A Dynamic Graph Model for Analyzing Streaming News Documents

Elizabeth Leeds Hohman
Naval Surface Warfare Center
Dahlgren, VA 22448
Email: elizabeth.hohman@navy.mil

David J. Marchette
Naval Surface Warfare Center
Dahlgren, VA 22448
Email: david.marchette@navy.mil

***Abstract*— In this paper we consider the problem of analyzing streaming documents, in particular streaming news stories. The system is designed to extract statistics from the document, incorporate these into a graph-based model, and discard the document to reduce storage requirements. The model is defined in terms of a changing lexicon and sub-lexicons at each node in the graph, with the nodes of the graph representing topics. We illustrate the methodology on a dataset of news articles, and discuss the dynamic nature of the model.**

I. Streaming Documents

Much research has been performed on text processing and there are many effective methods for representing and clustering a corpus of text documents. However, most of these methods assume that all documents in the corpus are available at once. Less work has been performed on text processing under the assumption that documents will continue to be presented. There are many types of evolving text sources available electronically such as on-line news sources and web logs. The quantity and nature of this text makes this an important area of research. We will refer to this problem as streaming document processing.

We should explain the difference between this and "streaming text". In the case of streaming text, the text is being presented in a streaming manner with no delineation between chunks of text. For example, someone typing at a keyboard or transcripts from conversations or news feeds. In this case, the processing occurs as each word is presented and before the full document is available. So each word or phrase can be thought of as an observation. In the case of streaming documents, each document is considered an observation, whether that be a news article, an email, a web log entry, or some other text observation. Any processing is performed on the full document.

The volumes of text data that can be considered in this way can vary greatly. News articles can be collected in several hundred to thousand per day while emails occur at several hundred per day (for an individual) to several thousand a day (at a server). Web postings may be in many thousands or even millions per day if one considers web logs, newsgroups, chats, and other sources. Other cases of streaming documents include incidence reports at a major computer vendor, air traffic reports, and hospital admittance reports (for detecting outbreaks). These are all examples of streaming documents that can range from several hundreds to thousands to millions per day depending on the level of data collection.

There have been algorithms developed for dynamic text clustering such as Hung and Wermter's dynamic adaptive self-organizing model (DASH) [2] and Zhong's on-line spherical k-means (OSKM) algorithm [4] but in both of these cases, a vector space model was used. This representation casts each document as a vector with length equal to the number of words in the lexicon and with each vector entry corresponding to a frequency-based weight for one of the words in the document. These algorithms used the precomputed vectors since in order to get those vectors, the entire corpus must be available. The work addressed streaming text but used a representation that cannot be achieved in a streaming context (the work developed streaming *algorithms* but applied them to precomputed text vectors). In Section I-B we will discuss methods of adaptively building these text vectors.

A. An evolving lexicon

Most text clustering and categorization methods use a vector space representation of documents. Each document is represented as a vector, $\mathbf{x_d} \in \mathbb{R}^L$, $d = 1, ..., D$ where D is the number of documents in the corpus and L is the number of words in the lexicon. Each dimension of the vector corresponds to a different word in the lexicon. The j^{th} entry of $\mathbf{x_d}$ depends on the number of times the j^{th} word in the lexicon occurred in document d.

If we no longer assume a fixed corpus, we cannot use this fixed-dimensional vector space model. Since the lexicon cannot grow without limit, approximations must be made to the representation. New words will appear in documents while words that have been seen in the past might not be seen again. Since the lexicon is constantly changing, we cannot pre-assign dimensions of the vector space to specific words. Also, since vector entries are dependent not only on the frequency of the word in the document but also on the frequency of the word in the corpus, the corpus frequency must be approximated as well in the case of streaming documents. These details are discussed in Section I-B.

B. Frequency-based word weighting

In the vector representation of a document, the j^{th} entry of the vector depends on the number of times the j^{th} word in

the lexicon occurred in the document. The value is usually the occurrence of the word multiplied by a word weight which is dependent on the frequency of the word in the corpus. This results in high-frequency but context-independent words such as "the", "and", etc. having low values.

The most common weighting method is Term Frequency Inverse Document Frequency (TFIDF) [3]. Let TF_{ij} be the frequency of word j in document i and let IDF_j be the inverse document frequency of word j in the corpus. That is,

$$TF_{ij} = \frac{n_{ij}}{\sum_{k=1}^{D} n_{kj}}$$

where n_{ij} is the number of times word j occurred in document i and

$$IDF_j = \frac{D}{b_j}$$

where D is the number of documents in the corpus and b_j is the number of documents that contain word j. Then the weight for word j in document i is given by

$$x_{ij} = TF_{ij} \log(IDF_j). \tag{1}$$

When the corpus is not fixed, the inverse document frequency cannot be calculated directly. One solution is to use a time window and calculate the document frequency within that window. In this work, we use an exponentially weighted moving average with a parameter α which allows for varying the amount of history influencing the value.

Instead of calculating the inverse document frequency, consider the document frequency, DF_j, for word j. That is,

$$DF_j = \frac{b_j}{D}.$$

Suppose that at each time, t, a new document is observed. Let $Y_j(t) \in \{0, 1\}$ indicate whether word j occurred in the document read at time t. Then we can approximate the document frequency for word j at time t by

$$DF_j(t) = \alpha Y_j(t) + (1 - \alpha) DF_j(t - 1) \tag{2}$$

which implements an exponential window on the documents. Then each document can be written as a vector using the inverse of (2) in (1).

C. A graph representation

We will represent the documents within the framework of a graph. The vertices (nodes) of the graph will contain statistics on a collection of articles that share a common theme or topic. The edges will represent similarities among the nodes. This representation (described in detail in Section III) is updated as each document is processed. The document is compared with the current nodes and is either added to a node or used to create a new node. The edges from the updated node are then updated to capture the changes to the topic within the graph. This representation performs two basic functions: it provides a reduced representation of the documents which incorporates information about the document topics and the relationships between topics; it provides a method for visualization and analysis of the topics and relationships. We will illustrate these ideas in Sections III and IV and discuss possible extensions and future work in Sections V. The graph is dynamic: the nodes and edges are created/deleted and modified as new documents are processed. Each node retains its own lexicon vector corresponding to the word frequencies for the documents in the node. This frequency vector is updated in much the same manner as (2). The frequency vector (and other statistics taken from the documents, such as keywords, title words etc) can be used to provide a summary of the node, the "topic" describing the node. These vectors are then used to define the similarity between nodes, and the edges are weighted by these similarities.

II. The Data - Google News

News articles were collected every day from January through August of 2006. Articles were retrieved through links provided by the Google news website using five categories defined by Google. The news categories and their labels were World (W), US (U), Business (B), Health (H), and Science and Technology (S). There were approximately 300 articles collected per day.

The articles were stripped of HTML tags and parsed in order to find the body of the article. The articles contained advertisements and links to other articles so effective parsing was important in order to ensure that the text clustering methods were working on the article body and not on other words that were unrelated to the articles. Perfect parsing was not possible and in cases where articles were found to be parsed incorrectly, they were eliminated from the data set. Still, it is possible that some retained articles were incorrectly parsed.

Articles were also stripped of non-alphabetic characters (all numbers and punctuation), hyphenated words were split into separate words, and words were stemmed. The stemming was performed using the Porter stemming algorithm [1]. Finally, the remaining words with two or more letters were counted for each document.

III. Representing the Text

The word counts from the cleaned, parsed, and stemmed article were used to create a vector representation for each article. In order to approximate the TFIDF representation, term frequencies for each article were multiplied by the $\log$ of the approximation of the inverse document frequency for each word as in (1). The document frequency for each word was calculated by (2). Each new article necessitated changing the document frequency for every word. The document frequency of the words that occurred in the article was increased and the document frequency for the words in the lexicon but not in the article was decreased.

A growing lexicon was used such that new words were added to the end of the lexicon. A counter was also assigned to each word in the lexicon and updated each time the word was seen in a document. Since the lexicon cannot grow without limit, an upper bound was placed on the lexicon.

Once the lexicon reached this size, a percentage of words were removed from the lexicon. The words chosen for removal were those with the smallest counter value. If a new document reintroduced a word to the lexicon, the comparison of the new document to old documents assumed (not always true) that the word was not in the old documents.

It is often the case that a list of context-free words, called a stop list, is used to eliminate the consideration of certain words in the document representation. We did not employ a stop list in this work. It was expected that the low weighting from the TFIDF representation would discount any effect from such words and the savings of only a few hundred words was not a consideration.

A. Document similarity

Typically, document similarity is determined by measuring the cosine of the angle between the vector representations of the documents. Since under the TFIDF representation the vector entries are in $\mathbb{R}^+$, the cosine of the angle between any two documents will be in $[0, 1]$. Documents that lie close to each other in document space will have a similarity measure close to one. Let $\mathbf{x}_a$ and $\mathbf{x}_b$ be the vector representations of documents a and b. Then

$$s(\mathbf{x}_a, \mathbf{x}_b) = \frac{\mathbf{x}_a \cdot \mathbf{x}_b}{|\mathbf{x}_a||\mathbf{x}_b|} \tag{3}$$

is used as a measure of similarity between the documents.

B. Creating and updating graph nodes

As mentioned in Section I-C, a graph is used to provide a reduced representation of the articles and the relationships between them and to provide a method for visualization and analysis of the news topics. An upper limit is placed on the number of nodes in the graph. As documents are read, they can either be assigned to an empty node in the graph or used to update an existing node. Associated with each node is an index into the lexicon which specifies which words in the lexicon were in articles assigned to the node. There is also a count on those words determined by the word occurrences in articles that have been assigned to the node. The count is updated in much the same way as the document frequency, by using an exponential window.

Suppose a new article is read at time t and the length of the lexicon (including all words in the article) is L. Let $\mathcal{I} \subset \{1, \ldots, L\}$ be an index set for the words in the lexicon that are also in the article. Let $z = (z_1, \ldots z_{|\mathcal{I}|})$ be the counts of the words that are in the article. If the article is assigned to an empty node in the graph, then that node will have the associated index set, $\mathcal{I}$ and set of counts, z. If the article is assigned to a node that is not empty, the index set and count set for the node will be updated. Suppose the article is assigned to the j^{th} node. Let $\mathcal{I}_j$ be the index set for the node prior to the assignment and let x_{jk} be the count for the word indexed by $\mathcal{I}_{jk}$. Then x_{jk} will be updated by

$$x_{jk}(t) = \begin{cases} z_k & \text{if } k \notin \mathcal{I}_j \\ \beta z_k + (1-\beta) x_{jk}(t-1) & \text{otherwise} \end{cases}. \tag{4}$$

As with the document frequencies for the words in the lexicon, this implements an exponentially windowed average for the counts in the node.

When a new article is read, the similarity between the article and all existing nodes in the graph is calculated using (3). If the most similar node has $s \geq \tau$, the article is added to that node according to (4). If the maximum similarity is less than τ and there are empty nodes in the graph, the article is put in an empty node. If the maximum similarity is less than τ and there are no remaining empty nodes, the article is put in the node that has gone the longest without being updated. In that case, the counts for the node are set to the counts for the article. (Note that this is equivalent to setting $\beta = 1$ in (4).)

The update rule in (4) requires the parameter β. One method for adaptively choosing β is to set

$$\beta = \begin{cases} 1 - s & \text{if } s \geq \tau \\ 1 & \text{if } s < \tau \end{cases}. \tag{5}$$

This has the effect of making large (small) changes to the node when the similarity between the article and the node is small (large).

To see what this rule does, consider the extreme cases. If the match to the node is perfect ($s = 1.0$) then the new document is providing no new information, so it is reasonable to leave the node unchanged; if $s < \tau$ for all nodes, then either a new node needs to be added or an old node needs to be replaced with the new document, starting a new topic; if $s = \tau$ then the document is very far from the topic of the node, indicating that the topic has changed, and the node statistics need to be changed maximally to adapt to the change. The parameter τ controls the amount a topic can change before a new topic (node) is created.

C. The adjacency matrix

In addition to the statistics from the news articles that are associated with each node, the graph adjacency matrix is used to retain information about the connections between the news topics. For a graph of order n (the order is the number of vertices in the graph), the graph adjacency matrix, $\mathbf{A}$ is an $n \times n$ matrix where A_{ij} holds information about the edge from vertex i to vertex j. In this work, the graph is undirected so the adjacency matrix is symmetric. For an unweighted graph the adjacency matrix is a binary matrix with $A_{ij} = 1$ if there is an edge from vertex i to vertex j and $A_{ij} = 0$ otherwise. The graph edges can also be weighted so that $A_{ij} \in \mathbb{R}$. In this work, the graph is weighted and the adjacency matrix holds the similarities between the vertices where the similarity is calculated using (3). Each time a node is updated, the similarity between that node and all other nodes in the graph is updated. That is, if vertex i is updated, then the i^{th} row and column (which are identical in the symmetric matrix) are also updated. This updating scheme is an approximation. Since the lexicon weights have changed one would need to update the entire adjacency matrix to reflect the change. In order to reduce computation, and thus allow larger graphs (more topics) we chose to implement this approximation. The approximation

may be skipped if the graph is sufficiently small and/or the rate of the stream is slow enough to allow full updating.

The adjacency matrix can be used to combine nodes that are similar. This can be done either by considering just the between-node similarity, or by using the neighborhood information within the graph to combine nodes that are both highly similar and have the same set of "most similar" neighbors.

IV. Visualization

The graph was built using an adaptive value of β as in (5). The data were articles from January 2006 from the five classes described in Section II. Fig. 1 shows the articles assigned to one node in the graph over several days' time. The first article is about bird flu. There were no further bird flu articles, so this node remained unchanged while other nodes were updated, until it became the oldest node in the graph. The article about Symantec was not similar (above the threshold $\tau = 0.1$) for any nodes, and so was assigned to this, the oldest node. Once this article is assigned to the node, the node is re-initialized and the words and corresponding weights from the first article are dropped. The following two articles are also about Symantec and are assigned to the node due to the high similarity. The fourth article, "Ahern leads trade mission to India", has low similarity but is assigned to the node due to the node remaining unchanged longer than any other node in the graph. All of the articles that follow are assigned to the node due to their similarity and show the evolution of the meaning of the node. We see that the node evolves from *India* to *India and Pakistan* to *Pakistan and the United States*, to *terrorism* and finally to *al-Qaida*. The latter articles show variations of the spelling "al-Qaida" and "al-Qaeda".

Fig. 2 gives another view of this node. The axes correspond to the words in the node and the article titles (time) with gray-scale value depicting the weight of the word at the time the article is incorporated into the node. For each article, the words chosen to display were those with the three highest TFIDF values. In addition to the top three words, the TFIDF value for other words is shown due to overlap in the lexicon at different times. For example, the Symantec articles also have the word "attack" which is a high weight word from the eleventh article ("US: Contacts with Pakistan Positive Despite Attack Protests").

The evolution of the word importance can be seen in the diagonal gray region. Early on, the important words are "Turkey", "bird" and "flu". Then words about Symantec become important for a while. The two resets are clear in this picture, indicated by the blocks in the lower corner. Then words about India become important, followed by Pakistan and then words related to terrorism. We can see here the progression of the topic about the India/Pakistani subcontinent morphing from stories about trade missions into stories about terrorism. The rate at which the node is allowed to change is driven by the parameters α and τ, which control the memory of the lexicon and the threshold of similarity for addition to the node.

V. Extensions

There are several obvious extensions that could be incorporated. First, the nodes could be allowed to split. For example, in the above example if articles about trade missions continued to be observed while articles about terrorism were also observed, one might want to split this node into two. This can be implemented through a hierarchical graph method, allowing each node to spawn its own subgraph. Thus, we would have high level generic topics, with subtopics in a graph underneath each topic.

We currently assign each article to a single node. It would be easy to allow articles to be assigned to multiple nodes, proportional to their similarity. However, care must be taken to ensure that this doesn't cause the nodes to all move together, resulting in a set of essentially identical nodes, each about "everything" and none specifically about anything.

Both α and τ are static in this implementation, parameters set by the user. We are investigating a method for adapting α by monitoring the "stable" words in the lexicon. That is, an appropriate value of α will result in a near-constant document frequency for some words. By adapting α based on the document frequency of these words, different text sources will adapt at different rates. For example, on-line news may adapt more slowly than a set of web logs but more quickly than articles from an on-line science journal. An adaptive and node-specific value of τ is another area of research and would allow different nodes in the graph to change at different rates.

VI. Discussion

We have described a method for analysis of streaming documents that uses dynamic lexicons and word weighting to organize the topics into a graph structure. We have not discussed the graph itself in depth, focusing in this paper on the methods for populating the nodes of the graph. Future work will involve extracting useful information from the graph, and implementing various extensions of these ideas.

The graph provides us with a convenient structure within which to both analyze the current topics and merge topic threads when they become similar. A hierarchical version of the graph would allow splitting of topics into subtopics. Extracting invariants from the graph may be a problem in a streaming environment, since many of the things one would like to do (spectral clustering, for example) are quite computational in nature. Fast approximations are needed to provide analysis in a streaming fashion. More sophisticated analysis could be done off-line, to provide snapshots of the document stream.

We have shown only the most rudimentary of visualization methods in this paper. More sophisticated ways of visualizing the data are clearly needed. Relationships between topics can be investigated through the similarities or links in the graph and descriptions of topics can be improved through dynamic lexicon weighting. Better ways to describe the topics and their relationships is an area for future research.

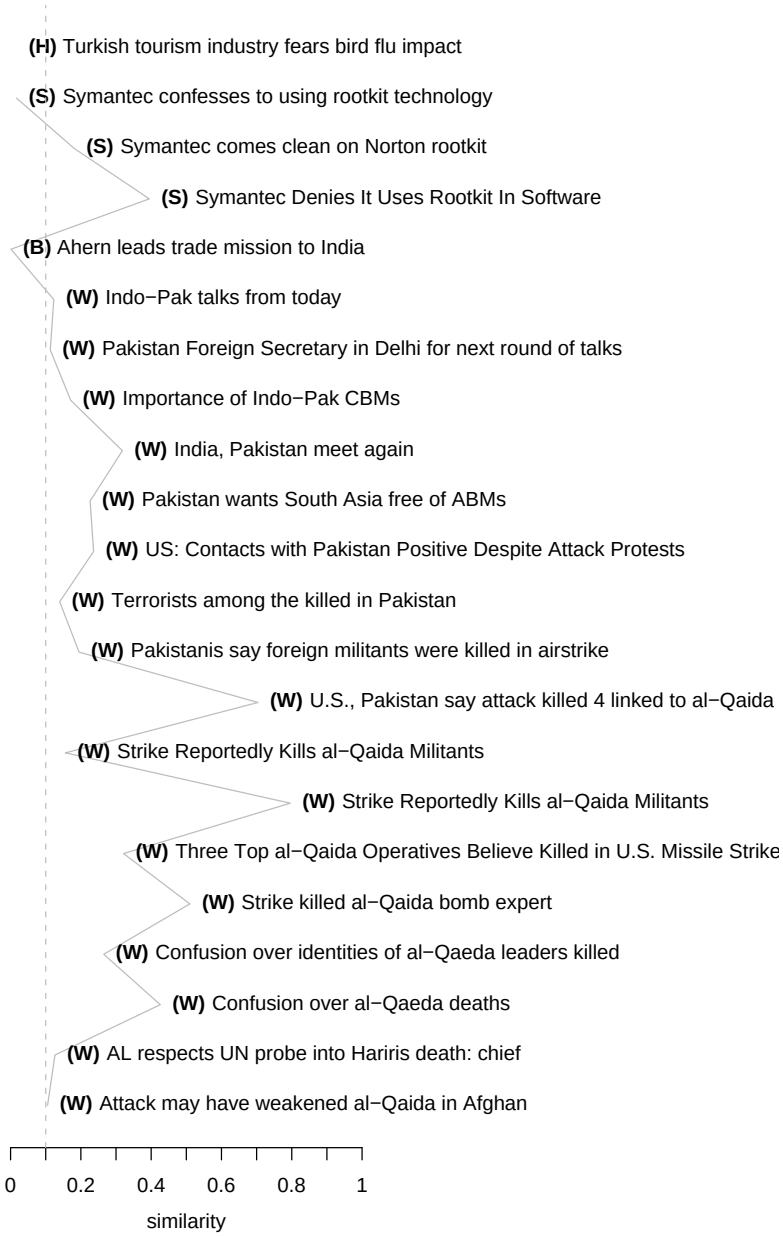

Fig. 1. **Articles assigned to one of the nodes in the graph** - The first article is shown at the top and the titles of the added articles are listed in order. The similarity between the article and the graph node is shown by the gray line to the left of the article titles. The value of τ is 0.1 and articles with a similarity less than this value were added to the node not due to similarity but because the node had been unchanged for longer than any other node in the graph.

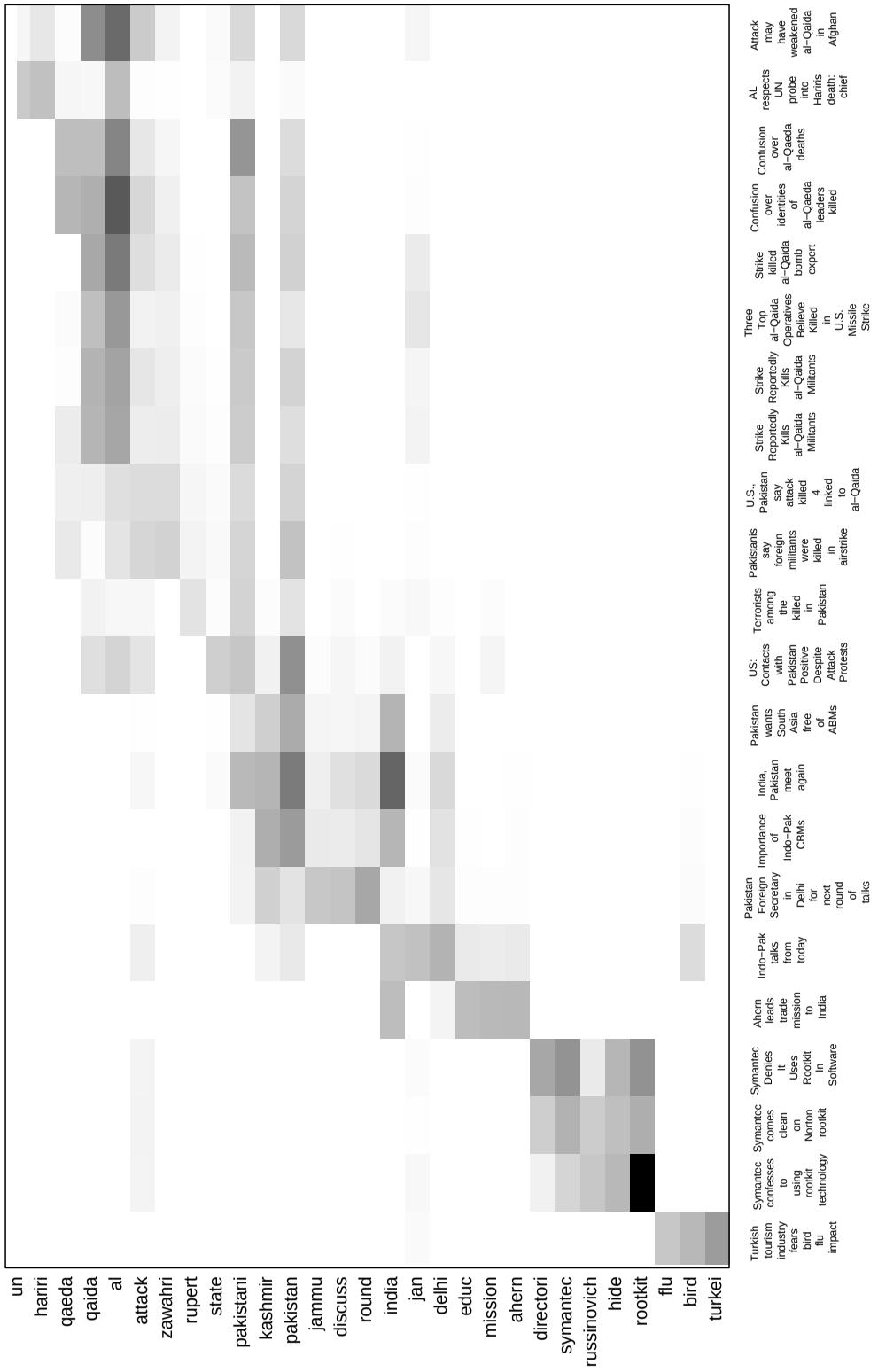

Fig. 2. **Important words in the node over time** - The words chosen for display were those with the three highest TFIDF values after the addition of each article. The gray-scale is proportional to the TFIDF value.

ACKNOWLEDGMENT

This work was funded by the Office of Naval Research through the In-House Laboratory Independent Research (ILIR) program.

REFERENCES

[1] M.F. Porter, "An algorithm for suffix stripping," Program, 14(3), 1980, pp 130-137.

[2] C. Hong and S. Wermter, "A dynamic adaptive self-organizing hybrid model for text clustering," in *Proceedings of the 3rd International Conference on Data Mining (ICDM 2003)*. Melbourne, FL: IEEE Computer Society, December 2003, pp 75-82.

[3] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*, New York: McGraw-Hill, 1983.

[4] S. Zhong, "Efficient streaming text clustering," Neural Networks, 18(5-6), 2005, pp 790-798.