Event Clustering for Time-Based Document Sets

By Elaine Mao

Introduction

This project demonstrates an approach to visualizing and understanding large document sets containing a chronological component. By applying standard text analysis techniques to a chronologically-sorted document set, it is possible to identify “events” in the document set which correspond to the historical record. The document set I used for this project is the WikiLeaks cables--a collection of more than 250,000 documents spanning a period of more than 40 years. A visualization of the first 500 cables can be seen here.

The original goal of this project was to examine the efficacy of using computational techniques to analyze large datasets, in comparison to the currently common practice of analyzing such datasets manually. When the WikiLeaks cables were first released, many news organizations mined this document set for stories, but the predominant approach was to look at only a specific subset of the cables. For instance, one common approach was to search for a particular term or location within the cables, and look only at the documents which contained the search query. This is a good technique for doing very focused stories on a predefined subject, but it is not necessarily the best approach for large document sets for which we don't have full knowledge of the contents.

An automated approach has the advantage of providing a more holistic view of the document set. However, when relying on an algorithm to do text analysis and event detection, it is important to determine how accurate the algorithm is. Compared to a manual analysis of the cables done by humans, there are obvious tradeoffs in terms of time and quality. Given these tradeoffs, at what point is an algorithm “good enough”?

In the process of trying to answer this question, I implemented several variations of a clustering algorithm. I then had workers on Mechanical Turk evaluate the different variations, to determine the version of the algorithm that produced the “best” results from a human perspective. However, this only raised a host of new questions. How do you actually evaluate the quality of a clustering algorithm? What differentiates one event cluster as being “better” than another one? What factors are important to take into consideration when evaluating clusters for use in a journalistic context? How do you design an experiment that appropriately captures the performance of the algorithm with respect to these factors? Some of these questions are beyond the scope of this semester-long project, but I hope that the results and observations presented here will be of use in future research in this subject area.

Related Work

Much of this project’s approach was based on the approach described in *Anytime clustering of high frequency news streams* (Moerchen et al. 2007). As the title suggests, the algorithm described in this paper is an online algorithm that is designed to quickly process incoming articles from a news feed. The algorithm processes documents as they come (chronologically), and makes no prior assumptions about the total number of clusters.

When the algorithm receives a document as input, it applies TF-IDF (text frequency document frequency) to the document, and then compares the document to the existing set of clusters. If the document is sufficiently similar to one of the clusters, it is assigned to the cluster with the highest similarity; otherwise, a new cluster is created with that document.

The paper also describes a few variations of the algorithm. One approach is to emphasize certain metadata in the TF-IDF analysis (as opposed to discarding it). Another approach is to limit the scope of the clustering step by only considering clusters that have been recently updated.

Text Processing
I processed the cables using standard text analysis techniques. First, I replaced all hyphens with spaces and removed any additional punctuation from each cable. Next, I removed “stopwords,” which are frequently occurring words that contain little information (e.g. “and,” “the,” “I,” etc.). The list of stopwords was obtained from the Natural Language Toolkit 3.0 for Python. I then split the resulting string on spaces to obtain a list of “tokens,” or individual words. I also extracted one piece of metadata (the name of the origin embassy), and appended this token to the list four times (in order to give this piece of information additional weight). All other metadata were discarded.

Having turned each document into a list of tokens, I then applied TF-IDF to the corpus. I used the TF-IDF implementation included in the Gensim library for Python. TF-IDF stands for “Term Frequency Inverse Document Frequency,” and it is a common text analysis technique. “Term Frequency” refers to the number of times a given term occurs in the document, and “Document Frequency” refers to number of documents the term appears in (within the entire corpus of documents). The Document Frequency is inverted to reduce the weight of terms that are common in many documents. Applying TF-IDF turns each document into a vector of features, where each feature corresponds to a word that could potentially occur in that document. The total number of features is equivalent to the total number of different words in the entire corpus. Gensim’s TF-IDF implementation generates a TF-IDF similarity matrix, where the similarity between a given pair of documents is the cosine similarity between the corresponding vectors. For more on TF-IDF, please refer to additional resources.

Clustering
The WikiLeaks cables are already sorted chronologically, so no additional processing was needed to sort the corpus. The base algorithm (without variations) is described below:

```
for each document in the corpus:
    if there are existing clusters that satisfy our [SELECTION CRITERIA]:
        for each cluster:
            calculate the similarity between the document and the cluster
        identify the most similar cluster
        if similarity to the most similar cluster is greater than [THRESHOLD]:
            add the document to that cluster
        otherwise:
                create a new cluster for the document
    otherwise:
            create a new cluster for the document
```

This is a basic online clustering algorithm, since it processes documents as they come, with no prior knowledge of the total number of clusters, or of the documents it has not processed yet. I implemented three different versions of the algorithm, which consist of different variations of the two bracketed values in the above pseudocode:

<u>Version 1</u> (**BASIC)**
[SELECTION CRITERIA]: none
[THRESHOLD] = 5%

<u>Version 2</u> **(TIMEOUT)**
[SELECTION CRITERIA]: only consider clusters which have been updated within the past year
[THRESHOLD] = 5%

<u>Version 3</u> **(HIGH THRESHOLD)**
[SELECTION CRITERIA]: none
[THRESHOLD] = 10%

The threshold value in the **(BASIC)** algorithm is approximately equal to the mean plus one standard deviation. For the **(HIGH THRESHOLD)** version, I simply doubled that value.

In **(TIMEOUT)**, the timeout limit of one year was chosen somewhat arbitrarily. The value may vary on a case-by-case basis, depending on how time-constrained our definition of an “event” is. Future evaluations could experiment with different timeout values.

Creating Mechanical Turk HITs
I applied each version of the clustering algorithm to a subset of the WikiLeaks cables consisting of only the first 10,000 cables, or less than 5 percent of the total corpus. It was more convenient and efficient to run the algorithm and the evaluations on this subset, since there were fewer cables to process and the resulting clusters were smaller and easier to evaluate. I generated two sets of comparisons to run Mechanical Turk experiments on:

1. **(BASIC)** vs. **(TIMEOUT)**
2. **(BASIC)** vs. **(HIGH THRESHOLD)**

For each experiment, I generated 50 comparisons, and I requested that each comparison be evaluated by 4 different evaluators. Workers were paid $0.20 for each comparison. Each comparison consisted of one cluster resulting from the **(BASIC)** algorithm and one from the variation I was comparing against. To avoid overwhelming the Mechanical Turk evaluators, if a cluster was longer than 10 documents, I randomly selected a subset of 10 documents for evaluation.

To choose the clusters themselves, I also used a random selection algorithm. I first generated a random number between 1 and 10,000, corresponding to a single document within the corpus. I then found the corresponding cluster containing that document for both algorithms in the comparison. If the length of either cluster was longer than 10 documents, I truncated the cluster by randomly selecting (without replacement) 10 documents, and then sorted the resulting set chronologically. I output the comparisons into a CSV file, which I then imported into Mechanical Turk to create the Human Intelligence Tasks (HITs). A sample HIT can be seen here.

Mechanical Turk workers were given the following instructions:

“Read the following two sets of text snippets. Each set contains **up to 10 text snippets**, each one corresponding to **the subject line of a document from the WikiLeaks cables**. A set may contain fewer than 10 text snippets. If there is not enough information in the text snippets provided, you can **click on the link at the bottom of the snippet to view the full text of the cable**. Please select the set of text snippets that you feel best corresponds to a single historical event. If both sets do an equally good job, please select the option: "Both sets are equal." **Please justify your answer with a short explanation in the box provided. ANSWERS WITHOUT JUSTIFICATION WILL BE REJECTED.”**

Workers were asked to answer the question, “Which set of text snippets better describes an event?” They were provided with a 200-character text box to justify their reasoning.

In previous iterations of the experiment, I provided workers with a more detailed description on what an “event” might entail, but in the final version I decided to simply rely on workers’ intuitive understanding of what an “event” is. I also previously neglected to mention that a set might contain fewer than 10 cables, so workers would often point out that one set was “broken” because it only contained a few documents.

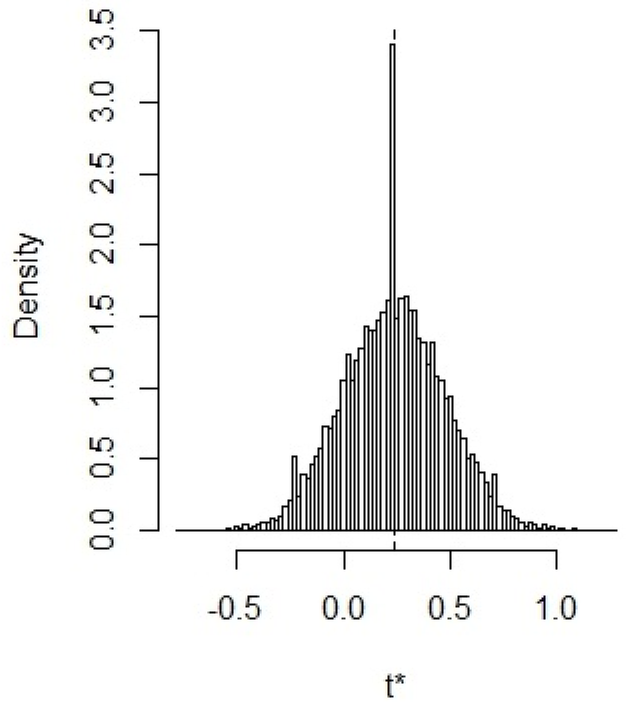
In the very first experiment design, I also made the explanation field optional, but the results (and the average amount of time spent per HIT) raised some doubts as to whether workers were just choosing at random. When the explanation field was optional, no workers provided an explanation, and the average amount of time spent on a HIT was in the range of 1-3 minutes. After making the explanation field mandatory, workers spent 13-15 minutes on each assignment. This suggests that workers were actually reading the text snippets, and in some cases maybe even clicking through to the full text of the cable. It is likely that further tweaking of the experimental design might further improve the quality of the results so that they more closely align with the intentions behind the experiment.

Mechanical Turk Results

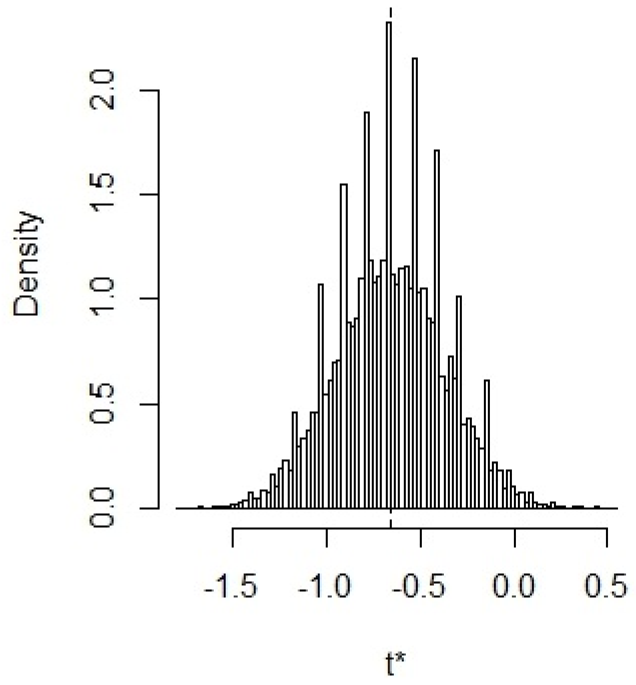
After running the two Mechanical Turk experiments, I ended up with two sets of 200 responses (50 comparisons, 4 for each comparison). To process the results, I assigned answers that said “Set A is better” with a value of -1, answers of “Set B is Better” with a value of 1, and “Both sets are equal” with a value of 0. For each comparison, I then calculated the total score by summing the values for the 4 individual answers, resulting in a score between -4 and 4.

After calculating a total score for each comparison, I loaded the data into R and used the “boot” library to apply the statistical method of bootstrapping to the data. Bootstrapping is a resampling method that makes it estimate a statistic (in our case, the mean) of a population from a sample. It is useful when the sample size is small, as in the case of our experiment, where we only have 50 comparisons. You can read more about bootstrapping here. To bootstrap the mean, I sampled 50 times (with replacement) from the original set of 50 total scores, and then calculated the mean of this new sample. I repeated this 10,000 times, and plotted the distribution of the mean for each of the two experiments below.

(TIMEOUT) vs. **(BASIC)**



(BASIC) vs. **(HIGH THRESHOLD)**



In the first graph, the mean tends toward the positive end, which shows a slight preference for Set B. In this case, Set A corresponds to **(TIMEOUT)** and SET B corresponds to **(BASIC).**

In the second graph, the mean tends toward the negative end, which shows a slight preference for Set A. In this case, Set A corresponds to **(BASIC)** and SET B corresponds to **(HIGH THRESHOLD).**

According to these results, Mechanical Turk workers have a slight preference for the basic implementation of the clustering algorithm, with no selection criteria for clusters, and a clustering threshold equal to approximately the mean plus one standard deviation.

Interpretation of Results and Further Questions
How should we interpret the results of the Mechanical Turk experiment? Can we conclude that the basic version of the clustering algorithm is definitively "better" than the two variations we chose to test? Are there additional factors (e.g. experiment design) that may have confounded the results? Does a member of the general public (e.g. a Mechanical Turk worker) define "better", "event", etc. in the same way that a journalist would? What makes a cluster useful? What makes a cluster accurate? Is there a difference between the two?

To attempt to answer these questions, I looked at the individual choices that Mechanical Turk workers made, and their justification for these choices. In many cases, workers gave reasons such as the following:

"Set B is more in depth and gives more details."
"Set A explains everything about Sudan Government."

“Set a is better because it provides more articles and in-depth information in regards to the event.”

It appears that in many cases, workers interpreted more details as meaning “better.” Although the question asked them to choose the set that “better describes an event,” often workers seemed to opt for the more detailed cluster (which was also often longer). In situations where one cluster was shorter than another, workers often chose the longer cluster, even if the shorter one was technically a more cohesive “event”:

“Set B is by far the leader here. Set A only has 1 result set while Set B is complete with a full table of results. Set B is superior in every way.”
“Set A is more superior because it has more diversity than Set B. Set B has fewer results but they both are about italy.”

In other cases, workers seemed to base their comparisons not on the quality of the clusters as descriptors of events, but on the quality or interestingness of the events that the clusters described. For instance:

“I think set A is superior because it's covering more important issues rather than set B. Set B appears to be minor issues whereas set A is going through turkey/nato issues.”
“I feel like set B has more important issues or something that has more value than Set A. Set A has a unique set of results but they don't have much impact. Set B has the full impact.”

Only a few workers seemed to grasp the actual intent behind the original experiment design:

“Both are about the UAE but neither are about a single event.”

It is clear that there was some level of misunderstanding that occurred in communicating the intent behind the experiment design to the workers. I had intended to find the algorithm which was best at identifying specific individual historical events. However, the workers’ preference for more detailed (and in many cases longer) clusters led to an apparent preference for the basic implementation of the clustering algorithm (which, by design, results in longer clusters than the other two variations).

One potential next step would be to continue adjusting the experiment design to produce more targeted results that are more closely aligned with my original intentions. However, the feedback from the Mechanical Turk workers also raises some questions about what actually makes a cluster “better.”

The definition I had previously been working off of was that one cluster is better than another cluster if it is cohesively about one event or topic--that is, if every document in the cluster “belongs.” Based on my current working definition of a “better” cluster, any cluster containing only one document is automatically (trivially) better than any other cluster of a larger size because it is more cohesive. However, there are many cases in which a longer, less cohesive cluster may be more useful or informative than a cluster containing a single document. But what level of cohesiveness is ideal? What types of clusters are the most useful from a journalistic perspective?

How should future iterations of the experiment be designed in order to capture this goal of journalistic “usefulness”? Clearly we want to eliminate situations where a worker believes that a more diverse cluster is better, or that a longer cluster is better simply by virtue of being longer. But is it necessarily wrong to say one cluster is better than another because it is more detailed? Since context is so important in journalism, might it even be desirable to privilege clusters that provide relevant context for an event?

Another potential flaw in the experiment design was the decision to randomly truncate long clusters. Workers may only prefer larger clusters up to a point--a threshold--until the clusters become unmanageably long and contain so many documents as to no longer be useful.

Lastly, there is one characteristic of a “better” cluster that is entirely overlooked by the design of the experiment. Ideally, a cluster should also not be missing any documents that should “belong.” There is perhaps no easy way to test for this characteristic, but it is worth mentioning as a potential direction for improvement.

Conclusion

The original goal of this project was to find a clustering algorithm that was “good enough” automate the processing of large document sets. I never actually got around to answering that question, because as it turns out, there are a number of other questions that need to be answered first. Namely, what qualities make an algorithm “good” in the first place? And how do we evaluate for these qualities?