

Finding surprising patterns in textual data streams

Tristan Snowsill, Florent Nicart, Marco Stefani, Tijn De Bie, Nello Cristianini
Intelligent Systems Laboratory, University of Bristol
Merchant Venturers Building, Bristol, BS8 1UB, UK

Abstract—We address the task of detecting surprising patterns in large textual data streams. These can reveal events in the real world when the data streams are generated by online news media, emails, Twitter feeds, movie subtitles, scientific publications, and more. The volume of interest in such text streams often exceeds human capacity for analysis, such that automatic pattern recognition tools are indispensable.

In particular, we are interested in surprising changes in the frequency of n -grams of words, or more generally of symbols from an unlimited alphabet size. Despite the exponentially large number of possible n -grams in the size of the alphabet (which is itself unbounded), we show how these can be detected efficiently. To this end, we rely on a data structure known as a generalised suffix tree, which is additionally annotated with a limited amount of statistical information. Crucially, we show how the generalised suffix tree as well as these statistical annotations can efficiently be updated in an on-line fashion.

I. INTRODUCTION

We are interested in the problem of identifying events in the online mediasphere, *i.e.*, identifying news. This is a task of pattern discovery, not pattern matching, in that we are not asking whether a particular event *has* happened, but rather we want to know about all events which have happened.

We define Events as surprising patterns in a textual data stream (TDS), which correspond to a change in the Source of the data. TDSs are commonplace on the internet (e.g. RSS feeds, twitter, blogs, subtitles in movies...). Also large enterprises and government departments keep logs of all relevant activities or issues in the processes they monitor, or storing user requests or complaints. Scientific literature can also be regarded as a TDS.

The ubiquity of text streams makes them ideal targets for pattern analysis. In many cases, the throughputs of text streams are overwhelming, and well beyond human intelligibility. A thorough analysis of these text streams can provide important benefits, such as a better understanding of a consumer and their behaviour, an insight into the major events reported in the world news, real-time detection of faults in the information pipeline of a large company, or quick insight into the latest trends and discoveries in science and technology.

In this paper, we will model such data as streams of text generated by a stochastic Source. We then consider the problem of detecting changes in the Source generating these texts, to which we will refer as Events. Note we capitalize “Event” here, indicating that it is used in a formal meaning as defined in Section II, although in Section IV we will show that Events are often good markers of informal “events”.

A. Text stream mining

One of the paradigms of text stream mining (and data stream mining in general) is one of bounded time and space resources. Intending to break away from machine learning techniques which iterate over a large database during analysis, data stream miners aim to develop space efficient techniques which do not have to “look back” at earlier data on the stream [1]. This commonly translates to demanding that algorithms have space requirements linear or even sublinear in the length of text seen so far.

In addition there are bounds on the amount of time that can be spent processing the stream. Importantly an algorithm must be capable of “keeping up” with the data stream, which results in work per unit time sublinear (again, ideally constant) in the length of the text seen so far.

In this paper we combine methods of statistical and syntactical pattern recognition to create a system that can be used in the above setting, requiring only constant time and space per time frame.

a) Event detection: Event detection is an aspect of temporal text mining which has attracted substantial interest. It is broadly divided into *retrospective* and *on-line* event detection. Retrospective event detection involves processing a collection of timestamped text samples taken from a text stream in a batch. On-line event detection involves detecting events as they are received in real-time, and is a data stream mining problem. [2]

Our method lies between retrospective and on-line event detection as it processes in small batches. These batches can span a short length of time if the text stream has high throughput. We therefore consider our algorithm to be an on-line event detection algorithm.

b) Contributions: The method presented in this paper combines several of the key benefits of [3], [4], and adds some important new ones. In particular, it is a time- and space-efficient online method, while it considers all n -grams in its search for Events, and does this despite unbounded alphabet sizes with no constraints on the distribution of the alphabet symbols. We do not reduce our search space of n -grams by placing bounds on the length or frequency of n -grams of words (*cf.* [5]), or restricting our attention to a certain type of n -gram.

There are several notable papers on the subject of event detection and relatives, such as [6], [3], [5], [4], [7], [2]. However, none of them combines pattern recognition with statistical analysis for an exhaustive monitoring of the statistical properties of all possible n -grams.

B. Outline of the paper

In Section II we describe a method for change point analysis in time series data which we will use as a basis to detect Events in text streams. The method uses the p -value statistic from hypothesis testing to assess the significance of the frequency of an n -gram given its frequency in the past.

In Section III we describe how we use a *suffix tree* data structure to efficiently implement the event detection analysis. We also include pseudocode for our algorithm.

In Section IV we demonstrate the results of applying our method to a large text stream formed from mediasphere web feeds. We demonstrate the validity of our method by identifying calendar events selected *a priori* and demonstrate further results showing the ability of our algorithm to detect events in the media. In this section we also describe a method for clustering n -grams with significantly varying frequencies to identify patterns which mark events.

II. EVENT DETECTION

In this section we formally define Events, and we describe how hypothesis testing can be used to detect them. This involves the definition of a null hypothesis, and the choice of test statistics that can be used to quantify deviations from the null hypothesis. We will define those test statistics in terms of the frequency of n -grams.

We approach the detection task by analysing each n -gram individually and hence we will generally talk about testing whether a given n -gram has experienced a significant change in frequency, but this test is applied to all n -grams rather than being query driven.

Firstly we provide some definitions of terms that will be used throughout this section.

Definition (Textual Data Stream, TDS). A TDS \mathcal{S} , is a sequence of text samples, S_t , sorted and indexed by time, t , such that S_{t_0} precedes S_{t_1} if $t_0 < t_1$. At a given time, t , no text samples, S_u , with $u > t$ will have been observed.

Definition (Alphabet). The alphabet, Σ , can be chosen to match the context, for example, when processing DNA sequences the alphabet $\Sigma = \{A, C, G, T\}$ is generally sensible, but when processing English prose it makes more sense to define the alphabet as English words, *e.g.*, “the”, “President”, “announced”; we adopt this approach. Any alphabet can be mapped bijectively to a range of integers.

Definition (Timeframe). A timeframe is a continuous range of time of duration T , defined with a closed beginning and open end, *i.e.*, of the form $[t_0, t_0 + T)$. We index timeframes such that a timeframe index i has a corresponding timeframe $[t_0(i), t_0(i) + T)$ and $t_0(i + 1) = t_0(i) + T$.

We define the range of text samples which occur in a text stream during a timeframe i as

$$\begin{aligned} \mathcal{S}(i) &= \mathcal{S}[t_0(i), t_0(i) + T] \\ &= \{S_t \mid t_0(i) \leq t < t_0(i) + T\}. \end{aligned}$$

We denote the number of text samples in a timeframe as $n_i = |\mathcal{S}(i)|$ and the number of text samples containing an n -gram x as $k_i(x)$.

Definition (Source of a text stream). We define the Source of a text stream as the stochastic process that generates each of the text samples in the text stream.

Examples of a Source are: a Markov chain of order one, a higher order Markov chain, a variable length Markov chain, as well as more complicated models with long range interactions. Note that any given Source will give rise to a certain distribution for the length of a random text sample in the stream. Furthermore, given the Source, the probability that a random text sample in the stream contains a given n -gram is fixed (even though for complex models it may be hard to compute it analytically).

While we allow each text sample to be generated by a complex stochastic process (the Source), we assume the separate text samples to be independently sampled from this Source. In practice, however, the source may vary over time.

Definition (Event). We define an Event as a change in the Source of the text stream. We will detect it as a surprising change in the frequency of an n -gram, as discussed below.

For example, a change in the transition probabilities of a (higher order) Markov chain is an Event. An Event can have various consequences, such as the change in probability of a given n -gram appearing in a text sample generated by the Source.

A. Detecting Events using n -gram frequencies

We wish to detect when the Source of a text stream changes. We will formalise this as a hypothesis testing problem. In particular, each timeframe, we will carry out a hypothesis test, where the null hypothesis is that the Source has remained unchanged.

Null Hypothesis. *For each timeframe, we consider the null hypothesis that the Source generating the text samples in that timeframe is identical to the Source that generated the previous text samples in the stream.*

Under a null hypothesis, it is possible to make predictions about the distributions of certain random variables. For example if the Source that generated the previous text samples is known, one could in principle compute the distribution of the proportion of text samples in a timeframe that contain a given n -gram. We refer to this number of text samples as the n -gram’s frequency.

Given the observed text samples within a timeframe, we can contrast the observed frequency of an n -gram within that timeframe with the expected frequency under the null hypothesis. We could quantify this by computing the probability under the null hypothesis of observing a frequency for that n -gram equal to or larger than its observed frequency. This probability is known in statistics as the p -value, and the n -gram’s frequency is known as the test statistic. Obviously, we could have as

many such test statistics as the number of n -grams, and for each of these we could compute a p -value.

Given that the text samples are independently generated by the Source under the null hypothesis, the probability distribution for the frequency of a given n -gram is binomially distributed [8], *i.e.*, if $f(x)$ is the expected proportion of text samples which contain the n -gram x within the i^{th} timeframe containing n_i texts, and $k_i(x)$ is the number of text samples in that timeframe containing the n -gram x ,

$$k_i(x) \sim \text{Bin}(n_i; f(x)).$$

From now on we will assume we always refer to a given n -gram x and drop the parentheses for simplicity, *i.e.*, $k_i(x) = k_i$ and $f(x) = f$.

Of course, in practice the expected proportion of text samples containing a given n -gram under the null hypothesis is unknown. All that is specified under the null hypothesis is that it is the same as it was in the past timeframes. However, we could estimate it by computing the average proportion of texts containing the n -gram in the past timeframes, and we could use this as an approximation for the frequency f .

In this paper we adopt a variation of this approach, which estimates the frequency f as a *weighted* average of past observed frequencies, with an emphasis on the more recent past. This has the key advantage that it is more robust, allowing the estimation of f to readjust after a change of the Source. At each timeframe i we make an estimate of f , $f_\alpha^{(i)}$, by taking a weighted average over the past,

$$f_\alpha^{(i)} = (1 - \alpha) \sum_{j=-\infty}^{i-1} \alpha^{i-1-j} \frac{k_j}{n_j}.$$

This estimator contains a weighting parameter, α , which determines the “memory” of the estimator. Low α values mean that recent measurements are given greater precedence while high α values tend toward an unweighted average. The estimator can be calculated iteratively,

$$f_\alpha^{(i)} = \alpha f_\alpha^{(i-1)} + (1 - \alpha) \frac{k_{i-1}}{n_{i-1}}.$$

In summary, for each n -gram, we estimate f with a weighted average, $f_\alpha^{(i)}$, of the n -gram’s observed frequencies. Given this, we can find a p -value for observing the n -gram in k_i of the n_i text samples in timeframe i , *i.e.*, the probability of observing at least as many text samples containing the n -gram as observed in timeframe i . If we let $X_i \sim \text{Bin}(n_i; f_\alpha^{(i)})$ be a random variable distributed as we expect the n -gram’s frequency to be distributed under the null hypothesis, we can calculate the p -value as

$$\begin{aligned} p &= \Pr(X_i \geq k_i) \\ &= \sum_{k=k_i}^{n_i} \binom{n_i}{k} \left(f_\alpha^{(i)}\right)^k \left(1 - f_\alpha^{(i)}\right)^{n_i-k}. \end{aligned}$$

This p -value tells us how likely it is that an observation has occurred due to chance (based on our null hypothesis). A very

low p -value indicates that an observation is very unlikely given our hypothesis, so the frequency of the n -gram has very likely increased by a significant amount. This change in frequency is a change in the Source of the text samples so must be an Event.

It should be pointed out that monitoring all n -grams is sufficient to detect changes in any Markov Source of unspecified order. Indeed, any change in the Markov transition probabilities will result in a change in the expected n -gram frequencies.

Pseudo-frequency parameter: In text, the alphabet size (*i.e.*, the number of words) is typically large, with many words and hence n -grams having an extremely small frequency. In addition to this new words are invented or old words develop new meanings and typographic errors are not uncommon.

For these n -grams which have not been seen before we would have a frequency estimator of 0. This presents a problem in our p -value computation as when $X_i \sim \text{Bin}(n_i; 0)$, $\Pr(X > 0) = 0$, *i.e.*, we attach infinite significance to the appearance as we believed it was a statistical impossibility.

An intermediate solution to solve this problem is to add a *pseudo-count* to our estimator,

$$f_\alpha^{(i)} = \alpha f_\alpha^{(i-1)} + (1 - \alpha) \frac{k_{i-1} + 1}{n_{i-1} + 1}.$$

This implies that at each timeframe we pretend we have seen the n -gram once. One difficulty with this definition is that there is no simple formula for f_i given $k_j = 0$ for all $j < i$. Alternatively we can define a *pseudo-frequency* parameter, ν , which allows us to specify what relative frequency we would expect, giving our estimator as

$$f_\alpha^{(i)} = \alpha f_\alpha^{(i-1)} + (1 - \alpha) \frac{k_{i-1} + n_{i-1}\nu}{n_{i-1}(1 + \nu)}.$$

For n -grams not yet seen, *i.e.*, $\forall i, k_i = 0, n_i > 0$

$$\lim_{i \rightarrow \infty} f_\alpha^{(i)} = \frac{\nu}{1 + \nu}.$$

In a Bayesian context, this can be interpreted as estimating the frequencies $f_\alpha^{(i)}$ as the posterior mean estimate with a Dirichlet prior.

What is significant?: To decide which p -values should be considered significant, we could set a threshold on the p -value below which they are considered significant. However, multiple testing issues severely complicate this matter. Indeed, each n -gram corresponds to a test of the same null hypothesis, and all these tests are dependent on each other. In the determination of a suitable p -value threshold we would have to quantify the resulting multiple testing effects, which is hard in this general setting. Only making further assumptions on the null hypothesis, which we would like to avoid, could make this feasible.

Alternatively we could content ourselves with simply ranking the n -grams in order of increasing p -value, and considering, for example, a fixed number N of smallest p -values. This approach has the practical advantage that post-processing on the results should take the same length of time.

B. Implementation requirements

An implementation of this event detection method must be able to store a weighted average of the frequency of each n -gram and retrieve it efficiently. This is a constant space requirement per n -gram but the number of possible n -grams is exponential in the alphabet size. When we consider that the alphabet can be unbounded, any trivial storage system will clearly be incapable of implementing our method. In the next section we introduce the suffix tree, a data structure which enables us to efficiently implement our method.

III. ALGORITHM

In this section we describe our algorithm which uses a data structure called a *suffix tree* to efficiently store the frequencies for all n -grams observed in the stream and thereby implement the statistical test described in Section II. We provide pseudocode for our algorithm to explain how the suffix tree is used and how we perform necessary calculations *just-in-time* to avoid having to perform a full traversal of the suffix tree at each step.

Finally we demonstrate how we can take advantage of the properties of our method to produce a constant space version of our algorithm.

A *suffix tree* is a data structure that indexes all the substrings of a string; a *generalised suffix tree* is a similar data structure that indexes all the substrings of a set of strings [9]. They are extremely efficient data structures, as they can be constructed using linear time and space [10], [11], and their availability allows a number of apparently complex operations to be carried out with remarkable efficiency. Several algorithms exist to construct the suffix tree of a string in linear time with respect to its length [9], [11], [10]. Our method is based on the on-line Ukkonen algorithm [11] which, in addition to being a linear time algorithm, allows one to add sequences into an existing suffix tree. There is no space to go in greater detail within the constraints of this paper. However, it is sufficient to know that every n -gram is indexed by associating it to a node in the suffix tree, or to an implicit node partway down an edge. Importantly, incrementally updating a suffix tree with more data can be done in a time that is linear in the amount of data added.

Our method combines syntactical and statistical pattern recognition by adding statistical annotation to the nodes of a suffix tree. In this way, we can efficiently perform statistical testing on every possible n -gram at once. The result is a new data structure for the discovery of surprising patterns in TDSs.

We augment this representation by further annotating the nodes with statistical information that can be used to detect significant changes in the frequency of the corresponding n -gram.

Our algorithm works as shown in Algorithm 1. We let N_i be the *total length of text* in the timeframe i such that $\hat{m}_i = \frac{N_i}{n_i}$ is the average length of text samples in the timeframe i . We let $N_{\dots i-1}$ denote the total length of text observed before the timeframe i .

Algorithm 1 Algorithm overview

- 1: Receive the text samples from timeframe i from the text stream (the total length of these is N_i)
 - 2: Add these text samples to the suffix tree (takes $\mathcal{O}(N_i)$ time [11])
 - 3: Assign frequency estimators to all new nodes in the suffix tree
 - 4: Run Algorithm 2 to calculate the p -values of all n -grams and update the frequency estimators
-

The frequency estimators for most of the n -grams are irrelevant at any one time, namely when their frequency is zero such that their p -value would be 1 (*i.e.*, completely insignificant). By storing an extra piece of information at each node – the timeframe during which the n -gram was last observed – we can calculate our frequency estimator *just-in-time*,

$$f_{\alpha}^{(i)} = \alpha^t f_{\alpha}^{(i-t)} + (1 - \alpha^t) \frac{\nu}{1 + \nu},$$

where t is the number of timeframes since the estimator was last updated.

Algorithm 2 Statistical computation algorithm (run after all text samples from a timeframe have been added to the suffix tree)

- 1: Initialise a mapping, `count`, from node ID to k_i (all node IDs initially map to zero)
 - 2: **for each** text sample in the timeframe **do**
 - 3: Calculate the set of nodes visited by the text sample and all its suffixes
 - 4: **for each** node in the set **do**
 - 5: Increment `count[node]`
 - 6: **end for**
 - 7: **end for**
 - 8: **for each** node such that `count[node] ≠ 0` **do**
 - 9: **if** timestamp of node older than timeframe $i - 1$ **then**
 - 10: Calculate f_i for node
 - 11: **end if**
 - 12: Calculate p -value for node
 - 13: Update frequency estimator for node to f_{i+1}
 - 14: Update timestamp of node to i
 - 15: **end for**
-

Analysing Algorithm 2 we note that Step 3 can be executed in $\mathcal{O}(N_i^2)$ worst case time (when $n_i = 1$), and in $\mathcal{O}(\hat{m}_i^2)$ time in the average case as we can trace each suffix of the text samples from the root in time at most linear in the length of the suffix.

Step 4 takes worst case $\mathcal{O}(N_i T(\text{increment}))$ time and average case $\mathcal{O}(\hat{m}_i T(\text{increment}))$, where $T(\text{increment})$ is the time taken to increment `count[node]`. Using a balanced binary tree to store `count` this can be achieved with worst case time complexity $\mathcal{O}(N_i \log N_i)$. Overall Step 2 is run n_i

times, therefore its overall worst case time complexity is

$$\mathcal{O}(n_i N_i^2 + n_i N_i \log N_i).$$

Step 8 is run at most $\mathcal{O}(N_i)$ times and each run takes constant time. In total the worst case time complexity is $\mathcal{O}(n_i N_i^2)$.

We can see that the worst case time complexity of our algorithm is independent of $N_{\dots i-1}$ (the total length of text observed before the timeframe i), and is only dependent on the total length of text observed during the timeframe i .

Result. We can calculate the p -value of frequency changes of all n -grams in the most recent timeframe in time which does not increase as more of the text stream is observed. When the rate of the text stream is constant these calculations can clearly be done in constant time.

IV. EXPERIMENTAL RESULTS

In this section we show experiments we have performed to test our method. We first describe the text stream on which we test the method, then show how we extract meaningful results and present our results.

A. Description of the text stream

We tested our method on a TDS found by scraping all articles syndicated by 2471 news feeds. The total number of articles scraped is 1,627,340, and the alphabet size (the number of different words observed) is 1,209,258. The suffix tree indexing this data, along with the statistical annotations, takes up 30GB disk space.

B. Processing results

The statistical computation produces a set of p -values for nodes corresponding to n -grams seen in the most recent timeframe (the p -value of other nodes simply being 1). We then sort this set by p -value in ascending order to obtain a listing of the most significant n -grams. We make two observations about this listing of n -grams.

Observation 1. The most significant n -grams on a day are often related to the date itself as many articles include the date in some form or another (see Section IV-D).

Observation 2. There is a great deal of redundancy in the results as the substrings of significant n -grams are also likely to be statistically significant.

Observation 1 is a helpful sanity check, it verifies that the algorithm works as it should do. In response we filter out references to the date. The redundancy issue highlighted in Observation 2 can be tackled by performing clustering on the list.

C. “Feature vector”-based clustering

Our method of clustering is based on the bag of words representation of n -grams with TF-IDF weighting [12].

We measure the proximity of clusters by using the angle between the cluster centroids. The cluster centroids are defined as the means of the feature vectors of n -grams in the clusters.

TABLE I

WE SANITY-CHECKED OUR APPROACH BY VERIFYING THAT IT CAN DETECT SPECIFIC EVENTS THAT ARE KNOWN TO BE IN THE NEWS STREAM, SUCH AS PUBLIC HOLIDAYS AND DAYS OF THE WEEK. THE RESULTS FOR PUBLIC HOLIDAYS ARE SHOWN IN THIS TABLE.

Date	Event	Cluster ranking
2008-10-31	Halloween	1 st
2008-11-11	Veterans Day	1 st
2008-11-26	Thanksgiving	2 nd
2008-12-25	Christmas	1 st
2009-01-01	New Year	1 st

The angle, θ , between two feature vectors, \mathbf{a} and \mathbf{b} , can be found using the dot product,

$$\cos \theta = \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|}.$$

We cluster hierarchically in a bottom-up fashion, starting with a separate cluster for each n -gram and then sequentially merging pairs of clusters with the smallest angles between their centroids. We continue until all clusters are orthogonal.

In order to represent the clusters in a way that is succinct and which best encapsulates the essence of the n -grams in the cluster we find a representative n -gram from the cluster. We select the representative of each cluster by finding the n -gram which is closest to the cluster centroid. Proximity is calculated using the angle between the feature vectors as before.

D. Validation results

The media system exhibits behaviour which can be used to validate our method. For example, the date is very often included in the full content of an article, and named calendar dates (e.g., Christmas, Halloween, Thanksgiving) are also often mentioned in the news on the appropriate days.

Using our method with $\{\alpha = 0.518, \nu = 0.001\}$ we found that the name of the day appears in n -grams ranked in the top five (before any post-processing) in over 85% of days and in the top twenty in over 95% of days.

Using the same parameters and after post-processing calendar events were detected as shown in Table I.

E. Further results

Using the same experimental parameters as before we ran our method from mid-August 2008 to January 2009. Table II shows some noteworthy results, corresponding to actual events found in the news in the time period of the analysis. Figure 1 shows the timeline over 5 days of the frequency, the expected frequency, and the p -value, for one noteworthy n -gram.

V. CONCLUSION

We have presented an on-line method for detecting Events in text streams by identifying statistically significant increases in the frequency of n -grams within the stream. The computation time for this detection is independent of the amount of text observed in the past. By taking weighted averages of frequencies our method also addresses the problem of concept drift (where slow changes occur in the setting of the problem).

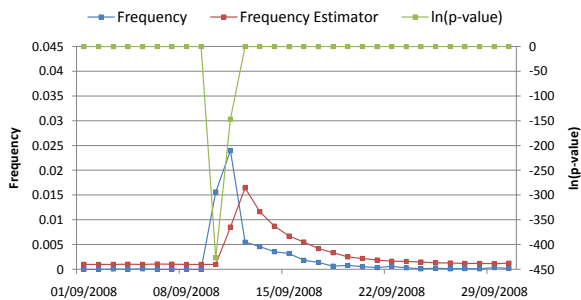


Fig. 1. This plot shows the frequency of the n -gram ‘lipstick on a pig’ over the last 5 days of September 2008, its estimated frequency in the null model, and the logarithm of the p -value. It can be seen that the p -value peaks on the first day the frequency of ‘lipstick on a pig’ increases drastically. After that, the estimated frequency increases, and the level of surprise and hence the p -value gradually return to normal. Doing this exhaustively for all n -grams in the news allows us to find surprising patterns efficiently and reliably.

TABLE II
HIGHLIGHTS FROM OUR EVENT DETECTION ALGORITHM

Date	Top five Event clusters
2008-08-26	“Michelle Obama”, “Gustav”, “the Democratic National Convention”, “Ted Kennedy”, “speech”
2008-09-10	“Large Hadron Collider (LHC)”, “lipstick on a pig”, “leader Kim Jong Il”, “beam of protons”, “Big Bang”
2008-10-03	“Palin and Joe Biden”, “the vice presidential debate”, “folksy”, “maverick”, “white flag of surrender”
2008-11-05	“Barack Obama’s victory”, “precincts reporting”, “Grant Park”, “defeated Republican”, “elected”
2008-11-26	“Suvarnabhumi Airport”, “Thanksgiving”, “Bangkok”, “main international airport”, “Thai”
2008-11-27	“attacks in Mumbai”, “the Taj Mahal hotel”, “Oberoi”, “hotels”, “gunmen”
2008-12-28	“ Hamas-ruled Gaza”, “Israel”, “Palestinian officials”, “Israeli warplanes”, “suicide bomber”

We demonstrated the power of our method by applying it to a text stream formed as an amalgamation of a number of news articles and validating it by identifying calendar Events. We also showed further results of our method, showing it identifying Hurricane Gustav, the Large Hadron Collider experiment, numerous political events surrounding the 2008 U.S. elections and violence in India and the Gaza strip.

In the future we aim to investigate improving our clustering by clustering n -grams which appear in similar sets of documents and to compare frequencies over different time spans to allow for identification of slower Events (e.g., the rising prevalence of eco-terms such as “carbon footprint”).

We will also investigate filtering n -grams which appear mostly in a single outlet (likely to be spam or disclaimers or signatures added to all articles).

The statistically annotated suffix tree which we produce in our algorithm also has many other uses. We will investigate the use of the suffix tree for creating language models and detecting long repeated substrings, which could tell us about how information is propagated through the mediasphere in the digital age where replication is precise and inexpensive.

ACKNOWLEDGEMENTS

This work is partially supported by EPSRC project EP/G056447/1, the European Commission through the PAS-CAL2 Network of Excellence (FP7-216866) and the IST project SMART (FP6-033917). Nello Cristianini is supported by a Royal Society Wolfson Merit Award.

REFERENCES

- [1] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, “Mining data streams: a review,” *SIGMOD Rec.*, vol. 34, no. 2, pp. 18–26, 2005.
- [2] Y. Yang, T. Pierce, and J. Carbonell, “A study of retrospective and on-line event detection,” in *SIGIR ’98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM, 1998, pp. 28–36.
- [3] R. Swan and J. Allan, “Extracting significant time varying features from text,” in *CIKM ’99: Proceedings of the eighth international conference on Information and knowledge management*. New York, NY, USA: ACM, 1999, pp. 38–45.
- [4] A. Apostolico, F.-C. Gong, and S. Lonardi, “Verbunculus and the discovery of unusual words,” *J. Comput. Sci. Technol.*, vol. 19, no. 1, pp. 22–41, 2004.
- [5] J. Leskovec, L. Backstrom, and J. Kleinberg, “Meme-tracking and the dynamics of the news cycle,” in *KDD ’09*. New York, NY, USA: ACM, 2009.
- [6] J. Allan, R. Papka, and V. Lavrenko, “On-line new event detection and tracking,” in *In Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1998, pp. 37–45.
- [7] E. Keogh, S. Lonardi, and B. Y.-C. Chiu, “Finding surprising patterns in a time series database in linear time and space,” in *KDD ’02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2002, pp. 550–556.
- [8] G. P. C. Fung, J. X. Yu, P. S. Yu, and H. Lu, “Parameter free bursty events detection in text streams,” in *VLDB ’05: Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment, 2005, pp. 181–192.
- [9] P. Weiner, “Linear pattern matching algorithms,” in *Proceedings of the 14th IEEE Symposium on Switching and Automata Theory*, 1973, pp. 1–11.
- [10] E. M. McCreight, “A space-economical suffix tree construction algorithm,” *J. ACM*, vol. 23, no. 2, pp. 262–272, 1976.
- [11] E. Ukkonen, “Online construction of suffix trees,” *Algorithmica*, vol. 14, no. 3, pp. 249–260, 1995.
- [12] G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” *Information Processing and Management*, vol. 24, no. 5, pp. 513–523, 1988.