Spatial - Temporal Random Indexing for Event Detection in Newswire Data

Rohit Menon
rmenon@cs.stonybrook.edu
107710848

Ankush Gulati
angulati@cs.stonybrook.edu
107569466

Overview:

The rapid increase in the amount of newswire stories stored on electronic devices and the internet raises new challenges for information retrieval. Use of query based tools is not suitable for generic searches. There is a need for an intelligent system capable of automatically locating events in continuous stream of newswire data. This is called automatic event detection.

In this project, we present a novel approach for automatic event detection in newswire data, using spatial temporal random indexing. Our approach involves building a 4 dimensional vector space where each word in the corpus is uniquely represented using a large sparse vector, unique in time and location. Next we analyze the spatially and temporally annotated space to determine significant semantic shifts over time and space. Finally we demonstrate, how the shift generated can be used to predict the events that occur.

Keywords:

Random Indexing, Spatial Temporal annotated Semantic space, Crawler, Semantic shift.

Central Idea

The central idea of the paper can be summarized by the following steps:

- Collect newswire data on topics of interest that span over different locations around the globe and also over a time span of days to months to years. This has been done by developing a customized crawler capable of collecting desired data in correct form.
- Clean and verify the format of the dataset before processing it to generate the vector space.
- Develop a spatially and temporally annotated vector space, of four dimensions, by assigning vectors to each word. For a set of interesting words, we calculate the semantics with at different timestamps at different locations.
- Compare the vectors in different location or time slices for same location, and calculate the semantic shifts. Use standard deviation and mean values to indicate significant semantic shifts.
- Trace back the shift to documents and manually determine the reason for the shift.

The project has been implemented considering numerous performance issues. Also experimental results with changes in vector sizes, window size, the density of vector values w.r.t. dimensions have been presented.

Basic Terminology:

- **Random Indexing**
  Random Indexing is a vector space technique used for approximation as proposed by Kanerva et al. (2000) as an alternative for Single Valued decomposition for Latent Semantic Analysis. While the SVD results in significant improvements in information retrieval, it is highly inefficient and requires the generation of the entire co -occurence matrix and its decomposition before any measurements can be made.

  Random Indexing avoids these by creating a short vector for each context, and producing the context vector for each term by summing index vectors for each context as it reads and thus helps in building the semantic space incrementally.

  A **d** length index vector is allocated to each unique context as it is found. These vectors are sparse and hence can contain mostly zeros and few +/- 1 (**∈**). Each element in allocated one of these values with the probability:

Value	Probability
+1	$\frac{\epsilon/2}{d}$
0	$d-\epsilon/d$
-1	$\frac{\epsilon/2}{d}$

  These index vectors are generated on the fly as more and more corpus is read. Each new context generates a new index vector. The index vector is sum of all the contexts in which the index vector occurs.

  The context vector for a particular term t present in context C_1 = [1,0,0,-1] and C_2 = [0,1,0,-1] would be [1,1,0,-2]. Now if suppose the context one occurs again, then instead of creating a new index vector, we would just do a summation to get the result as [2,1,0,-3]. Random Indexing provides support for incremental sampling. Thus context can be added on the fly.

- **Semantic Space**
  Semantic space represents a word's semantics mapped to high dimensional vectors in geometric space. The semantics of a word **w** is defined as follows:

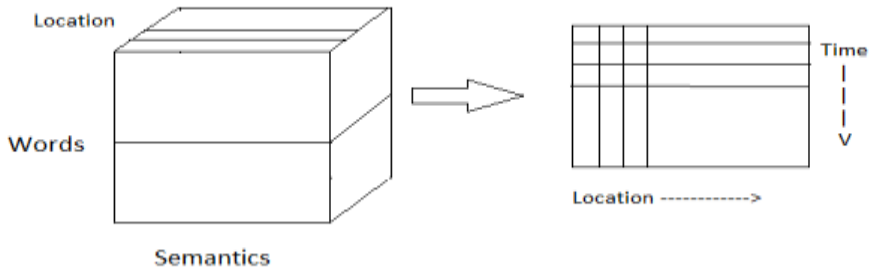
$$semantics(w) = \sum_{\forall c \in D} \sum_{-n \leq i \leq n} index(w_i)$$

  Here w is the focus word, w_i be a co-occurring word at distance i and index(w_i) be its index vector. Now n represents the window size over which co - occurring words are chosen.

  The dimensions in space represent the difference in meaning between the words, thus words with similar semantics will have similar vector representation.

- **Spatial Temporal Annotated Semantic Space:**
  Adding space and time dimensions to the semantic space model could be an effective way of tracking changes in semantics of words over space and time. There are three ways these could be implemented
  a) A separate semantic space for each location and each time.
  b) A separate semantic space for each location. But each location space will contain all the time spans in the same space.
  c) A single combined semantic space containing each location and each time span within the location.

  The drawback of choosing the first two designs are that when multiple semantic spaces are used, there is no guarantee that specific semantic meaning associated with some dimension *i* will be same in another space.



Tensor Representation of Semantic Space

- **Spatial Temporal Random Indexing**

  Spatial Temporal Random Indexing incorporates time and space into a single semantic space. Thus instead of using just word X Semantic Vector. It now uses Word X Semantic Vector X Time Tension X Location Tensor. This has been effectively illustrated in the above diagram.

  This design has numerous advantages, mentioned as follows :

  a) As the semantic space is created incrementally, we can add any number of documents and the space would accommodate the change.
  b) Next it is possible to do a comparison of words over arbitrary time spans over same location. As well as compare shifts over different locations over different times.
  c) It has greatly reduced time and memory requirement as compared to using SVD on the same sized corpus.

  Input Documents : $D = (\{(L_0, T_0, D_0), (L_0, T_1, D_1), \ldots\ldots\ldots\ldots, (L_0, T_k, D_k)\}$
  $\{(L_1, T_0, D_0), (L_1, T_1, D_1), \ldots\ldots\ldots\ldots, (L_1, T_k, D_k)\}$

  $\{(L_j, T_0, D_0), (L_j, T_1, D_1), \ldots\ldots\ldots\ldots, (L_j, T_k, D_k)\})$

  Where L_j - Is the jth location where the corpus exists, and D_i is the set of documents occurring at time T_i.

  $$Semantics(w,t,l) = \sum_{Dj \in Lj} \sum_{Ct \in Dj} \sum_{-n \le i \le n} index(w_i)$$

  Where Dj is all documents belonging to location Lj and Ct are all documents in Dj with timestamps t. The slice can be defined as follows :

  $$Slice(w) = \{L_j, Semantics\ of\ (w,t,l) | D_i \in L, w \in D_i, i = 1, k\}$$

  Using the above slice formula, we can compute the semantic shift over different time ranges as well as over different locations.

  Also it is possible to compare interesting words over locations using the above formula by using the semantic shifts for same time span. The final semantic shifts can be tracked back to the documents to determine the reason for the shift.

Baseline for Project:

The baseline model for the project is based on the idea of tf - idf (term frequency - inverse document frequency). This is often used in information retrieval and text mining. It basically represents a weight that indicates how important the word is to the document.

Term count = $$\frac{No\ of\ occurrences\ of\ term}{Total\ Number\ of\ terms\ in\ the\ document}$$

If there is a certain term T that is highly popular and appears a large number of times N in the corpus. But suppose in latter time span, the number reduces to k << N, then the term T is surely going to undergo a semantic shift.

Thus, change in term frequency over time and location can be used as a very simple way of determining semantic shift.

The results of Baseline using Term Frequency ratio for dataset of " Satyam" with
Location = Bangalore, India
TimeSpan = 6months
Interesting Words : mahindra, india, fraud, satyam
Start Time : Dec 2007

Words \ Timespan	Dec-07	Jun-08	Dec-08	Jun-09	Dec-09	Jun-10	Dec-10
mahindra	0	0	0.0018	0.2060	0.2155	0	0.0236
India	0.0123	0.0678	0.0144	0.5472	0.4977	0	0.8229
fraud	0	0	0.0047	0.1993	0.1867	0	0.6770
satyam	0.0067	0.0211	0.0420	0.4391	0.2672	0	0.6145

The baseline is stored in a 4 dimensional hash map structure.
{Interesting word, Location, Time Span -> Term frequency Ratio}

Thus it has the same structure as the resultant matrix, to store the baseline values. The baseline values provide a threshold and also an simple analysis as to which slots are going have zero shifts, as the word does not occur in the corpus.

Data Collection

We have collected the data from news agency websites like www.reuters.com. The dataset consists of articles classified on the basis of timestamp and location posted on the websites of news agencies like Reuters.

- **Data Set**

  We have compiled a dataset of more than 5000 articles on varied topics like apple inc, satyam technologies, congress, etc. over a period of more than four years across different geographical locations across the globe.

  To present a broader picture of our work and its features, we have tried to select topics that have undergone substantial change either over a period of time or if they vary a lot for different locations or in some cases show variation on the basis of both time and locations. For instance, the renowned Indian IT company Satyam Technologies went bankrupt in 2009 when a financial fraud came to light. The company was later acquired by Tech Mahindra and was renamed as Mahindra Satyam. We have analysed the semantic shift for Satyam in detail in our results section to provide a clear picture of our work and its key features.

- **Process**

  Because of the large volume of dataset we needed, the process of data collection was automated.

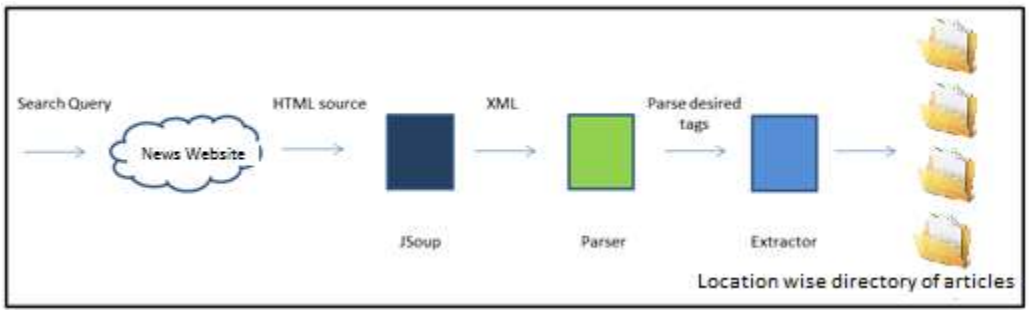


Figure: System Architecture of Crawler for Data Collection

We wrote a simple JAVA Crawler that crawls www.reuters.com and scraps all the articles related to a desired search query. It then writes all the searched articles into a directory named as per the search query in various files each representing a different geographical location. Each of the document contains multiple articles one per line sorted on the basis of their timestamps. For example, a document for location X would be X.txt and have the following format:

0122012200
<Article 1 Text>
0122022212
<Article 2 Text>
.

The information like geographical location and timestamp represent the place and time of article publish and are collected while crawling from the article page itself. There is a limitation both on side of parser and reuters.com that certain times only the recent news articles are displayed by search and hence only they are crawled.

Methodology:

The entire process can be broadly broken down into four major parts:

1. Creating the dataset
2. Generating random vectors
3. Reading & preprocessing the dataset.
4. Generation of semantic vector space.

1. Creating the dataset

To accumulate a large repository of articles, we automated the data collection process. We wrote a crawler that searches the desired news agency website and extracts the relevant articles for us in a specified format. We have implemented a website specific crawler for www.reuters.com that takes the input query and fetches all the articles related to the query from the website by parsing the DOM structure of the HTML page. The implementation of the web crawler is in JAVA and uses an open source library JSoup for converting the DOM Structure of the source of crawled html page to a clean XML that can then be read using simple JAVA routines. It takes generic parameters like search query, URL (reuters.com) & the result directory path and writes the relevant articles from the search result in the directory with the name of the search query in separate documents for different locations in a 'one article per line' format.
The system can be easily used and extended by others who need to generate large datasets of news articles for their research work.

We have written a java program Caller.java that works as a handler and runs the other three subsections one after the other. We have implemented the entire functionality using multithreading where in a thread is created for each document (i.e. per location) thus allowing faster computations for large datasets.

2. Generating random vectors

Every word in the corpus is assigned a random index vector. Each occurrence of the word, is assigned the same vector and a new vector is not generated each time. Each vector is made up of 10,000 dimensions in our implementation. Here each dimension is responsible for representing a different semantic for each word.

To generate a random index vector for each word in corpus, we perform the following steps:

a) Generate a random number between 4-6 which represents the window for sparseness of the vector. Let this be X.
b) Then generate X random numbers between 0 and 10,000. Which ever dimension has not been used for more than 5 times, by other words, we use those. Thus fill in X positions in 10,000 dimensions vector.
c) Now to fill in the value, we have an option of either 1 or -1. We generate a random number between 0 and 1000. If number is less than 500, we assign 1 else we assign -1.

We perform this entire process for every word in the corpus. These vectors are almost perfectly orthogonal as a dimension is not used for more than 5 times. Moreover using just 4-6 dimensions of the vector makes it highly sparse considering the size if 10,000 dimensions.

3. Reading & preprocessing the dataset

Once the dataset has been generated and stored in the specified format in flat files with the help of the crawler, we read the dataset from these files into a *{location, timestamp, article}* format hashmap. Similar to other semantic space approaches that used web-gathered data, before the corpus is used for performing event detection, the corpus is preprocessed to render it more uniform. While writing the articles into the hashmap we apply the following 9 preprocessing rules:

a) Remove all numbers
b) Remove all html mark-up and email addresses
c) Remove unusual punctuation, and separate all other punctuation from words
d) Remove stop words like the, has, it from article text for better precision.
e) Converting all words to lower case

f) Strip HTML tags (if any) from the text.
g) Remove email ids/ web addresses from the article text.
h) Discard articles with missing timestamps/location information.
i) Associate each entry with a numeric timestamp and a location.

This pre-processing allows the model to gracefully handle several irregularities in writing style, such as inconsistent use of punctuation and capitalization. Once the *{location, timestamp, article}* hashmap is generated, it is input into SemanticVectorGenerator.

4. Generation of semantic vector space

The semantic vector space is passed the Hashmap*{location, timestamp, article}* along with the timespan, interesting words list, and the window size. Now the structure of the semantic vector is space is as follows : Hashmap*{Interesting word, HashMap{Location, HashMap{Timespan, Vector}}}*. Thus the space represents the vector of each interesting word, at each location, at each timespan.

On receiving the input of HashMap*{location, timestamp, article},* we use multiple threads to process for each location. We analyze each interesting word for every location. For each location, first we determine which articles are in the same time span. Once we are determine that, then we go to every occurrence of one interesting word. Next using the window size we compute a sum of the vector along with the vectors of the neighboring words. We do this for every interesting word. We keep on doing till all documents for a location is not processed. Thus we get the resultant of Hashmap*{Interesting word, HashMap{Location, HashMap{Timespan, Vector}}}*
At the same time we also populate a structure called *bSpace* which stores the baseline values which are directly computed by using term frequency as explained above

Once the space is ready, now we need to calculate the shifts between vectors of the words in different time span. For determining semantic shift, we use cosine similarity. Cosine similarity indicates how similar two vectors are. Hence we use 1 - *(cosine similarity)* as a measure of semantic shift.

Thus we loop over the entire Semantic space called *stSpace*, and to determine the semantic shift for a word between different time spans. Once this is done, the entire results is stored in a resultant space called *rSpace* of the format Hashmap*{Interesting word, HashMap{Location, HashMap{Timespan, Semantic Shift}}}.* Finally using the result *rSpace*, we determine the standard deviation for each timespan.

Results and Evaluation:

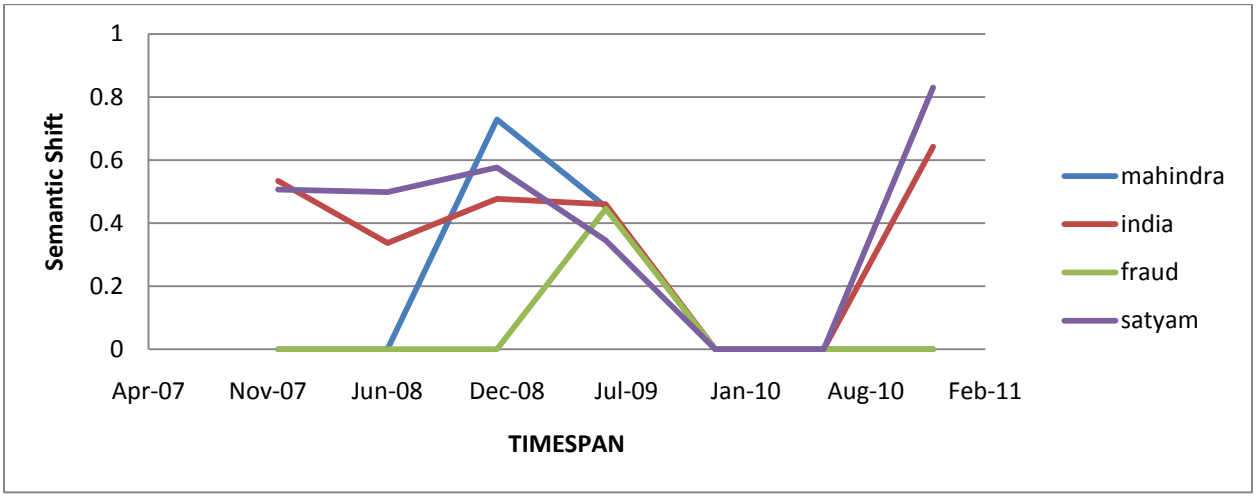
In this section, we present a detailed set of results and their analysis. Also numerous variations of the results can be observed.
Default window size :- 4, Default vector size: 10,000, Sparse :- 4-6

- Results of the algorithm when applied on dataset "Satyam" with a timespan of 6 months with Location : Bangalore.

Timespan / Interesting Word	Dec-07	Jun-08	Dec-08	Jun-09	Dec-09	Jun-10	Dec-10
Mahindra	0	0	**0.7284**	0.4515	0	0	0
India	**0.5342**	0.3367	0.4771	0.4593	0	0	0.6426
Fraud	0	0	0.4457	0.3457	0	0	0
Satyam	**0.5065**	**0.4978**	0.5765	0.3446	0	0	**0.8298**
Standard Deviation	0.4632	0.4128	0.6582	0.5154	0	0	0.66

a) Here the entire span of the time ranges from Dec 2007 to Dec 2010 with a steps of 6 months. The table presents the semantic shifts that every interesting word undergoes over every time span.
b) A value of 0 for semantic shift indicates the term does not occur in the timespan. Whereas the value of 1 indicates that there is a complete change in the semantics of the word. We observe that, as compared to the shifts determined from the baseline, these values are quite high and more accurate. This is because, baseline computation takes into consideration only the term frequency, while the implemented algorithm not only considers the frequency but also the neighboring words to incorporate the context of each word.
c) Standard deviation is used as a threshold factor to determine which shifts are significant and which shifts are not. The values in bold indicate the shifts that are greater than standard deviation for each time span.
d) Lets us consider each Bold value and hence **conclude on the effectiveness of the method for event detection -**
1) Mahindra took over Satyam on 14th April 2009. This lies in the time span of 6 months starting Dec 2008. Till then Mahindra was never associated with Satyam. Hence the values are 0 for previous spans. And then there is news about Mahindra take over Satyam in every newspaper. Hence the sudden semantic shift. Thus using the time span we could track back the reason for semantic shift.

2)Satyam fraud case by Ramalinga Raju came out in Jan 2008. Hence as you can see the result for fraud first shows a value in range Dec 2008 span of 6 months.
3) Before Dec 2008, we cannot find words other than Satyam to occur with high semantic shift. This was because during this time, Satyam was in news for different reasons ranging from the their quarterly results to Satyam being announced the 2008 winner of the coveted Golden Peacock Award for Corporate Governance under Risk Management and Compliance Issues.

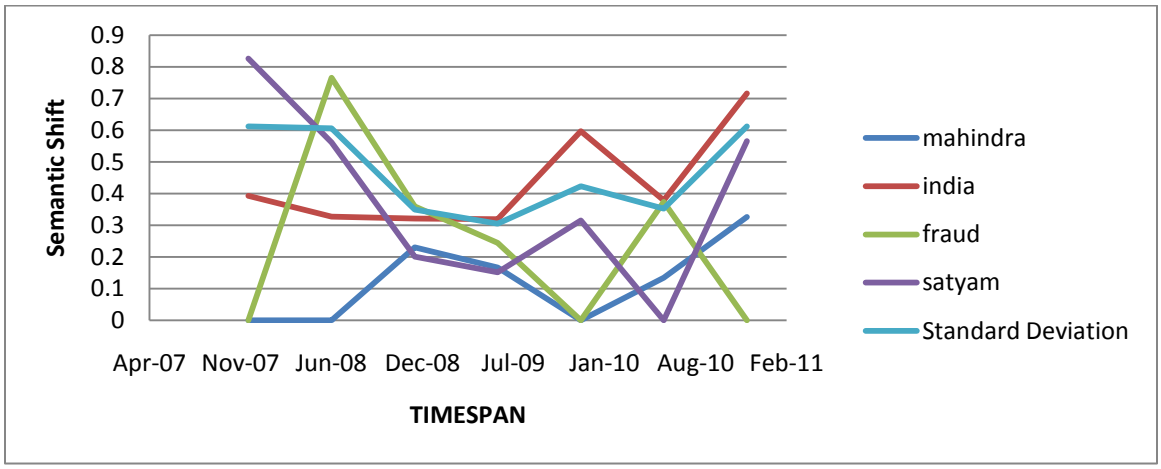


The above graph represents the same semantic shifts shown in the table. Thus from the analysis of the results we see that, we can effectively detect events by running Random Indexing.

- Results of the algorithm when applied on dataset "Satyam" with a timespan of 6 months with Location : Mumbai

Timespan\ Interesting Words	Dec-07	Jun-08	Dec-08	Jun-09	Dec-09	Jun-10	Dec-10
mahindra	0	0	0.2304	0.1669	0	0.1344	0.326
india	0.3923	0.3267	0.3212	**0.3189**	**0.5968**	**0.3789**	**0.7155**
fraud	0	**0.7649**	**0.3594**	0.2444	0	**0.3734**	0
satyam	**0.8261**	0.5613	0.2006	0.1511	0.315	0	0.5656
Standard Deviation	0.6118	0.6061	0.3497	0.3047	0.4228	0.3518	0.6118

The above results are from a new location Mumbai, but for the same time span. We see a considerable change in the results and the pattern. This is mainly because, Satyam has numerous offices in Bangalore, plus Bangalore is the IT Hub (Silicon Valley) of India. Whereas Mumbai is the financial center and hence the incident lost its impact with time and you can see that the word **India** has the most semantic shift in most timespans.



The above graph clearly shows a different trend over the previous graph for location of Bangalore.

- Results for algorithm applied on dataset of "Satyam" with a timespan of 1 year and location of Bangalore

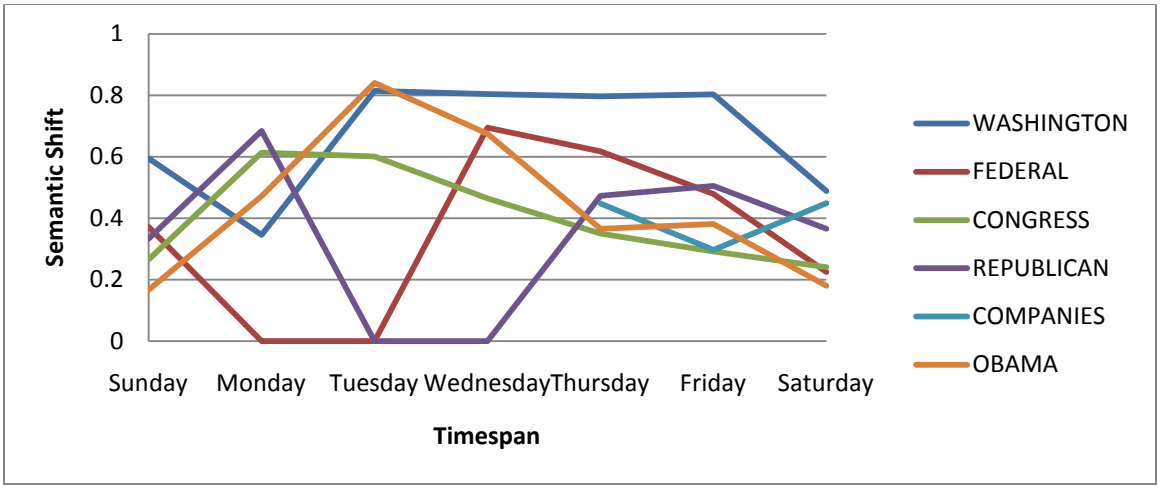
Timespan\ Interesting Word	2007	2008	2009	2010
mahindra	0	**0.4591**	0.101	0
india	**0.4451**	0.3456	**0.2155**	**0.6519**
fraud	0	**0.4312**	**0.2533**	0
satyam	**0.3956**	0.1321	0.1873	**0.6635**
Standard Deviation	0.3746	0.4127	0.2019	0.5853

From the above table, we clearly see that it is difficult to detect events, if the timespan is large. This is because events generally occur over a short span of few months and then disappear. Thus shorter the time span, better is the accuracy of semantic shift calculation and also event prediction.

- Result on Algorithm applied on dataset of "Congress" over a period of 1 day each in a week with location of Washington.

Timespan/ InterestingWord	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
washington	**0.5943**	0.3449	**0.8139**	**0.8035**	**0.7963**	**0.8025**	**0.488**
federal	0.3716	0	0	0.6947	0.6169	0.4787	0.2244
congress	0.265	0.6131	0.6007	0.464	0.3492	0.2912	0.2399
republican	0.3328	**0.6841**	0	0	0.4721	0.5049	0.365
companies	0.3894	0	0	0	0.4476	0.2965	0.4486
obama	0.1659	0.4718	**0.8404**	0.6749	0.3655	0.3803	0.1803
Standard Dev	0.4842	0.6233	0.7661	0.7592	0.6636	0.6328	0.4529

The above table represents semantic shifts calculated over a period of one week with shifts of one day each. As this is too short duration to predict any strong event we observe the entire shifts to be scattered.



Thus we can conclude that when the algorithm is utilized for detecting events over a short span of time, if the corpus is not heavily oriented towards the event, the algorithm might not perform well. But in other case, it might surely give a correct result. Thus the algorithm can be used for live detection of events as it allows incremental sampling as mentioned before.

- Results for same data set as above of "Congress" for location washington, but with a changed window size of 100

Timespan/ Interesting Word	Sun	Mon	Tues	Wed	Thurs	Fri	Sat
washington	0.4940	0.3482	0.7499	0.7604	0.7257	0.7441	0.3977
federal	0.4056	0.0000	0.0000	0.7373	0.6702	0.5425	0.2698
congress	0.2684	0.5926	0.5841	0.4683	0.3553	0.2931	0.2350
republican	0.3115	0.6941	0.0000	0.0000	0.4351	0.4881	0.3320
companies	0.4317	0	0	0	0.5018	0.3296	0.4813
obama	0.1387	0.4202	0.8021	0.6399	0.3277	0.3430	0.1539

The above table indicates that semantic shift values do not show major change on changing the window size but it does show smaller shift values. Thus we can conclude, that we could increase the window size, if the event has not been very popular. This would help consider more neighbouring terms and hence may show better results.

- Results for same data set as above of "Congress" for location washington, but with a changed value of sparseness from 6-4 to 950-1000

Timespan/ Interesting word	Sun	Mon	Tues	Wed	Thurs	Fri	Sat
washington	0.5293	0.3386	0.7692	0.7747	0.7463	0.7372	0.4259
federal	0.3544	0	0	0.6735	0.6141	0.4722	0.2347
congress	0.2836	0.6141	0.6186	0.4984	0.3950	0.3258	0.2509
republican	0.3300	0.6891	0	0	0.4506	0.5024	0.3527
companies	0.4185	0	0	0	0.4881	0.3140	0.4615
obama	0.1336	0.4203	0.7947	0.6225	0.3098	0.3226	0.1484

The above table indicates that decreasing the sparseness of the index vector does no result a huge change in the semantic shifts computed. However it does reduce the orthogonality of vectors. This can reduce the accuracy of the algorithm as the vectors need to be orthogonal. Thus the index vectors should be as sparse as possible.

Bibliography

1. **Curran, James Gorman and James R.** *Random Indexing using Statistical Weight Functions.* University of Sydney. s.l. : Conference on Empirical Methods in Natural Language Processing, 2006.

2. Standard Deviation. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Standard_deviation.

3. **Stevens, David Jurgens and Keith.** *Event Detection in Blogs using Temporal Random Indexing.* University of California, Los Angeles. s.l. : Association for Computational Linguistics, 2009.

4. tf-idf. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Tf%E2%80%93idf.

5. *reuters.* [Online] http://reuters.com.

6. *jsoup : Java HTML Parser.* [Online] http://jsoup.org/.

7. *epoch convertor.* [Online] http://www.epochconverter.com/.