WaterlooClarke: TREC 2015 Temporal Summarization Track

Ahsan Raza
University of Waterloo
Waterloo, Ontario N2L 3G1
Email: m6raza@uwaterloo.ca

Devin M. Rotondo
University of Waterloo
Waterloo, Ontario N2L 3G1
Email: drotondo@uwaterloo.ca

Charles L. A. Clarke
University of Waterloo
Waterloo, Ontario N2L 3G1
Email: charles.clarke@uwaterloo.ca

I. Temporal Summarization Track

The Temporal Summarization Track looks at providing meaningful summaries of major events and sub-events as they occur. Difficulties arise due to the unique nature of the temporal summarization task in which the corpora is constantly changing along with the known information about the event [1]. This year, the temporal summarization track consists of three tasks, two filtering and summarization tasks and an on-topic summarization task. In the filtering and summarization tasks, relevant content must be extracted from a continuous stream of documents and gradually summarized while maintaining low redundancy, latency and verbosity. In the third task, only the summarization must take place on a corpora filled with many documents related to the event [2]. The overall goal of the track is to provide a sequence of short meaningful sentence length updates over the duration of the events taking into account sentence redundancy and latency. The summaries are scored using an expected latency gain metric (ELG) in which sentences are rewarded for containing handpicked key updates (nuggets) and are penalized for redundancy, latency and verbosity [3]. This track can help users deal with all of the vast information that comes as major events progress through the use of general impersonal updates.

Our approach for this track consists of a two step process. The first stage is a data preprocessing stage which extracts the information from the supplied web documents and only retains the information considered necessary for the second stage. The second stage is the filtering and summarization stage used to find and push the relevant sentence length updates. We have applied this methodology to both the second and third task. A total of six different runs have been submitted with all of them being judged. The run scores are comparable to those in 2014, however, with a larger latency gain and lower latency comprehensiveness. The submitted task 2 runs appear to have a slightly better performance than those of task 3. However, there were no apparent differences between the task 2 or the task 3 runs.

II. Data Preprocessing

It seems to be very common to usually filter out irrelevant content when searching and summarizing data no matter the source. For the temporal summarization corpus, once the data was decrypted and decompressed into a binary form, it was taken and processed using the TREC-KBA streamcorpus tool which provides three different views on each document. In order to better clean and extract information from the documents, the clean HTML view was used. The clean HTML was processed with python-goose [4], a data processing tool which obtains the "relevant text" and document title from news articles, and python-readability [5] to get document statistics such as the number of paragraphs and images. Additionally, for UWCTSRun3 and UWCTSRun6, the cleaned HTML is transformed into a cleaner form first using python-readability before processing with python-goose. A simple way to look at the "relevant text" extraction is that it removes most of the advertisements and irrelevant data that make it harder to find the focus of the article. The last step of the preprocessing is to extract all of the sentences from the python-goose text and store them for streaming. This step was performed for each document and the preprocessed data for each document was stored in its own file.

A document metadata file was created which contained the metadata about each document, such as the document ID, stream time and document type. The metadata file allows for the documents to be looked up in an easy to process time ordered sequence. The entire data preprocessing did not take too long and would be well suited for a live temporal summarization task.

III. Approach

A. High-level Approach

Dealing with a large continuous stream of data brings with it some issues. There are several additional details that should be considered such as deciding to filter and summarize the documents as they arrive or to wait a while until you have a perhaps more representative corpus of the event. It is reasonable to say that the first relevant documents on an event immediately after it occurs will contain some important information such as the date, time, and location and perhaps initial estimates of damages. The corpus should later expand to provide more precise information in the form of updates. Additionally, if we only look at news articles then these documents have a specific structure ideal for temporal summarization. News articles often have a short and relevant sentence as the title pertaining to the newest information on the event and are then structured as an inverted pyramid in which the first paragraph explains the new

Algorithm 1 Document Summarization Algorithm

```
startTime ← query.startTime() , endTime ← query.endTime() , maxHits ← 20
previousHighWatermarkDocument ← NULL , previousDocumentsSentAsUpdate ← {}
for i = startTime, i ≤ endTime, i += MINS_5 do
    lookAheadTime ← min(i + MINS_5, endTime)
    documentsWithinTimeRange ← getDocumentsWithinTimeRange(i, lookAheadTime)
    DocumentIndex.index(documentsWithinTimeRange)
    documentHits ← DocumentIndex.queryIndex(maxHits, query.getQuery())
    if documentHits ≠ Empty then
        if previousHighWatermarkDocument = NULL then
            documentHits.retainAll(documentsWithinTimeRange)
            hitDocument ← getNonRedundantHit(documentHits, previousDocumentsSentAsUpdate);
            if hitDocument ≠ NULL then
                PUSH hitDocument
                previousHitTime ← lookAheadTime
                previousHighWatermarkDocument ← hitDocument
                previousDocumentsSentAsUpdate.add(hitDocument)
            end if
        else
            previousHighWatermarkRank ← documentHits.indexOf(previousHighWatermarkDocument)
            if previousHighWatermarkRank ≥ 0 then
                newHits ← documentHits.subList(0, previousHighWatermarkRank)
            else
                newHits ← documentHits
            end if
            newHits.retainAll(documentsWithinTimeRange)
            hitDocument ← getNonRedundantHit(newHits, previousDocumentsSentAsUpdate)
            if hitDocument ≠ NULL then
                PUSH hitDocument
                previousHitTime ← lookAheadTime
                previousHighWatermarkDocument ← hitDocument
                previousDocumentsSentAsUpdate.add(hitDocument)
            else if lookAheadTime - previousHitTime ≥ MINS_60 then
                previousHighWatermarkDocument ← NULL
            else if lookAheadTime - previousHitTime ≥ MINS_15 then
                if previousHighWatermarkRank = -1 then
                    newWatermarkRank ← length(documentHits)
                else
                    newWatermarkRank ← min(length(documentHits), currentWatermarkRank * 2)
                end if
                previousHighWatermarkDocument ← documentHits.get(newWatermarkRank)
            end if
        end if
    end if
end for
```

information on the event followed by supportive information in the later paragraphs [6]. This tends to make summarization of these types of articles a lot easier than the less structured web blogs or other such documents. Our approach attempts to build on this reasoning in creating event summaries.

In our approach we first index 20,000 documents prior to the event in order to have a reasonable document baseline for the event in task 2. These documents should all be irrelevant to the event assuming that the start time given by the user is a reasonable start time for the event. In the case of task 3, only those documents from the task 3 corpus are used which amounts to a much smaller amount of documents indexed prior to the event. Apache Lucene [7] has been used in this application for indexing, ranking and searching the documents using the Okapi BM25 ranking. Then at every five minute interval, the new data is indexed and ranked. The new data is additionally filtered prior to indexing in task 2 by removing those articles that are not news or MAINSTREAM_NEWS

from the stream. The Rocchio relevance feedback algorithm is also subsequently applied in order to complement the user query with additional search terms in order to obtain a more relevant top match. Based on this, the document summarization algorithm above is applied to decide if any of the new documents should be pushed for an update.

B. Algorithm

The document summarization algorithm is given a query object (query) —an object containing information about the event such as the name of the event, the start and end time of the event— and attempts to **PUSH** updates for that event. The document summarization algorithm processes documents in 5 minutes intervals, by looking at documents in each 5 minute interval after the event start time. It obtains the list of documents within the current interval and adds them to the document index (line 5-6). Following this, the top 20 (maxHits) documents with the highest score according to the Okapi BM25 scoring and relevance feedback are returned (line 7). If no hits were found, the algorithm will continue checking at every 5 minute interval for a hit. Alternatively, if any hits were found then there are several options that can be taken. If a document has not recently been pushed, then the top ranked hit that is not redundant with any of the previous updates will be pushed (line 9-16). This document is now set as the high watermark, which is used to determine whether to push new updates later on or not.

If a document has recently been pushed then a high watermark will exist. All of the hits that are ranked higher than the high watermark document are obtained (line 21), and from those only documents that are within the current time interval are kept (line 25). From the remaining hits, the hits that are not redundant with any of the previous updates are retained (line 26). If any documents remain then the highest ranked one is pushed (line 28). If no documents remain, then the algorithm proceeds either to decrease the high watermark document (line 34-40) or to remove the high watermark (line 32-33).

The document summarization algorithm promotes pushing an update when updates have not been pushed for a while. The algorithm attempts to push lower scoring documents as time passes even though these documents may not contain the best information on the event at the time in order to consistently provide some new information on the event. The algorithm avoids pushing updates that are redundant to previous any of the previous updates, regardless of how long ago the update was sent. To do this, it uses cosine similarity on document titles in order to determine which documents are similar to each other. When the corpus is rapidly changing with respect to the event in the query, the algorithm tries to only give out updates that are ranked higher than the previous hits. This requirement decreases exponentially as time passes and no new hits are found, allowing for lower ranked hits to be sent as updates.

Cosine similarity is used with a threshold of 0.4 in order to avoid pushing redundant updates. This threshold was determined empirically after some testing with the documents. The similarity check is based on the titles from each of the previously pushed documents with the title of the proposed documents. Additionally, a custom metric has been added to remove irrelevant documents based on their content. In order to exclude these documents, only articles with at least 3 paragraphs along with articles where number of paragraphs is greater than the number of images times 1.5 are kept. This ensures that if a sentence from a document is pushed for an update, the document is a news focused article and not an article that is primarily image or video-based.

IV. Track Runs

Each run differs in the summarization strategy that it uses or on the task in which it is ran. The summarization strategy in UWCTSRun1, UWCTSRun2 and UWCTSRun3 for task 2 correspond to UWCTSRun4, UWCTSRun5, and UWCTSRun6 in task3. UWCTSRun1(4) follow the strategy of pushing the first sentence found in each article as determined natively by python-goose. For UWCTSRun2(5), the document summarization algorithm pushed only the document headline for each of the selected documents. For UWCTSRun3(6) a combination of python-readability and python-goose is used. Python-readability parses the clean HTML and only keeps the HTML containing the main article. Then python-goose is used to extract sentences from the HTML and the first sentence is pushed. All these approaches differ in that the updates pushed for UWCTSRun2(5) will be a short concise update for each event whereas for UWCTSRun1(4) and UWCTSRun3(6), the updates will be much more detailed and perhaps more informative. Table 1 illustrates the average results for all judged runs across all queries. Bold numbers representing best run results across the different evalation measures.

TABLE I
Track run results for UWCTS run 1 to run 4.

Run ID	$EG_\tau\,(S)$	$C_\tau\,(S)$	H
— Task 2 —			
UWCTSRun1	0.1547 ± 0.13	$\mathbf{0.2065} \pm 0.09$	$\mathbf{0.1553} \pm 0.09$
UWCTSRun2	$\mathbf{0.1811} \pm 0.13$	0.1496 ± 0.08	0.1523 ± 0.09
UWCTSRun3	0.1491 ± 0.12	0.2026 ± 0.09	0.1511 ± 0.09
— Task 3 —			
UWCTSRun4	0.0751 ± 0.09	0.0583 ± 0.06	0.0571 ± 0.06
UWCTSRun5	0.0830 ± 0.12	0.0279 ± 0.04	0.0398 ± 0.06
UWCTSRun6	0.0753 ± 0.09	0.0545 ± 0.07	0.0553 ± 0.07

From Table 1, there does not appear to be any difference between the task 2 or the task 3 runs. The only apparent difference appears to be between task 2 and task 3 showing a better overall performance across all of the task 2 runs. This suggests that the algorithm may require modifications when dealing with a smaller and more focused corpus such as pushing updates more frequently.

V. Improving the Processing and Results

The document preprocessing and processing should perform adequately with a real-time continuous stream of articles. However, as the current metric limits scoring to the original tokenized sentences, the sentences that are pushed to the user

must currently be matched to those sentences for an update. This increases the amount of processing done currently for each update. Additionally, the proposed document summarization algorithm assumes that the top hits are well structured and focused primarily on the one query topic. However, in rare occasions the news articles contain information about more than one news story leading the summary to stray off-topic. Possible future considerations could be taken by further applying additional natural language processing in order to obtain a more relevant sentence for these documents.

REFERENCES

[1] Q. Guo, F. Diaz, and E. Yom-Tov, "Updating users about time critical events," in *Advances in Information Retrieval*. Springer Berlin Heidelberg, 2013, vol. 7814, pp. 483–494.

[2] J. Aslam, F. Diaz, M. Ekstrand-Abueg, V. Pavlu, and T. Sakai, "Temporal Summarization 2015 Guidelines," in *Proc. of the Twenty-Fourth Text REtrieval Conference*, 2015.

[3] J. Aslam, F. Diaz, M. Ekstrand-Abueg, V. Pavlu, and T. Sakai, "Temporal Summarization 2015 Metrics," in *Proc. of the Twenty-Fourth Text REtrieval Conference*, 2015.

[4] X. Grangier, "Python-goose," 2015. [Online]. Available: https://github.com/grangier/python-goose

[5] Y. Baburov and R. Harding, "Python-readability," 2015. [Online]. Available: https://github.com/buriy/python-readability

[6] E. A. Thomson, P. R. R. White, and P. Kitley, "Objectivity and Hard News Reporting across Cultures: Comparing the News Report in English, French, Japanese and Indonesian Journalism," *Journalism studies*, vol. 9, no. 2, pp. 212–228, 2008.

[7] Apache, "Lucene," 2015. [Online]. Available: https://lucene.apache.org/