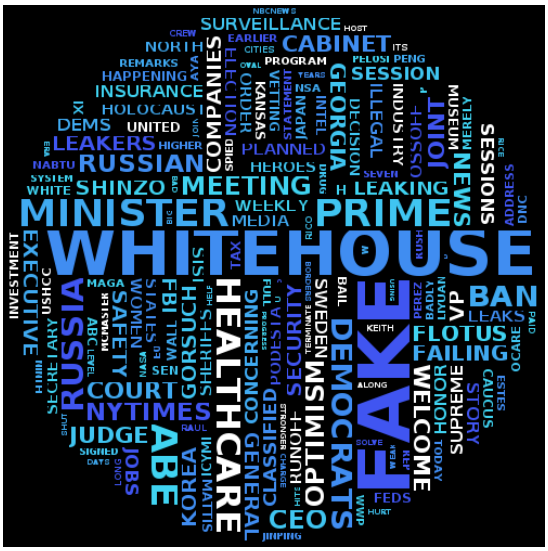


Figure 1: The GUI, featuring a generated wordcloud.

(a) The two months leading up to Trump's formal announcement of candidacy on June 16th 2015. Trend-based scoring, no search terms.

(b) The two months following Trump's formal announcement of candidacy on June 16th 2015. Trend-based scoring, no search terms.

Figure 2: Two different time periods.



(a) The first 100 days of Trump's presidency. Trend-based scoring, no search terms.

(b) The first 100 days of Trump's presidency. term frequency-based scoring, no search terms.

Figure 3: Trend-based and term frequency-based scoring.

(a) The first 100 days of Trump's presidency. Trend-based scoring, filtered with the terms "north" and "Korea".

3.1 Analysis

Looking at the period just before (figure 2a) and just after (figure 2b) Trump's announcement of his candidacy, there is a clear change in vocabulary towards the topic of illegal immigration, especially that from Mexico - an issue at the core of Trump's campaign.

Figure 3a and 3b indicate that the trend factor does seem to succeed in favoring terms that better represent the essence of the messages in Trump's tweets. Not only does figure 3b contain more common words that reveal very little context, opinion or disposition, but the distribution of the scores causes the wordcloud to print fewer words with increased sizes, providing less information to be analyzed.

Figure 4a is simply an example of how the working set of tweets can be filtered using search terms. This is convenient for analysis within a certain issue or topic. The wordcloud may or may not give a fair representation depending on the boolean query's effectiveness in filtering out relevant tweets.

4 Discussion

This section discusses the implications and possible alternatives for some of the design- and methodology choices, and provides some more subjective evaluation of the results with regard to the aim of this work.

4.1 Elasticsearch

Elasticsearch proved to be a tool with many different pros and cons. Starting with the pros, indexing tweets into the REST-API was not an issue and the linguistic preprocessing tools (tokenization, stemming and stop words) provided by Elasticsearch worked as intended and probably saved us a lot of time. The search engine also provides many

different statistics about the data, and does this while maintaining a relatively short time to process the queries issued to it.

However, there are also a number of cons associated with using Elasticsearch. We found the documentation, especially for the Java API, to often be lacking, which made implementation of many features quite difficult. For example, extracting the total term frequency of a certain term over all tweets in a working set is simply not possible using Elasticsearch. Thus, we had to manually calculate tf by going through the term vectors of the documents and summing up frequencies. Another downside to Elasticsearch is the fact that it approximates some values, such as document frequency. This is something we did not know until we started noticing that some of our results were odd, and we ended up calculating that manually as well.

4.2 Scoring Function

The frequency function performs as one might expect, words that are really common in Trump's vocabulary get a very large font. These common words can be words that are interesting and highlight what is on Trump's mind, sadly that is rarely the case as many wordclouds using this scoring function only contain very common words that do not reveal anything special about Trump's opinion or position regarding subjects that are relevant in some working set of tweets.

The design goal of the trending function was to give terms that occur abnormally much in the working set a higher score. This is a goal that is, as many things in IR, hard to measure. Looking at the results though, specifically Figure 3a and 3b, it does seem like very common words that appear when using term frequency as a scoring function grow smaller or disappear when using the trending function. Words that appear instead seem to be words that are relevant for the working set. This is one indication that the scoring function does what it was intended to do.

4.3 Wordcloud as a Visualization Tool

As a visualization tool, wordclouds have both good and bad properties. Something we noticed was that the wordcloud generation took much longer than expected, and for a user having to wait several seconds for a result might not be feasible. Another thing we wanted was to have each word retain their position within the wordcloud in order to make it easier to perceive the difference between two slightly different wordclouds, for example to visualize how certain terms' scores change over time. Intuitively, this might sound trivial, but constructing such a wordcloud algorithm might be harder than expected, and might slow down computations even further. Another thing that is lost by generating wordclouds is the context of each tweet. If for example "Fox News" and "terrible" are in the same tweet, there is no way of telling in what context they were originally written when they have been scrambled and presented in a wordcloud.

5 Concluding remarks

The conclusions of this work are that search engines seem to provide flexible models for storing and structuring data for quick and scalable retrieval, as well as tools for language (pre)processing and analytics of both structured and unstructured data, allowing a new breed of applications to solve problems on the vast volumes of data that are created.

Furthermore, that wordclouds can be a quick and intuitive way to visualize a dataset, but that their tumultuous appearance makes it very difficult to identify subtle differences between one cloud and another. They also lack axis, making it difficult to tell how each term is assigned its priority in the cloud.

5.1 Future work

Future work regarding this subject could include some kind of topic or semantic analysis on Trump's tweets. Other scoring functions could also be investigated. Finally, other datasets such as Hillary Clinton's twitter feed could be used to highlight differences and similarities between the two presidential candidates. Maybe even include visualizations of statistics about Trump's speeches vs Hillary's speeches, and in this case other American presidents such as Obama, Bush or Kennedy could be included.

References

[1] P. Zikopoulos, D. deRoos, C. Bienko, and R. Buglio, *Big data beyond the hype: A guide to conversations for todays data center*. McGraw-Hill Education, 2014, ISBN: 9780071844659. [Online]. Available: https://books.google.se/books?id=MdfPoQEACAAJ.

[2] *IBM - what is big data?*, https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html, Accessed: 2017-05-05.

[3] G. Grefenstette and L. Wilber, *Search-based applications: At the confluence of search and database technologies*, ser. Synthesis lectures on information concepts, retrieval, and services. Morgan & Claypool Publishers, 2011, ISBN: 9781608455072. [Online]. Available: https://books.google.co.uk/books?id=3NAt4cAwVV4C.

[4] MongoDB, "White paper:top 5 considerations when evaluatingnosql databases", Tech. Rep., Nov. 2016. [Online]. Available: https://www.ascent.tech/wp-content/uploads/documents/mongodb/10gen-top-5-nosql-considerations-february-2015.pdf.

[5] J. Johnson, "I will give you everything. Here are 282 of Donald Trump's campaign promises.", *The Washington Post*, Nov. 28, 2016, Accessed: 2017-05-04. [Online]. Available: https://www.washingtonpost.com/politics/i-will-give-you-everything-here-are-282-of-donald-trumps-campaign-promises/2016/11/24/01160678-b0f9-11e6-8616-52b15787add0_story.html.

[6] K. Vafa, *Inauguration word clouds with tf-idf*, Accessed: 2017-05-12, Jan. 21, 2017. [Online]. Available: http://keyonvafa.com/inauguration-wordclouds/.

[7] R. Atenstaedt, "Word cloud analysis of the BJGP: 5 years on", *British Journal of General Practice*, vol. 67, pp. 231–232, 658 May 2017. DOI: https://doi.org/10.3399/bjgp17X690833.

[8] *Kumo*, https://github.com/kennycason/kumo, Accessed: 2017-05-08.