Election Prediction Based on Wikipedia Pageviews

Georgiana Diana Ciocirdel
Vrije Universiteit Amsterdam
g.d.ciocirdel@student.vu.nl

Mihai Varga
Vrije Universiteit Amsterdam
m.varga@student.vu.nl

1. INTRODUCTION

On November 8th 2016 the United States of America will elect a new president and the incumbent president, Barack Obama, will step down and be replaced by one of the following candidates (listed here in no specific order): Hillary Clinton (the Democratic nominee), Donald Trump (the Republican nominee), Gary Johnson (the Libertarian nominee), Jill Stein (the Green party nominee) or one of the other 24 third-party candidates and independents. Since August 2015 the US have seen a series of public events closely related to this year's elections - primary debates broadcast live on national television, primary elections and caucuses (between February and June 2016), national nominating party conventions and now, finally, the live presidential debates between two of the nominees, Hillary Clinton and Donald Trump.

The campaign and election periods create a tremendous buzz on the internet. It is common that people go to social networks to express their agreement or disagreement with candidates or (more specifically for news outlets) post live updates and news about the election events and polls. Besides the social networks hype, people also tend to query search engines more, about topics closely related to the election events (like for example candidate names, topics discussed during the debates or even about the presenters of the debates). If we look at data extracted from Google Trends[1] from the past 30 days (Figure 1) about the two most popular candidates (as per US national polls[2]) we can see that the number of searches for the two presents a spike (a sudden growth) around the dates of September 26th and (more or less) October 9th - these are the dates of the first two presidential debates.

Very often the first result returned by search engines is the Wikipedia[3] page matching the search query. Wikipedia is also the go-to place when looking for extensive details about a person, a place, an object or an event. Wikipedia is an online encyclopedia first published in the early 2000's[4] and currently maintained by contributors around the world. It has over 40 million articles[5] in all supported languages and over 5 million in English only. Wikipedia is by no means a social network, but rather a content repository.

In this paper we want to assess the impact the various election events from the past year and a half have had on the number of views of Wikipedia pages. We are particularly interested in discovering which (or if any) Wikipedia pages register a spike in the readers' activity during the election events. Moreover, based on the number of views the spiking pages have and on the correlation between these pages and the presidential nominees or their political program, we would like to assess if elections can be predicted.

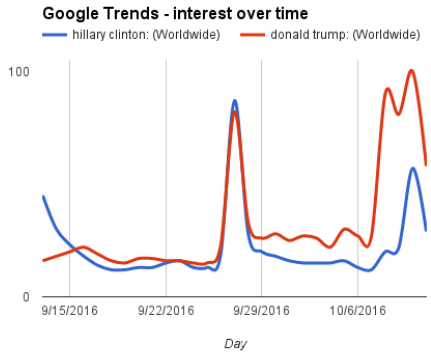


Figure 1: Interest over 30 days in Google Trends in Hillary Clinton and Donald Trump.

2. RELATED WORK

In this section we will talk about two models we have used to identify trending or spiking pages on Wikipedia, a parametric model and a data-driven model. We will also talk about predicting election outcome.

2.1 Trend Detection

The most challenging part of our work is to determine spikes and trending periods in the number of views of Wikipedia pages. For any given Wikipedia page each visit represents an event we observe in a time-varying signal. For this observed signal, we are interested in windowing it and determining which windows present a deviation of the signal from its "normal" behavior, in other words, where does the signal spike, where does the number of views increase so much that we (as humans) could say that the page has become trending?

In his Master's thesis [4] Stanislav Nikolov highlights two basic approaches in identifying these spiking/trending windows: parametric and data-driven models. We will present a parametric model we have used in our research which was

[1]https://www.google.com/trends/
[2]http://www.realclearpolitics.com/epolls/latest_polls/
[3]https://en.wikipedia.org/wiki/Wikipedia:Introduction
[4]https://en.wikipedia.org/wiki/History_of_Wikipedia
[5]https://meta.wikimedia.org/wiki/List_of_Wikipedias

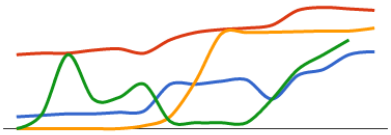


Figure 2: Possible ways in which a signal becomes trending.

inspired by an algorithm published on Stackoverflow[6] and a data-driven model documented by S. Nikolov in his thesis.

2.1.1 A Parametric Model

A parametric model which identifies trends and spikes in a time varying signal relies on a set of parameters estimated from the input data in order to detect anomalies in the signal. S. Nikolov highlights [7] that this approach in identifying trends is not reliable due to too many ways in which a signal could become trending (Figure 2) - there could be a gradual rise in the signal (red), the signal could suddenly jump "a step" (yellow), it could be alternating between increasing and decreasing (blue and green) and so on. A fixed value for a parameter might not properly capture all the possible changes correctly.

However, we will see that spikes and trends in Wikipedia data can be modeled with a parametric algorithm presented on Stackoverflow [8] in 2014. The algorithm performs as follows: the algorithm keeps track of a moving mean and a standard deviation of the observed events in a time varying signal. Initially, the mean and the standard deviation are computed over the first window of K observations. Then, for each observation with an index larger than or equal to K the algorithm checks whether the given point is a preset number of standard deviations (called *dist*) away from the moving mean and outputs 1 if it is, or 0 otherwise. On each step the moving mean is updated as the mean of its previous value and the value of the current point in the time series; the standard deviation is updated to be half of the distance between the previous standard deviation and the distance between the new point and the previous mean. An important feature of the algorithm is that a spike in the data will not corrupt the moving mean or standard deviation, as spikes in the signal do not contribute to the update. Another parameter called *influence* (with values between 0 and 1) could be included in the algorithm to allow the spikes to influence the moving mean.

[6]http://stackoverflow.com/tour

[7]https://snikolov.wordpress.com/2012/11/14/early-detection-of-twitter-trends/ - The Problem with Parametric Models

[8]http://stackoverflow.com/questions/22583391/peak-signal-detection-in-realtime-timeseries-data/22640362#22640362

input : input signal, the parameters K, *dist*, *influence*
output: a binary output signal

initialize *StdDeviation* as the standard deviation of the first K observation in the input signal;
initialize *Mean* as the mean of the first K observations in the input signal;

for $i \leftarrow 0$ **to** K - *1* **do**
  output 0;
end

for $i \leftarrow K$ **to** *the end of the input signal* **do**
  if *current observation* $>$ *dist* $*$ *StdDeviation* + *Mean* **then**
    output 1;
    StdDeviation = *StdDeviation* + *influence* $*$ abs(current observation $-$ *Mean*) / (1 + *influence*);
    Mean = *Mean* + *influence* $*$ current observation / (1 + *influence*);
  else
    StdDeviation = (*StdDeviation* + abs(current observation $-$ *Mean*)) / 2;
    Mean = (*Mean* + current observation) / 2;
  end
end

Algorithm 1: A simple parametric algorithm that finds peaks and trends in an input time-varying signal.

2.1.2 A Data-Driven Model

The disadvantage of a parametric model is that we need to estimate the values of the parameters and very often (in our experiments as well), the parameters are set to magic numbers. A "cleaner" approach in identifying trends and spikes is data driven rather than parametric - instead of trying to hard set parameter values, we could compare the input signal with real examples of time series that we know are trending or non-trending and, based on the level of their similarity, attribute the signal to either one of the two classes (trend/non-trend). Hendrickson, Kolb, Lehman and Montague build on this idea in their white paper [3] and propose an algorithm inspired by and similar to the one described by Nikolov in his thesis. In our research on Wikipedia page views, we have used the algorithm presented in the white paper, as it was backed up by samples of code and thus easier to understand.

This algorithm was first used by Nikolov [4] for finding trending Twitter topics (keywords in tweets that are suddenly used by people more than they are on a regular basis). The algorithm relies on the existence of two initial sets of time series that correspond to topics that have either become trending or not. We call these reference sets $R+$ and $R-$, respectively. We are interested in computing the probability of an observed signal of belonging to each of the classes. The two reference sets can be viewed as a very small number of different signal sources each producing noisy versions of a prototype signal. So for example even though the signals in $R+$ are different between each other, they can be clustered into patterns of activity [9] and they have all been produced by "trending" signal sources. And the same applies to the

[9]https://snikolov.wordpress.com/2012/11/14/early-detection-of-twitter-trends/ - Figure 4

non-trending signals in $R-$. The sequence of observations in the input signal - in Nikolov's thesis, this would be the sequence of tweets related to a topic; in our case this would be the sequence of accesses of a Wikipedia page - is a sequence of events that occur randomly over a period of time. The evolution in time of the signal is described by a stochastic process and each observation in the signal is independent to all the other observations. Nikolov [4] proposes a stochastic model to estimate the closeness of a signal s to a source q:

$$\mathsf{P}(s \textit{ generated by } q) \propto e^{-\gamma d(s,q)} \tag{1}$$

where $d(s,q)$ is a distance function computing the distance between the signal s and the source q.

Hendrickson, Kolb, Lehman and Montague build on this theory and propose a trend detection algorithm in their whitepaper [3]. To classify a signal s as trending or non-trending, we first have to find the distance between s and each q, where q is in $R+$ or $R-$. First, both signals, s and q, are unit-normalized and then the Euclidian distance is used as the distance function:

$$d(s,q) = \sum_{i=1}^{N} (s_i - q_i)^2 \tag{2}$$

where N is the length of the two time series and q_i and s_i are the ith observation along the two signals.

Having defined our distance function, we will also define a weight function similar to the probability density function modeling the probability of signal s having been produced by a source q (equation 1):

$$W(s,q) = e^{-\gamma d(s,q)} \tag{3}$$

As we are actually interested in finding the probability of s having been produced by **any** source q in $R+$ or $R-$, we will sum up all the weights and introduce a new metric:

$$\eta(s) = \frac{\sum_{q \in R+} W(s,q)}{\sum_{q \in R-} W(s,q)} \tag{4}$$

where $\eta(s)$ quantifies how much s looks like a trend $R+$ rather than a non-trend in $R-$. We will say that the signal s exhibits a spike or a trend if the value of the function $\eta(s)$ is above a certain threshold θ.

In practice the reference trends/non-trends and the input signal have different lengths. If the input signal is shorter than the reference set, then the above algorithm will not be able to classify it. If the input signal is longer than the reference signals, then it can be windowed into chunks of reference length and the algorithm will be run over each window. The reference signals must be of the same length in order to produce one unique window over s.

We conclude this subsection with algorithm 2 showing the pseudo-code for the data-driven approach for trend detection.

input : input signal s, $R+$, $R-$, parameter θ
output: $true$ if $s \in R+$ or $false$ otherwise for each window in s of reference length

foreach *window w of reference length in s* **do**
  do unit-normalization over w;
  apply $\log_{10}$ over each element in w;
  $WR+ = 0$;
  $WR- = 0$;
  foreach $q \in R+$ **do**
    $WR+ \mathrel{+}= W(w,q)$;
  end
  foreach $q \in R-$ **do**
    $WR- \mathrel{+}= W(w,q)$;
  end
  $\eta = \frac{WR+}{WR-}$;
  if $\eta >= \theta$ **then**
    output $true$;
  else
    output $false$;
  end
end

Algorithm 2: A data-driven algorithm that finds trends and non-trends in a windowed signal based on the two reference sets, $R+$ and $R-$.

2.2 Predicting Elections

Trying to predict the outcome of elections by using publicly available data from social networks is quite common in research. In his paper [1], for example, Daniel Gayo-Avello cites and analyses a number of different other papers that claim to have found a trustworthy method of predicting who will win and who will lose elections based on tweets. However, he is skeptical that the methods presented in these pieces of research are accurate and argues that in fact it is not possible to make such predictions from social network interactions, primarily due to a strong imbalance in the demographic groups represented on Twitter. Actually, as Bloomberg Politics show in one of their blog posts [2], polling is anything but trivial and it is essential that polls capture the correct distribution of the various demographic groups that participate in the elective process. With this is mind, we doubt that our results will give an accurate prediction of the upcoming election for two reasons: firstly, Wikipedia being an information source and not a social network, it is impossible for us to even analyze the feelings or the opinions of its users, so there is no way of telling if the readers that have accessed a certain page are pro, against or neutral to the given subject; secondly, Wikipedia query logs are captured in such a way, that it is impossible to infer anything related to the demographic distribution of its users (such as gender, race, level of education, etc.).

However, in our experiments we have also tried to determine whether a higher number of people accessing pages related to the candidates is an indication of a general preference towards that candidate or not.

3. RESEARCH QUESTIONS

In our experiments on detecting trends and spikes in the number of views of Wikipedia pages during presidential election events in the United States we will try to answer the following questions:

- Which pages do spike or become trending on Wikipedia during the election events?
- Can we predict the outcome of polls (and eventually

elections) based on the number of views the pages related to various candidates receive?

The end goal of our project is to create a visualization in the form of a web page to help the reader and ourselves answer these questions. This web page will hold information about each election event (starting from August 2015), about trending pages associated with each event and also about polls.

We are also interested in a reliable and flexible way of parsing the input data. Wikipedia has over 40 million online articles, 5.2 of which are in English and 1.3 in Spanish. The team regularly publishes page views statistics at https://dumps.wikimedia.org/other/pageviews/. These statistics are hourly aggregated and hold information about the language of the article, the medium it has been accessed from, the name of the article and the number of views from that hour. We are interested in looking at pages in English or Spanish (we think this is the most relevant language segment for US citizens). After this filtering by language our solution needs to work with approximately 82GB of gzip-ed data (the unfiltered data has a size of 500GB). So on the technical side of the project another question arises:

- Which tools and languages should we use that will allow us to easily filter, window and process the available data?

The next two sections will address these questions and will present the technical solutions we have chosen, how we have implemented the two algorithms described in the previous section and what conclusions have we drawn from our results.

4. PROJECT SETUP

4.1 Data Collection

The necessary data for our project (the Wikipedia pageviews) have been downloaded to the SURFSara[10] cluster. We have also gathered data about the election events that have taken place between August 2015 and September 2016: Democratic[11] and Republican[12] presidential debates and forums, Democratic[13] and Republican[14] primary elections and finally data about the undergoing presidential debates[15] between Hillary Clinton and Donald Trump. From these events we have collected data about the participants, places where they have taken place and, the names of moderators and the topics of the discussions (in case of debates). There is a total of 138 election events we have data about.

We have also extracted poll data from the RealClear Politics website[16], which aggregates data from multiple polls. The website allows downloading the poll results in a JSON format. This amounts to one poll result per day for three different polls: Clinton vs. Trump, Clinton vs. Sanders and all the Republican candidates vs. each other.

4.2 Technologies Used

We started our work by analyzing small portions of data in order to generate a "ground truth" result that we could use to assess that our future pipelines produced the desire result. We have used python for parsing a small number of files (each containing Wikipedia dumps for one hour). The output of the script was a csv file. To plot our results we have used Google Sheets[17], which may not sound like the optimal tool to use for plots, but we found it very easy to import csv data into a Sheet, to generate and advance edit plots and ultimately to share plots and linking them in other documents.

Later on, in order to process all the data from Wikipedia that we wanted to inspect we have decided to use Spark[18] and to write our batch pipelines in Java. We chose to work with Spark because it was easy and intuitive to use for pipeline stages definitions and general flow specification and jobs deployment and tuning. Also, it did not require any extra installations on SURFSara. For Java we have used the JDK 7 as this was the version available on the cluster. Java and Spark have been an easy to use combination - the definition of stages is intuitive with names like "*filter*" or "*reduceByKey*" and we found it very natural to encapsulate the logic for each stage into stand-alone objects passed in to the stage.

As for the trend detection algorithm, we have started with the data-driven algorithm and then moved to the parametric model. The algorithm presented by Hendrickson, Kolb, Lehman and Montague [3] in their white paper was backed up by python code on Github[19]. Starting from their code and the explanations they gave in their paper, we have translated the code to Java. We have also downloaded their code and ran it on small portions of our data to again find a "ground truth" that we used in unit tests to assess that our translation was running as expected.

Also from their Github repository we have used the reference sets they provided for the two clusters of trending and non-trending topics. These topics had been extracted from Twitter. We considered this to be a major drawback for our experiments and we searched for a viable reference set for activity on Wikipedia, but unfortunately we couldn't find one online. We will later discuss the impact this had on our findings.

The parametric algorithm presented on Stackoverflow[20] was originally written in Matlab. We have again translated it in Java (and simplified it). Also, we have used the original algorithm to generate expectations for our unit tests and also to fix the parameters needed by the algorithm to match the magnitude of our data set.

In the end, for producing the desired visualization, we worked with HTML and JavaScript. We used the Highcharts[21] library to produce the charts in our visualization

[10] https://userinfo.surfsara.nl/systems/lisa
[11] https://en.wikipedia.org/wiki/Democratic_Party_presidential_debates_and_forums,_2016
[12] https://en.wikipedia.org/wiki/Republican_Party_presidential_debates_and_forums,_2016
[13] https://en.wikipedia.org/wiki/Democratic_Party_presidential_primaries,_2016
[14] https://en.wikipedia.org/wiki/Republican_Party_presidential_primaries,_2016
[15] https://en.wikipedia.org/wiki/United_States_presidential_election_debates,_2016
[16] http://www.realclearpolitics.com/epolls/latest_polls/
[17] https://www.google.com/sheets/about/
[18] http://spark.apache.org/
[19] https://github.com/jeffakolb/Gnip-Trend-Detection
[20] http://stackoverflow.com/questions/22583391/peak-signal-detection-in-realtime-timeseries-data/22640362#22640362
[21] http://www.highcharts.com/

and Materialize CSS[22] for the general look and feel of the page.

4.3 Pipeline Building Blocks

In our experiments we have used two Spark batch pipelines, which we called *Trend Detector Pipeline* (Figure 3) and *Trend Extractor Pipeline* (Figure 4).

4.3.1 Trend Detector Pipeline

In its first stage, the *Trend Detector Pipeline* reads a batch of files from the SURFSara cluster, stored in the Hadoop File System. The Spark API allows reading full files and returns a String to String mapping from file name to file content. Spark also provides a Java method for reading files line by line, but this method proved ineffective and caused our jobs to hang on the cluster. Each ingested file has an approximate size of 47MB, is compressed in a gzip format and has a name with the following pattern: pageviews-YYYYMMDD-HH0000.gz, where YYYYMMDD and HH represent the date and the hour for which the file stores page views stats. Luckily, HDFS allows Spark to seamlessly read and decompress gzipped files. The content of each file is split into lines. On each line we can find the following tokens, split by space: a code of the form "language_code[.medium]", where the language code is the ISO code of the language in which the article is published and "medium" is an optional token holding information about the medium from which the article has been accessed; article name; number of views for that hour for the specific article.

We use a *flatMap* operation on the next stage in order to remap the name of the file to each line in the file (the *flatMap* operation allows us to emit zero or more records for each input record). A subsequent stage splits each line into tokens and emits a remapping from article name (topic) to a tuple of timestamp (extracted from the file name) and number of views the page received for that timestamp.

We also only keep track of those articles that are either in English or Spanish. Initially, we did not filter out any information, but when deploying to cluster, the jobs would get stuck on the shuffle phase, due to a very high number of keys. We have eventually decided to look only at the rows that started with "en" or "es", regardless of the medium they have been read from.

A *reduceByKey* stage performs a mergesort over the keys in the mapping of the previous stage. The result is that now the article name will be associated with an array of tuples holding the timestamp and number of views for that timestamp.

We then *map* the trend detection algorithm on each record. For each article, we window the array of tuples into buckets of fixed size (the size of the signals in the reference set for the data driven model and a bucket of 72 hours for the parametric model) and feed each window to the algorithm. Regardless of which algorithm we choose to use, it will output a parameter telling us whether that window is trending or not - the data driven model will output η (showing how close the window is to the trending signals as compared to the non-trending signals) and the parametric model will output 0 (non-trending) or 1 (trending). We add this output to the previous tuple, so now the article name will be mapped to an array of tuples of timestamp, number of views and a floating point value, η, 0 or 1.

[22]http://materializecss.com/

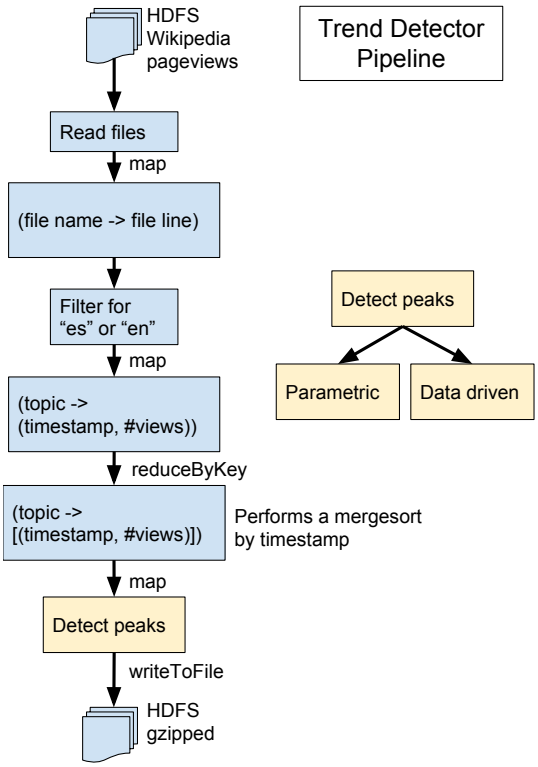


Figure 3: Trend Detector Pipeline building blocks.

The non-parametric model needs to read the reference sets of trending and non-trending signals. We have serialized these signals (which are basically one-dimensional arrays), saved them to binary files and added these as jar resources. In this way, they are copied over on each worker, which will then de-serialize the information and use it. The two files amount to 2MB in size.

In the final stage the RDD is written to HDFS files on the SURFSara cluster. After trying different partition values and watch our jobs fail, we decided to let Spark choose the optimal number of partitions, so the job finished successfully.

4.3.2 Trend Extractor Pipeline

In order to actually detect if and when the page views timeseries spike, we have written the *Trend Extractor Pipeline*. First, it reads the files produced by the previous pipeline. In the resources of the deployed jar we have also included a small text file (500bytes) with the dates of each election event we are interested in, which is read by each worker. We apply a *flatMap* operation on the resulting mapping (file name to file content): we check whether a spike occurs in the array of tuples (timestamp, number of views, η, 0 or 1) from 48 hours prior to an election events to 48 hours after an election event. If so, we wrap this information in a JSON object.

We then repartition the RDD into a single partition so that it will be written into one single file and dump the JSONs produced before to a text file.

4.4 Deployment to Cluster

To ease the packaging and deployment process we have

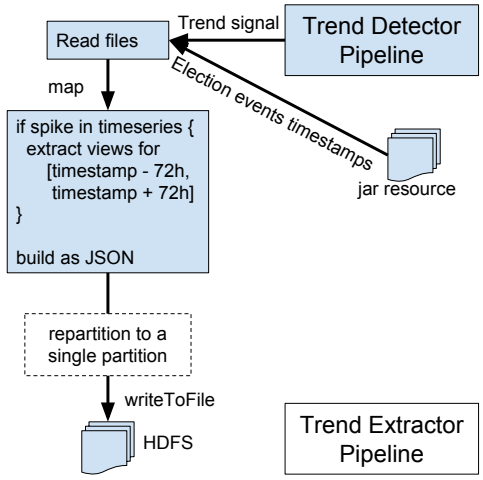


Figure 4: Trend Extractor Pipeline building blocks.

used Apache Maven[23]. Before deployment we packed the project and all its dependencies and resources into an uber-jar[24] and *spark-submitted*[25] it to the SURFSara cluster.

4.5 Visualization Walk-through

In our visualization we have included the following components:

- A chart like the one in Figure 5. On the x axis we plot the date of the event and on the y axis the number of spiking Wikipedia topics for that particular event. Each event is drawn as a bubble and the size of the bubble is proportional to the total number of page views for the spiking pages. As you can see in Figure 5 this is more or less the same for all the events;
- When clicking on an event the page will be populated with the names of the Wikipedia articles that have spiked before, during or after that event. For each of these pages we have included a sparkline (like the one in Figure 6) that shows how the page views have changed over 48 and 24 hours prior to the event, during the event and 24 and 48 hours after the event (the red dotted lines);
- Finally, we have also included three charts showing how the polls vary withing the 96 hours. The charts show polls for Trump vs. Clinton, Clinton vs. Sanders and the Republican candidates vs. each other.

5. EXPERIMENTS

As we have mentioned in a previous section we started our experiments locally on the data from Wikipedia by looking at stats spanning over two weeks, from September 14th 2016 to September 28th 2016. We chose these dates because the first presidential debate took place on September 26th 2016. To ease the pain on our local machines, we have only looked

[23]https://maven.apache.org/
[24]http://stackoverflow.com/questions/11947037/what-is-an-uber-jar
[25]http://spark.apache.org/docs/latest/submitting-applications.html

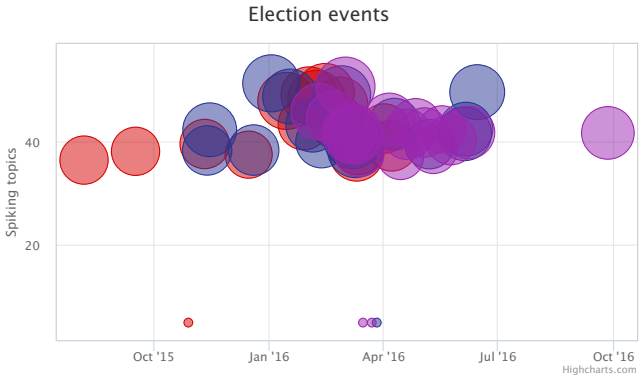


Figure 5: Visualization chart.

Figure 6: Visualization sparkline.

at views for the pages containing the keywords "clinton", "trump", "republican" or "democrat" in their name and we have only selected the first 20 pages by number of views. In these first experiments we wanted to gain insight on how the data is structured, whether the theory we started from (that Wikipedia pages related to election events become trending during these events) is even true and also to generate a "ground truth" for our future experiments on Spark. In Figure 7 we see that the pages of Donald Trump and Hillary Clinton are the first two pages by the number of views from these top 20 and also that they spike the most right after the debate. So we were convinced that we might obtain interesting results from running on larger data sets.

We moved on to building the first pipeline, *Trend Detector Pipeline*, described in the previous section. In the beginning we used the data driven algorithm to search for trends in the Wikipedia page views timeseries. At this point we were not yet filtering by the language of the page (we later decided to only keep those pages in English and Spanish) so when

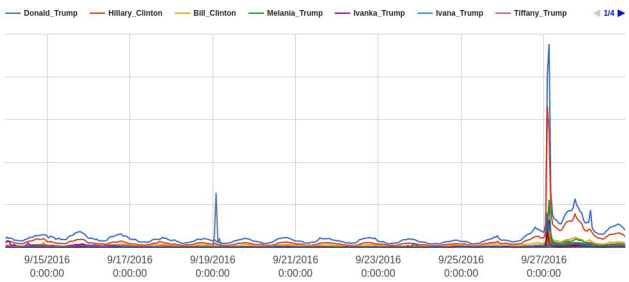


Figure 7: Experimenting on our local machines. Top 20 Wikipedia pages (by number of views) between Sept 14th and Sept 28th 2016, containing the keywords "clinton", "trump", "democrat" or "republican" in their title.

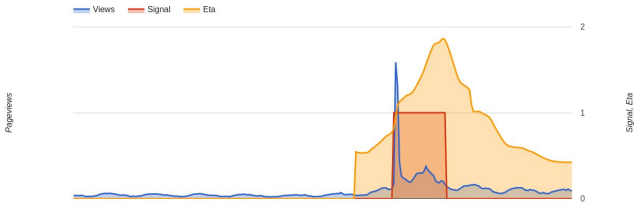


Figure 8: Experimenting on SURFSara. Number of views on Hillary Clinton's Wikipedia page (blue), η (orange) and the binary signal (red).

we ran our job on the SURFSara cluster on data between September 20th and 30th 2016 it took the shuffle phase more than 5 hours to finish successfully. We cherry-picked the results and looked at how the algorithm performed for the pages of Hillary Clinton and Donald Trump (we knew they were spiking right after September 26th from our previous experiment). In Figure 8 we have plotted with blue the number of views for Hillary Clinton's page and with orange the value we obtained for η over the timeseries by using the data driven algorithm. We will leave out the red signal for now. Looking at this figure we tried to find a sensible value for θ. Remember that if $\eta >= \theta$ (a fixed threshold) the input signal is found to be trending (and non-trending otherwise). An example value for θ would be 1, for example - this would mean that if η is greater than or equal to 1 the Wikipedia page is trending or presents a spike and as soon as it drops below 1, the page views would revert to their normal values. The problem is that there is actually no good value to set θ to, as the interval does not correctly capture the spike in the image and the small trending portion after the spike. If we set a value larger than or equal to 1 for θ we do not capture the spike. If we set a value lower than 1 for θ we capture too much of the signal after the spike. This is not what we want - we would like to be able to capture only the spike and the small trending portion after it. We looked at the results for some other Wikipedia pages generated by this run and we could see that η exhibited the same behavior.

We can identify two problems in our approach so far: firstly, it took too long to run our pipeline, due to the amount of time the jobs were spending in the shuffle phase; secondly, the data driven algorithm was not yielding the results we were hoping to obtain.

To solve our first problem we decided to only take into consideration the pages in English and Spanish. This means that from a total of 40 million pages that amounted to 500GB of compressed page views statistics from August 2015 to September 2016, we remained with only 82GB of data after the filtering stage which amounted to 5.2 million pages in English and 1.3 million pages in Spanish. This means that we have reduced the number of keys we were performing the reduction on to 16.25% of the initial number. However, if we look at language statistics for the United States[26] we see that 92.4% of the people speak English or Spanish at home so it does make sense to only look at these truncated statistics. After the filtering the pipeline performed very well - it managed to process **all** the data we needed (so from August 2015 to September 2016) in only 2 hours.

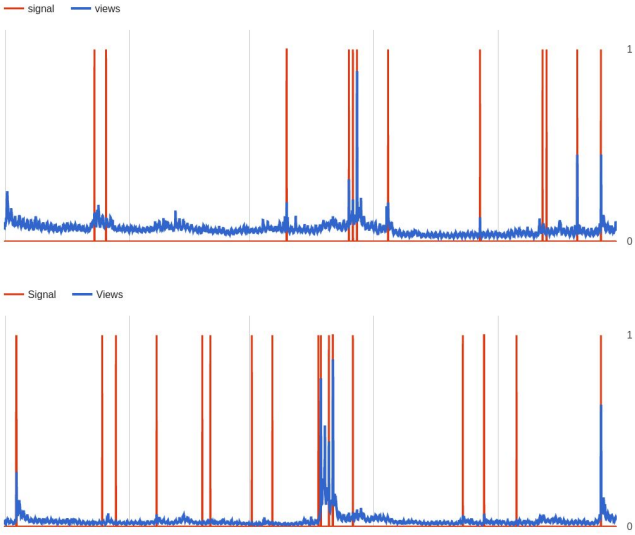


Figure 9: Experimenting on SURFSara. Number of views on Barack Obama's (up) Donald Trump's (down) Wikipedia page (blue) and the binary signal (red), between May and September 2016.

Our second problem was that the data driven algorithm did not perform as we were hoping it to on the Wikipedia data set. We think that the problem is not with the algorithm, but rather with the reference sets that we have used. As we have previously mentioned we used the reference sets published in the Github repository of the Gnip Trend Detector[27]. These had been extracted from Twitter and were bucketed into 150 chunks of one hour. From what we have understood from visualizing stats for a number of Wikipedia pages, a Wikipedia page usually spikes, rather than becoming trending (so there is rarely one long period of a high number of views, but rather a sudden short period of a very high number of views) and this happens over the course of a few (3-5) hours. We searched online for reference sets for Wikipedia articles, but we could not find anything available.

This is when we decided to switch to the non-parametric algorithm we found on Stackoverflow. We ran the pipeline over the same data from between September 20th and 30th 2016. In Figure 8 we plotted with red the binary signal output by this algorithm. As long as the page is not trending or spiking, the signal is 0 and, as soon as the page presents a local spike the signal becomes 1. It is clear from the image that the signal captures the spike and the short trending period very well.

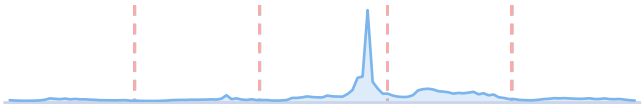
If we look at other runs and pages we see that the algorithm really does perform well on the Wikipedia data set. For example, in Figure 9 we see the views and signal for Barack Obama's (up) and Donald Trump's (down) page between May and September 2016. The signal becomes and stays 1 during every spiking period we can see in the timeseries.

We conclude that the non-parametric algorithm described in Section 2 is the algorithm we should use to identify trends and spikes in the Wikipedia page views timeseries.

[26]https://en.wikipedia.org/wiki/Languages_of_the_United_States [27]https://github.com/jeffakolb/Gnip-Trend-Detection

(a) Bernie Sanders

(b) Donald Trump



(c) John Kasich

Figure 10: Sparklines for the Wikipedia pages of Bernie Sanders (Democrat), Donald Trump and John Kasich (Republicans) before, during and after two election events that have a common date: Democrat semi-closed primary in New Hampshire, Republicans binding primary in New Hampshire (February 2nd 2016)

(a) Hillary Clinton

(b) Donald Trump

Figure 11: Sparklines for the Wikipedia pages of Hillary Clinton (Democrat) and Donald Trump (Republican) before, during and after two election events that have a common date: Democrats closed primary in New York, Republicans binding primary in New York (April 19th 2016)

We move on to determining whether Wikipedia pages related to an election event exhibit an anomaly in their number of views shortly before, during or after that particular election event. For this, we have put together an HTML + JavaScript visualization. There is a total number of 138 election events from between August 2015 and September 2016, but there are really 52 **distinct** calendar dates for these events. The *Trend Extractor Pipeline* builds the JSON output based on these 52 dates.

In Figures 10 and 11 we see the sparklines of the Wikipedia pages of five politicians, members of the Republican or Democratic party. The sparklines are divided into five different sections by red dotted lines - these sections correspond to the portion of the sparkline that shows views 48 and 24 hours prior to the primary elections listed in the description of the figures, on the day of the elections and finally 24 and 48 hours after the elections. In these figures we see that the pages of the various politicians spike almost at the end of the election day and not before (well, the exception is Donald Trump's page, but from what we have seen from other various primary elections this is just an exception). As the pages do not present a spike before the elections, we could not, based on a spiking factor, predict the elections.

Figure 12 shows how the views for the pages of Donald Trump and Hillary Clinton again spike right at the end of the day in which the first presidential debate between the two took place. In the case of the debates we were interested to determine whether Wikipedia is used as a fact checker or information source for the audience. The sparklines in the visualization show that it is mostly the pages of the politicians participating in the debate that exhibit a spike, and not other pages related to the debated topics. Furthermore, the pages that spike do so right after and not during the debates.

Wikipedia remains an information source for the viewers, but perhaps it is not a fact checker used during debates.

(a) Hillary Clinton

(b) Donald Trump

Figure 12: Sparklines for the Wikipedia pages of Hillary Clinton (Democrat) and Donald Trump (Republican) before, during and after the first Presidential Debate that took place in New York on September 26th 2016

6. CONCLUSIONS

In our experiments we tried to determine which Wikipedia pages related to the US election events between August 2015 and September 2016 present an increase before, during or after these events - in this case we would say the page becomes trending or exhibits a spike. We were interested in answering this question because we wanted to see whether the activity on these Wikipedia pages could help us predict the outcome of elections.

As we had to work with a relatively big data set (500GB of data), too large for our local machines to handle, we wrote Spark jobs to help us gain insight into the data. Spark proved to be a reliable and easy to use tool, unless it had to perform a reduction on a very large number of keys (40 million in our case). The workaround for this problem was to significantly reduce the number of keys fed into the shuffle stage. The language distribution of the Wikipedia pages was helpful in this case.

To find trends and spikes in the timeseries we first tried to use a data driven non-parametric algorithm that classified the timeseries based on previous knowledge. Unfortunately, this approach did not work well for our data set, primarily because we could not find reference sets for Wikipedia pages and had to use Twitter trending and non-trending topic examples. A parametric algorithm proved to work very well on the Wikipedia data set and helped us correctly identify spikes and trends in the timeseries. The disadvantage of the parametric algorithm was that we had to estimate the input parameters of the algorithm. One advantage was that this algorithm worked directly on the raw timeseries, so no smoothing was needed.

We have created a visualization in the form of a static web page to help the reader browse through the data output by our Spark jobs. In the page we show all the election events that have occurred between August 2015 and September 2016, as well as the pages that spike or become trending during each election event. We also plot various polls that show how the candidates fare compared to one another during these events.

The results we have obtained have shown us that Wikipedia is not, in fact, a reliable polling source. During the primary elections the pages of various politicians spiked right after the event and not before, so we could not use this information to determine whether a certain person will win or lose. Furthermore, during debates, Wikipedia was not used as a fact checker. As in the case of the elections, the platform was used mostly after the debates and most of the pages that spike are again those of politicians', so not necessarily those related to discussed topics.

7. REFERENCES

[1] D. Gayo-Avello. A balanced survey on election prediction using twitter data. May 2012. https://arxiv.org/pdf/1204.6441v1.pdf.

[2] K. Goldstein. The bloomberg politics poll decoder. http://www.bloomberg.com/politics/graphics/2016-poll-decoder/, 2016.

[3] S. Hendrickson, J. Kolb, B. Lehman, and J. Montague. Trend detection in social data. Technical report, Twitter Inc., June 2015. https://github.com/jeffakolb/Gnip-Trend-Detection/blob/92d71c3460db1482dc5bb0e640cea2d4d725e5ec/paper/trends.pdf.

[4] S. Nikolov. Trend or No Trend: A Novel Nonparametric Method for Classifying Time Series. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, USA, 2012. https://dspace.mit.edu/bitstream/handle/1721.1/85399/870304955-MIT.pdf?sequence=2.