Analyzing Topic Evolution in News Collections

Mihai Paraschiv

University POLITEHNICA of Bucharest

Faculty of Automatic Control and Computers, Computer Science Department

Email: mihai.paraschiv@cti.pub.ro

1. Abstract

This paper introduces a system that detects and tracks topics aggregated on a two level hierarchy. One of these levels contains static topics (called stories from this point forward), which are used to group articles referring to short term news subjects (i.e., an event in a series of related ones). The second level contains dynamic topics, which accumulate stories pertaining to a series of related events. This system can be used as basis for novelty detection mechanisms or personalized summary generation algorithms.

Keywords: topic discovery and tracking, system, news article analysis

2. Introduction

2.1. Motivation

Nowadays, content is almost always created by humans, but in the near future, we will observe a shift towards machine generated content. At first, this will be done by mixing pieces of human-authored textual content taken from multiple sources. An example of an approach that fits this description is news summarization. In this case, multiple related articles are clustered and blocks of text that best describe the topic are shown to the user in a story-like order. While successful in academia, this technique hasn't yet been applied in important user facing applications. In this phase, we are going to see developers exploring ways of ranking content sources at topic level, determining influences and personalizing summaries according to what users have already read. Provided that summarization methods are good enough to create almost human-like text, we will see context being adapted to reading context. This will be especially important for reading on small screens like those of mobile devices.

The next phase would probably involve augmenting the summarized content with information from semi-structured sources like Wikipedia and various knowledge bases. Unlike today, when such information is simply overlaid on top of raw content, we will likely see full syntactic integration with the base text. By this time, social information will be seamlessly merged with the content. Text will contain references to articles already read or photos and videos already seen, not only by the users, but also by their peers. Moreover, we will probably see how points of view expressed in the text relate to those expressed by people the users follow. By the end of this phase, the reading experience will have changed dramatically. Content will not just be adapted, but users will have the option of allowing how much content is made available to them and how they will digest it. For example, if a user wants to read about new subjects for ten minutes every morning, a narrative that will contain information to fit his interval will be presented. Besides time constraints, this story will have to account for news diversity, relevance to the user and fluency.

After these developments, we are going to see the fully machine generated content. Initially, text will be created by superficially understanding previous work. Later, we may have techniques for creating original content like predictions and reports through analyzing similar happenings. However, content creation will not be limited to text. Machines will probably be able to edit both photos and videos for inserting alongside the main text.

2.2. Topic detection and tracking system

This paper introduces a system that detects and tracks topics aggregated on a two level hierarchy. One of these levels contains static topics (called stories from this point forward), which are used to group articles referring to short term news subjects (i.e., an event in a series of related ones). The second level contains dynamic topics, which accumulate stories pertaining to a series of related events.

After this system is complete, one can envision using it as the basis for novelty detection mechanisms. Additionally, on top of the current system, we can add personalized summary generation and recommendation algorithms.

2.3. Paper structure

The rest of this paper consists in an overview of the research field, a description of the models used in the system and a presentation of the architecture and implementation. Moreover, the first evaluations of the system along with further work and conclusions are shown at the end of the paper.

3. Background

According to [1], a news article consists of three main parts: attribution, abstract and story. Attribution is observed from two perspectives: the author(s) and the source of text (e.g., a news agency). The abstract indicates the nature of the article and is made of the lead sentence and a headline. The lead describes essential facts and introduces the actors involved in the event. The body of the story is commonly separated into one or more episodes which group events with a common location or shared actors.

The events are the main elements of a story. Their description consists of setting (place and time) and actor (e.g., people, organizations) information, observed through their actions. A story has one or more central events, but it also contains references or summaries of previous ones, which are typically found in background sections of the text.

An author does not only narrate what has happened, but also provides commentaries on the event. A common function of these remarks is helping readers understand the subject by giving context information. An author can evaluate the consequences of the described actions and may present personal opinions. Other people attitudes toward the topic, sometimes expressed through quotations, can be added to support the comments.

3.1. Story development

In order to build a system capable of identifying new topics and track user specified ones, one has to have a good understanding of the types of documents on which the algorithms are applied. A characterization of the common types of news articles, seen from the perspective of novelty detection, is shown in [2]:

- **Recap**. These are article which are relevant for a tracked topic, but do not carry novel information.
- **Elaboration**. Most novel information is extracted from these articles. These add new information and commonly include briefs of other stories relevant to the topic.

- **Offshoot**. Such articles are relevant to the mainstream readers and are related to the topic, but they shift the discussion to other subjects. When building a graph of topics, these stories provide meaningful branching points.

Topic Detection and Tracking (TDT) refers to automatic techniques for locating topically related material in streams of data such as newswire and broadcast news [3]. The work done on the project was intended to discover algorithms for analyzing news feeds with respect to the topics that had been discussed in the articles. In TDT research, a “topic” is defined to mean a specific event or activity plus directly related events or activities. TDT comprised of the following tasks: story segmentation, topic detection, topic tracking, new event detection (first story detection) and link detection.

3.2. Real systems

The architecture of a topic tracking system could include the following three layers [4]:

- Input processing: This layer deals with the information sources and transforms the documents into internal models. Commonly found components are: web crawlers, feed readers, lexical analyzers, section extractors.
- Representation and transformation: This is where the processing takes place. Components handle tasks like: clustering, parsing and entity recognition.
- User interface: This is the view part of the architecture. Besides outputting information to the user, this layer can include an API or provide feeds.

Since the beginning of TDT research, various systems that fit in the topic tracking category have been developed. One of the best known is Newsblaster [5]. The system produces news updates by crawling sites, grouping articles according to their topic and generating summaries. Articles are gathered from a variety of sites by traversing up to maximum depth and news text is extracted based on a set of rules. Stories are organized in a hierarchical structure consisting of three levels: predetermined categories, topically related events and distinct events. The system builds clusters using a groupwise agglomerative technique and similarity is measured through a combination of tf-idf weights and linguistically motivated features.

NewsInEssence [6] is a service created as part of a university research project at the University of Michigan. The main object in the system is a cluster containing topically related articles. News summaries are produced with a multi-document summarizer called MEAD, introduced in [7].

Newistic [8] is a content gathering and analysis platform which can be used to track news articles and blog entries for specific keywords, topics or categories. Newistic tracks subjects in multiple languages and provides various search options.

4. Models

The model hierarchy is made up of three levels: resources, stories and topics. Common to all is the usage of the same feature types: terms and named entities. These features are extracted from the content of news articles and are currently the only type of textual information that the clustering algorithms use as input.

4.1. Resources

Resources form the base layer of the model hierarchy and are used to represent objects with an URI, and author and an available publishing time (e.g., a blog post, a news article, a tweet, a photo). In addition to the three properties, resources have a content property, which varies in form according to the resource's type.

- URI: In the case of objects available on the web, this is a URL.
- author: The person or team of people who produced the resource.
- publishing time: The time at which the resource was made available.
- content: In the case of articles, the content can be is made of title and article text. In the case of a photo, the content is made of title, description and image. In many cases, author assigned tags can be included in the content.

In the system, a resource's main purpose is to show an author's point of view, on a specific matter, at a given time. Modeling an author's opinion in a manner that facilitates clustering and comparison is difficult when we consider free, natural text. Focusing on news content, we can observe that most text deals with facts and real entities. As a result, it may be enough for our system to create a resource model by studying only the relationships between its concepts.

A resource has a simple representation of content features. These are extracted at processing time and their importance is assessed only with respect to the internal structure of the resource's content. Unlike representations found in models higher in the hierarchy, feature representations for resources are static with respect to time and can only change if the referred resource is updated.

4.2. Stories

Stories make up the first layer of generated models and are used to cluster related resources. Stories vary significantly in their composition. They range from the ones that contain a single article along with related social responses, to those stories including thousands of articles, photos and videos pertaining to an important event. No matter their size and structure, all stories serve a similar purpose: to capture views from multiple authors on specific news subjects.

As we can see, a story is defined by its subject. However, understanding the nature of the subject is not straightforward because we have to account for all types of news content and do this in the context of resource clustering. Nonetheless, in practice we may not need to use define the subject in this manner. Instead, we can classify most stories as event or theme based.

Usually, the story's subject is an event of interest to the general public. In this case, most of the content will probably be found in articles that objectively describe what has happened. Additionally, if the subject is significant it will probably be addressed by a large number of news sources and will attract a great deal of commentary on social platforms. Traditional commentary (i.e., subjective articles) may also be present in these stories.

Other types of story subjects are those that group together resources with a common theme. This is the case of stories that have a subjective or investigative nature and do not discuss a core event, but a topic of interest (e.g., price changes, an impending war, the status of a nation's natural resources). Most of the time, these type of stories take shape around a very tight group of articles, maybe even a single one, and can be seen as a distributed (from the point of view of where the content is found) discussion on a common topic.

Clustering resources into stories must take into account two main issues. First, most news articles describe multiple subjects and it is not always easy to select the core one. Second, we want to add resources to stories as they are found, either one by one or in very small batches. As a result, we cannot assume that we know at story inception time what its subject should be. For example, after an unexpected event has just occurred, there will a lot of raw information coming from various sources. After a while, new articles will contain details of the event, its actors and possible connections to other events. Thus, we should assume that the story's subject has to remain open for change.

Stories are created from resources, so we can naturally use the same representation for properties related to content. The story will contain a feature model that is directly built upon the one used for extracted resource features. However, feature scores cannot be static, but have to be incremental. Initially, the story's set of features is just a copy of the one found in its seed resource(s). When a related resource is detected, the story's model is augmented in an incremental fashion: feature scores from the story are summed up with those from the new resource. Furthermore, a resource may have an importance value. Thus, feature scores increments should be directly proportional to this value.

While resources can be expressed through their relationships between concepts, stories can be understood by the connections between their child resources. A story model must facilitate the aggregation of information extracted from resources. Examples of group properties that can be assessed at story level are: resource importance, people's interest and people's sentiment on the subject or on the way it is portrayed. Additionally, by grouping resources from multiple sources, we can evaluate the importance and truth value of extractable facts.

4.2.1. Story score

Story models are incremental, but growth reaches a limit when the story can be considered complete. While we may always find new resources that can be added to a story (e.g., a very late trackback to an article in the story), their significance would not justify the effort to permanently keep track of all clusters. This is because a story’s importance also decreases as time passes by.

Because a story’s score is formed by summing up the scores of its resources, this value is considered to decay exponentially. Thus, we assume that a story is relevant as long as its children resources are significant.

4.2.2. Story states

The story begins in the *open* state. In this state, the system will consider it when searching for stories that match an incoming resource. As its exponentially decreasing score reaches the required threshold, the state is changed into *closed*. In this state, the story is no more available for resource insertion.

4.3. Topics

Topics are used to model the change of news subjects and are built by analyzing relationships between stories. In the context of subject evolution, a story can be seen as a snapshot. A topic aggregates all these snapshots and facilitates seeing the news subjects as streams of transitions between snapshots. Thus, a good mechanism for building topics can reveal how opinions change or how relationships between entities transform. Moreover, following feature evolution provides information about how and why old and current trends emerge.

A topic starts with one seed article, which provides the first set of statistics for its model. As new articles are added to the topic, the statistics change, but this process is different for each feature. Some of them turn out to be the core elements in the topic, as they appear in many entries. Meanwhile, other features fade in importance. In addition, new features emerge and signal new directions in the overall story and possibly new subtopics. The purpose of the method discussed next is to help build algorithms for identifying trends.

Consider a topic T, which has a set of articles. Each article a has a set of features. The relevance $\mathrm{r_a}(f)$ of a feature f in an topic is computed based on the importance of the feature in the a. Each feature may be found in multiple entries. A simple way to compute this importance score is to use the term frequency.

The score that is due to one article is:

$$S_\mathrm{a}(f,t) = \lambda^{t-t_0} * \mathrm{r_a}(f)$$

The total score of the feature in the topic is:

$$S_T(f,t) = \sum_{e\in T} S_\mathrm{a}(f,t)$$

When a new article is added (at time t_n), the total score for a feature is computed as:

$$\begin{aligned} S_T(f,t_n) &= \sum_{\mathrm{a}\in T} S_\mathrm{a}(f,t_n) + \mathrm{r_a}(f) \\ &= \sum_{\mathrm{a}\in T} \lambda^{t_n-t_m} * S_\mathrm{a}(f,t_m) \\ &+ \mathrm{r_a}(f) \\ &= \lambda^{t_n-t_m} * \sum_{\mathrm{a}\in T} S_\mathrm{a}(f,t_m) \\ &+ \mathrm{r_a}(f) \\ &= \lambda^{t_n-t_m} * S_T(t_m) + \mathrm{r_a}(f) \end{aligned}$$

5. Implementation

The core part of the system is a pipeline that combines article discovery, processing and clustering. Components included in this pipe communicate through custom built message queues. Secondary components are implemented as Apache Thrift[1] services and are used for purposes like text extraction and model update.

Model data is stored in several MongoDB[2] databases.

[1] http://thrift.apache.org/

[2] http://www.mongodb.org/

5.1. Text Extraction Service

Before the content of a resource can be analyzed with NLP techniques, the text should be cleaned from markup. However, because we gather raw pages from the web, we also have to determine which text segments form the actual content and which ones have web site specific functions, e.g., navigation, advertising, comments. The text extraction service does both these tasks by wrapping the BoilerPipe[3] open source library, which provides several shallow parsing algorithms and applies various heuristics to control the content detection process. The service uses the standard Article Extractor, which is tuned towards news text.

5.2. Message Queue Service

Message queues are the main means of communication between system components. The service is built on top of a MongoDB database and is inspired by the Amazon Simple Queue Service. The protocol provides methods for both message and queue management, as one server can handle multiple queues. In the database, queues are represented using MongoDB collections and messages using documents. The message type's main characteristics are: identifier, body, creation time, availability status.

Once a message is added using the PUT command, the server checks if a queue with the desired name exists and creates one if not. Next, the message is inserted in the respective MongoDB collection and is immediately available for reading. At inception messages are given an identifier and are put in the available state. When a request for reading a message is received through the GET command, the server marks the message as unavailable and returns it to the client. When the client finishes processing, it calls the DELETE

[3] http://code.google.com/p/boilerpipe/

command with the message's identifier. Messages stay in the unavailable state until a client asks for deletion or a predetermined amount of time has passed. This way, messages are not lost, even in the case of a client failure.

5.3. Discovery

The discovery process is designed for supporting the evaluation of system algorithms. As a result, it consists of several tools for aggregating news articles from a set of web feeds. The discovery process is done in four steps. In the first one, an application continuously reads feeds and saves the identified feed entries. The feed list can be either specified at application start time or detected at run time, as in the case of crawling the NewsBrief site. Then, another application loads web pages corresponding to discovered resources. Loading pages is necessary because not all feeds contain the full entry content. In the third step, the resources discovered by the feed reader are merged with their web pages and saved for evaluation. Finally, these evaluation resources are ordered by their publishing time and sent to the discovered resource queue.

5.4. Processing

In the processing phase, the system takes discovered resources, analyzes them and creates the final resources. The Processing component is designed by following the Chain of Responsibility pattern and consists of a Processor, several Command classes and an error handling mechanism. The Processor is the main unit of the component and controls its main cycle (input, analysis and output), creates statistics and handles errors that appear in resource processing.

The following commands are used in the processing phase:

1. Input. Items are read from the discovered resource queue and added into a per-resource shared context.
2. Text extraction. The main textual content is selected using the Text Extraction Service.
3. Parsing. The default parser in NLTK is used to identify parts of speech tags in the resource's content.
4. Term extraction. All nouns and verbs are selected and lemmatized using WordNet. The term scores represent the number of occurrences of the lemma in the content.
5. Entity extraction. Alchemy API is used to extract entities from the content. Entities detected by the web service are normalized and have a relevance score that is saved in the output resource. Because settings may be adjusted in the system evaluation process, the service responses are cached.
6. Output. Aggregates the results from all commands and creates the final resource. This is then added in the processed resource queue and the input message is deleted from its queue.

Exceptions that are raised during processing are sent to predefined handlers. A special category of exceptions consists in those used for filtering resources based on criteria like content length and feature count.

5.5. Model management

Before resources are made available to users, they have to be attached to stories, a process which involves two steps. First, the story that best fits the resource's model is identified or, if no suitable match is found, a new one is created. In the second step, the story's model is updated with information from the resource. As we will see, this separation allows to the system to group the updates into batches in order to reduce the number of expensive operations required for saving a story.

The system retrieves items from the processed resource queue, which are then inserted into MongoDB and the main index. Next, the system searches the open story index for the one most similar to the resource. If the story's similarity score is lower than a predefined threshold, a new model is created and inserted in the database. In either case, a story update operation referencing the new resource is generated.

The second stage of attaching an incoming resource to a story takes place in the Story Updating Service, which exposes a method that updates a specific story. When this method is called, all operations that were queued in the story's update buffer are applied. In this process, the story's model is updated with resource information, saved into the database and re-indexed. Delegating updates to a service provides several advantages when trying to improve the system's design. First, in the case of stories that receive a lot of updates in a short period of time, it may be beneficial to accumulate the operations and run all of them together. This way, we avoid costly operations like MongoDB updating and ElasticSearch indexing. Second, the design of the resource insertion mechanism is not influenced by decisions affecting story management like, e.g., obtaining locks on story objects.

The process that checks for stories which need to be closed and thus removed from similarity search runs in parallel to resource insertion. At constant time, this component verifies if the closing score of any of the stories is below a predefined threshold. If true, the process sends a request to the Story Updating Service to remove the story from the open story index. The closing score is based on the sum of resource scores and decays exponentially.

5.6. Search

Indexing and searching of models is accomplished with ElasticSearch[4], which is a library that facilitates the control of distribution of Lucene indexes on multiple machines. Additionally, it provides schema free document indexing and a REST interface, but inherits the limitations of found in Lucene, e.g., fixed form of similarity scoring formula.

One of the goals of the system is to cluster resources into stories in an iterative fashion. Consequently, we need a mechanism that can compare models (resources, clusters of resources) based on the features extracted in the processing phase. Because Lucene's default analysis, indexing and search mechanisms are geared towards textual content, we have to alter them. A simple solution to the problem of storing features in the index is to approximate their scores with integers. Then, we need to change only the default text analysis mechanism to emit a token's (feature name) appearance as many times as required. However, not all scores can be adequately approximated like this. First, entity scores for one resource take values between 0 and 1. Thus, we would need to scale these scores. Furthermore, a cluster would contain scores of varying orders of magnitude. Second, we may need to consider to index models with scores that decay as time passes and this would require dynamically setting the scaling factor. Finally, the search algorithms would be less precise than they are when using floats. The mechanisms described below try to solve the core issue in a more satisfactory way.

In addition to storing the frequency of a term in a document, Lucene indexes can record term payloads. The normal Lucene strategy is to simply multiply the values found in these slots with term frequencies. Our models contain a set of feature maps

[4] http://www.elasticsearch.com/

(feature name, feature value). For indexing, these maps are serialized into strings that are sent to a built-in payload tokenizer.

Search operations are performed in Lucene with a generalized tf-idf formula. One of the values that enter in the computation of the similarity score is the field length normalization factor. Since we do not store normal text in feature fields, but a map serialization, we can omit this value. Instead of using it, we normalize the feature scores with the Euclidean norm of the whole feature set.

Model comparison is done using a query that is designed to handle payloads. Multiple instances of this query are created for each feature and chained with the Boolean OR. Because not all features have to be deemed important and due to the performance hit of large queries, the system limits the features added to the query. First, a predefined number of entities are selected. Then, the query is completed with terms.

The system stores models in two indexes. The first one is the main index, which contains all resource and story models. The second one contains only open stories and their resources. This index is used for finding stories that are similar to incoming resources. While the first index continuously grows, the second one is kept small be discarding stories which have lost significance (i.e., their closing score is below the predefined threshold).

In both indexes, we find stories along with resources. The decision to create mixed clusters has been motivated by the need to get good approximations for the inverse document frequencies of terms in stories. Indexing the stories by themselves would result in values that would have no real meaning. Take for example two features: A and B. Consider that A appears in all the resources of a given story and B occurs in only one them. If we were to use document frequencies evaluated on a per cluster basis, both A and B would have the same value (at least in relation to the example story). In contrast, if we were to use a mixed index, the idf values would be controlled by the items as there are more of them than clusters. The main reason for using this technique is that the document frequency update mechanism is controlled by Lucene directly and ElasticSearch indirectly, which makes it difficult to reliably modify.

6. Evaluation

Currently, the process of clustering stories into topics is not complete. As such, the evaluation focuses on story creation and updating.

To evaluate the system, a collection of 20,000 news articles has been created based on clusters from the NewsBrief[5] site.

Before clustering, resources that did not meet the following requirements were removed:

- minimum content length: 100 characters
- minimum terms (nouns or verbs): 3
- minimum entities: 1

In all tests, it was assumed that a named entity is ten times more important than a term.

6.1. Measure similarity

The system uses a similarity based on Lucene’s default measure, which was designed for searching in document collections. As we are using the same mechanism to look for matching clusters, we need to assess its properties. The tests were performed on two collections of resources and consist in computing all pair-wise similarities between 100 models, in both senses (both paired models take the role of query).

[5] http://emm.newsbrief.eu/

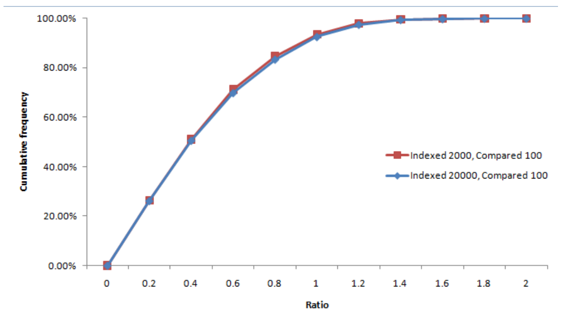

Fig. 1 Ratio between similarity difference and average similarity

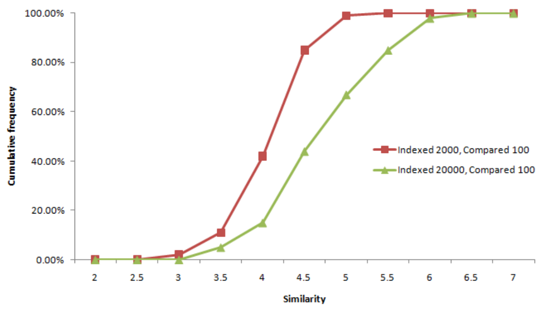

Fig. 2 Self similarities

The cumulative distributions shown in Fig. 1 are almost identical, despite that one index is an order of magnitude larger than the other. However, similarities between the same resources (Fig. 2) are lower on the smaller index.

6.2. Resource clustering

The output of the resource processing pipeline was evaluated by comparing system generated story sets with the NewsBrief cluster collection. It is worth noting that it is difficult to assess the quality of this approach as we do not have access to the same data sets. Two causes contributed to this: some of the articles have not been available for downloading and the text extraction algorithm is probably different from the one used by NewsBrief.

For the following tests, it was considered that a resource's half life is 7200 s and a story's minimum time in the open state is 86400 s. A resource's score is 1 and the score at which a story is closed is 2.

The minimum similarity for matching a resource to a story is 0.2 in the first test and 0.4 in the second.

	NewsBrief	Similarity 0.2	Similarity 0.4
Stories	998	783	1623

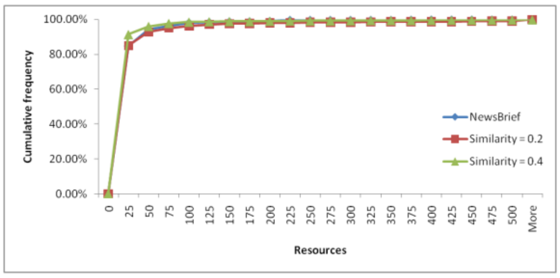

Fig. 3 Resources per story

The graph in Fig. 3 shows that the proportion of resources per story is almost equal for the three collections.

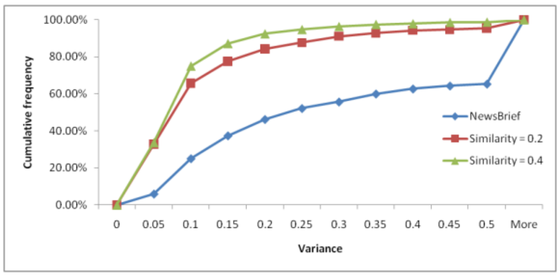

Fig. 4 Within-story variance

Generated clusters have a lower within-story variance than those created by NewsBrief, suggesting that the former clusters contain more closely related resources.

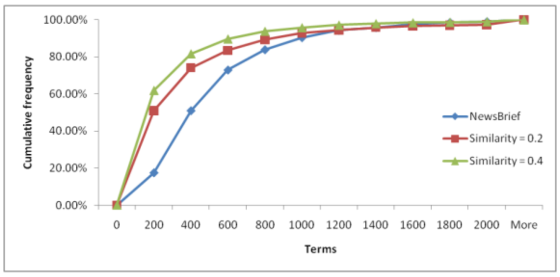

Fig. 5 Terms per story

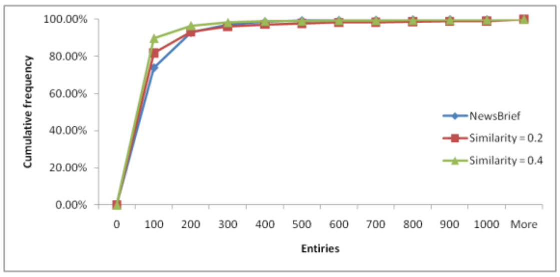

Fig. 6 Entities per story

Stories clustered by NewsBrief contain more terms (Fig. 5), but the number of entities is almost equal in all sets (Fig. 6).

7. Further work

The project will continue with the implementation of topic analysis and trend detection. Currently, a larger collection of news articles is created based on NewsBrief data and this will allow us to evaluate the system on more diverse data sets. In addition, new tests will be designed for testing the quality of the system's output. Furthermore, the applications have to be tested in a distributed environment and in real life stress conditions.

8. Conclusions

This paper introduced a new design of news article clusters and presented a working implementation. Additionally, algorithm evaluations were made for some parts of the processing pipeline. While the system has not yet been tested in a distributed environment, the components were designed to work in real life settings. Current results are promising, but more testing needs to be done, especially on the output quality to determine the behavior of the system when diverse, possible corrupt or incomplete data is used.

References

[1] A. Bell and P. Garrett, *Approaches to media discourse*, Wiley-Blackwell, 1998.

[2] E. Gabrilovich, S. Dumais, and E. Horvitz, "Newsjunkie," *Proceedings of the 13th conference on World Wide Web - WWW '04*, 2004, p. 482.

[3] C. Wayne, "Multilingual topic detection and tracking: Successful research enabled by corpora and evaluation," *Language resources and evaluation conference (LREC)*, Citeseer, 2000, p. 1487–1494.

[4] J. Allan, *Topic detection and tracking: event-based information organization*, Springer, 2002.

[5] K.R. McKeown, R. Barzilay, D. Evans, V. Hatzivassiloglou, J.L. Klavans, A. Nenkova, C. Sable, B. Schiffman, and S. Sigelman, "Tracking and summarizing news on a daily basis with Columbia's Newsblaster," *Proceedings of the second international conference on Human Language Technology Research* -, 2002, pp. 280-285.

[6] D. Radev, J. Otterbacher, A. Winkel, and S. Blair-Goldensohn, "NewsInEssence," *Communications of the ACM*, vol. 48, 2005, pp. 95-98.

[7] D. Radev, "Centroid-based summarization of multiple documents," *Information Processing & Management*, vol. 40, 2004, pp. 919-938.

[8] O. Dan and H. Mocian, "Newistic: a distributed news gathering and analysis platform," *Proceedings of the International Conference on Web Intelligent Systems '09*, Chennai, India: 2009.

[9] Y. Chen and L. Tu, "Density-based clustering for real-time stream data," *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '07*, vol. d, 2007, p. 133.

[10] V. Hatzivassiloglou, L. Gravano, and A. Maganti, "An investigation of linguistic features and clustering algorithms for topical document clustering," *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '00*, 2000, pp. 224-231.