University of Waterloo at the TREC 2013 Temporal Summarization Track

Gaurav Baruah[1], Rakesh Guttikonda[1], Adam Roegiest[1] and Olga Vechtomova[2]

[1]David R. Cheriton School of Computer Science, University of Waterloo
[2]Department of Management Sciences, University of Waterloo

Abstract

The University of Waterloo participated in the Temporal Summarization Track at TREC 2013 and submitted 8 runs for the Sequential Update Summarization Task. Methods like query likelihood ranking, pseudo relevance feedback, BM25 and cosine similarity, as well as, algorithms for passage retrieval and term expansion using distributional similarity to a set of seed words, were used for returning relevant sentences from a stream of time-ordered documents. Higher scores relative to the average score for all submitted runs were achieved on the Latency Comprehensiveness Metric (returning as many nuggets as possible), however, submitted runs performed poorly on the Expected Latency Gain Metric (speediness of updates).

1 Introduction

The Temporal Summarization Track allows researchers to investigate the problem of retrieving relevant material from a collection that is dynamic in terms of size and is ordered by time. This closely reflects real world applications in the way documents are generated (on the web) and then collected and processed by information retrieval systems. In addition, by enforcing the retrieval unit to be a sentence, the track encourages researchers to develop algorithms that not only are capable of handling high-volume document streams but are also adept at finding specific highly relevant content within the stream.

Although iterations of the Microblog Track [5] address the problem of a time ordered document stream, the documents are short blurbs of at most 140 characters. The collection used for the Temporal Summarization Track is much more diverse and contains web documents of various lengths and content types like news articles, blogs and forums. The text in the documents was also tagged with Named Entities, which could possibly be used for improving retrieval performance.

Our motivations for participating in the track were manifold. We wanted to understand: how to work with a time-ordered document stream; if the methods used for past Microblog tracks would work for Temporal Summarization; if fundamental algorithms would return relevant sentences with adequate performance on the metrics. We were also driven by the fact that the track's task of providing speedy news updates has direct real world applications.

Techniques like cosine similarity, query likelihood ranking, pseudo-relevant feedback, passage retrieval, Okapi BM25 and term expansion using distributional similarity to a set of seed words, were employed to find and rank relevant sentences from the corpus. The collection was processed and converted to a form that was flexible enough to support the application of the techniques. It was found that our approaches find relevant sentences (within a given time-frame), however, getting speedy/early relevant updates may require a different approach than the one described in this work.

1.1 Track Description

The Temporal Summarization Track at TREC 2013 [1], aims to return short relevant updates about an event from a time-ordered stream of documents. TREC 2013 is the first iteration for this track and consisted of two tasks. Task 1, Sequential Update Summarization (SUS), required finding relevant and novel updates (sentences) for an event. Task 2, Value Tracking, required estimating values for a particular attribute for an event.

A participant was required to simulate a system as follows:
Input1: a time ordered stream of documents
Input2: topic description with query (event), start

[1]`http://www.trec-ts.org/`

```
<event>
<id>1</id>
<start>1329910380</start>
<end>1330774380</end>
<query>buenos aires train crash</query>
<type>accident</type>
<locations/>
<deaths/>
<injuries/>
</event>
```

Figure 1: Example Query: "buenos aires train crash".

time, end time, type of event ∈ {accident, bombing, earthquake, shooting, storm}, attributes ∈ {deaths, displaced, financial impact, injuries, locations}.
Output:
Task 1: Sentences relevant to the query between start and end times
Task 2: Estimate of the values for the attributes in the query as time progresses.
Constraints: we cannot use corpus statistics (or external corpora) that are in the future with respect to the query start time.

Figure 1 shows a query of type accident. The "user" wants to be updated about attributes like locations, deaths, injuries for the given accident. The query spans 10 days (240 hours) starting 2012-02-22-11. A training topic "iran earthquake" (referencing the 2012 East Azerbaijan earthquake), along with relevant nuggets was provided by the track organizers.

Two metrics, `Expected Latency Gain` (ELG) and `Latency Comprehensiveness` (LC), were developed by the track organizers to measure the quality of runs. ELG (when only considering binary relevance) can be thought of as similar to traditional precision except that runs are penalized for delaying the emission of relevant updates. LC is analogous to traditional recall and it measures the coverage of relevant nuggets in a run. The University of Waterloo submitted 8 runs for the SUS task. We achieved above average performance with respect to LC for most of the runs, but performed poorly with respect to ELG.

2 The Preliminaries

Described in this section are the steps that were taken to process the KBA stream-corpus[2], the time-ordered document collection for the track. The corpus was converted to a TREC-style document formatting and

[2]Details available at `http://trec-kba.org/kba-stream-corpus-2013.shtml`

```
<doc>
<docno>...</docno>
<lang>..</lang>
<abs_url>...</abs_url>
<original_url>...</original_url>
<time>...</time>
<tagger>...</tagger>
<sentences>
...
</sentences>
</doc>
```

Figure 2: Outline of the TREC-style format of Documents.

all documents within the query time were scored using query likelihood model. Sentences were extracted for output only from those documents that had query likelihood scores above a specified threshold. The following subsections elaborate on our corpus preprocessing steps.

2.1 Corpus Collection and Preprocessing

The KBA Corpus was downloaded locally in late May 2013. Once the corpus was secured, the C++ thrift sample from `https://github.com/trec-kba/streamcorpus` was modified to allow an examination of the content provided by the Thrift format. On examination of the Named Entity Tags in the document text, it was seen that some tagged elements did not appear to correlate well with their intended meaning. Therefore, these tags were ignored during preprocessing as they could also be generated at a later stage. Further, it was decided that the document chunks would be converted from their Thrift format to a TREC-style format to facilitate easier processing and avoid the Thrift processing overhead. Figure 2 illustrates the TREC-style format that each document was converted to. Note that: `<time>` refers to the Epoch timestamp, `<tagger>` refers to the sentence tagger, and `<sentences>` is a list of the sentences in the document, such that each sentence in the Thrift format corresponds to a line in the sentences field.

2.2 Initial Wumpus Attempt

Initially, the search engine of choice was the Wumpus Search Engine [3], which implements the ranking algorithm described in Section 2.4 (as well as

[3]`http://www.wumpus-search.org`

others) and various forms of pseudo-relevance feedback (PRF). However, there were three main issues in getting the KBA corpus to work with Wumpus: (1) Wumpus would not index some documents of the type `MAINSTREAM_NEWS` and it was uncertain how such removal would affect run performance, (2) duplicate document identifiers are present in the corpus and it would have required significant preprocessing to remove them and allow Wumpus to index the collection, (3) Wumpus indexes documents in the order they were received and accordingly would either require extensive preprocessing of the hourly blocks or delaying return to the end of the hour. Due to these issues, it was decided that Wumpus should be replaced with our own solution to facilitate formulation of runs.

2.3 Hour-Wise Query Likelihood with Dirichlet Smoothing

The time-ordering of documents was a major hurdle to overcome when using the corpus. Even if an index could have been built with Wumpus, it could have been difficult to limit the collection statistics to times before the start times of each query. A further complication was that the documents in the collection, though grouped together in hours, were not ordered within the hour.

The following procedure outlines our process for generating runs:

1. Rank documents using query likelihood.
2. Extract ranked documents (above specified threshold) from the collection (Document Set S).
3. Process ranked documents in order of increasing time stamps and extract relevant sentences using various algorithms.
4. Construct runs using the extracted sentences.

To facilitate techniques like PRF or re-ranking, as well as steps 1 and 2, we converted the collection to a "reduced" form with:

i Each hour's documents reduced to their term-frequency feature vectors in one file, with each line containing features of one document.

ii Each hour's documents' metadata in a file with each line containing doc-id, source `.gz` file of the document, document timestamp and document length.

iii Each hour's vocabulary and term-frequency (`hour-term-counts`) in a file.

iv Other statistics of interest like the number of terms in the collection (l_C) up until that hour and the number of terms in each hour (l_H), were also noted.

This reduced dataset (approximately 600 GB compressed) helped us in computing (using cumulative sum of `hour-term-counts`) the collection statistics up to a query's start time and continue on to the query's end time easily. Of course, this resulted in the collection statistics being updated every hour and not for every time-ordered document in the stream as it becomes available to rank.

2.4 Computing `hour-wise` Document Scores

Language Modeling with Dirichlet smoothing (LMD) was used to score documents with respect to the query (as described in section 9.3 of Büttcher et al. [1]), as shown below:

$$score_{LMD} = \sum_{t \in q} q_t \cdot log\left(1 + \frac{f_{t,d}}{\mu} \cdot \frac{l_C}{l_t}\right) - n \cdot \left(1 + \frac{l_d}{\mu}\right) \quad (1)$$

where, t is the term, q is the query term vector, q_t is the term frequency of t in the query, $f_{t,d}$ is the term frequency of t in document d, l_C is the length of the collection (the total number of times *all* terms appear in the collection), l_t is the number of times the term t appears in the collection, and μ is the Dirichlet smoothing factor. For our experiments we set $\mu = 1000$.

Given the query, its `start_time` and `end_time`, and the reduced corpus, a `query.score` file was generated for each hour between the start and end times containing for each document in that hour: its document id, document timestamp, and $score_{LMD}$. The following algorithm was used to compute LMD and collection statistics:

```
for each hour from collectionStart_time \
              until query start_time:
  l_t += hour-term-counts(t)
  l_C += l_H
end for
for each hour from query start_time \
              until query end_time:
  for each document in hour:
    compute LMD score for document
    write out LMD score to query.score file
  end for
  update l_t and l_C for the hour
end for
```

It was found during testing that the corpus had duplicate document identifiers. Further, the content within each duplicate was different, resulting in different scores. It was decided that all documents that shared document identifiers would be ignored, even if they were relevant. Hour-wise LMD worked out favourably here because it allowed duplicate documents to be easily identified and ignored using the `query.score` files.

Once the documents were ranked for a query, all documents with $score_{LMD} > 0$ were extracted from the corpus for further processing (document set S). A score greater than zero would mean that the document contains query terms with a reasonable frequency while being of reasonable length. Note that the threshold was not set after observing the scores of all documents, or to ensure a definite number of documents in the result set. Rather, the aim was to find all documents that were likely to be relevant and hence the cut-off threshold of 0. An arbitrary score cut-off does not break the system simulation as specified by the track guidelines, whereas a fixed number of top ranked documents for every hour would.

Event	**Pattern**
Accident	"accident|calamity|casuality |disaster|hazard|mishap|pileup |setback|collision|fender-bender|smash"
Bombing	"bombing|bomb|explosion|device |explosive|charge|shell|projectile |rocket|missile|mine|homemade |terrorist|detonate|timer"
Earthquake	"earthquake|shock|fault|quake |shake|shaking|tremor|fault|temblor |quake|quaking"
Shooting	"shooting|shots|shot|fired|firing |gunfire|gun|gunning|trigger|bullet |machine|fire|terrorist|firefight |automatic|rifle|shotgun"
Storm	"storm|blast|blizzard|cyclone |disturbance|downpour|gale|gust |hurricane|monsoon|snowstorm |squall|tempest|tornado|twister |windstorm"

Table 1: Patterns issued to `egrep` for each query.

3 Runs CosineEgrep and NormEgrep

The runs in this section were intended to be baseline approaches and aimed to explore the differences in the two similarity measures. Accordingly, both runs for each hour ran `egrep` over documents in S and extracted all lines that contained any word in a list of synonyms with the event type. The list of synonyms was hand-crafted by one of the authors. Table 1 lists the queries provided to `egrep`. Note that for the submitted runs the ignore case option of `egrep` was not used. The results of using the ignore case option are reported in Table 8. Inclusion of lines returned through the ignore case option, increased the number of sentences for each topic across the board (except Topic 7) with substantial increases for Topics 3, 5, and 6, which saw 1M+ increases.

Once the sentences were collected from S, they were processed first in order of their document id and then their order within a document. A sentence was selected to be emitted for a topic if it was sufficiently dissimilar to the previously emitted sentences. Two similarity metrics were used on a feature vector comprising of the following: the frequency of each lower case character, the frequency of each upper case character, and the frequency of each digit (e.g. 0-9). This results in a feature vector of 62 elements which is the basis for the 2-norm based similarity metric.

Sperling et al. specify the 2-norm based metric in their work on identifying duplicates in legal e-discovery [6]. The formula used in this work was as follows:

$$\|x - q\| >= \gamma \|q\| \quad (2)$$

where x is the feature vector corresponding to a candidate sentence, q the feature vector for a previously emitted sentence, and γ affects how dissimilar a sentence must be before a document is emitted. Sperling et al. prescribe a γ of 0.25 or 0.5, however, in some very crude experiments with the training topic a value of 0.75 was selected as other values appeared to either be too restrictive or too permissive. Although, a more rigorous study would need to be conducted to make any conclusions.

The second similarity metric was the standard cosine similarity metric such that a cosine of greater than 0.5, which corresponds to a 60 ° between the two vectors, was used to indicate that a sentence was too similar. However, little empirical testing was done to select this value aside from some very rudimentary experiments with the training topic. It is worth acknowledging that this threshold is also very conservative with respect to the norm-based metric and that both similarity metrics are conservative in general.

For both metrics, the score of a sentence was relative to its emission, i.e., the first sentence emitted from the first document processed had the highest score and subsequent scores would be monotonically

decreasing.

4 Runs UWMDSqlec2t25 and UWMDSqlec4t50

The problem of identifying a "highly" relevant sentence from a document lends itself to the passage scoring method described in section 9.6.1. of Büttcher et al. [1]. Given the length of the collection (counting all terms) l_C, the number of times the term t appears in the collection l_t, the score for the passage can be computed as

$$score_{cover} = \sum_{t \in q'} (log(l_C/l_t)) - m \cdot log(l) \quad (3)$$

where q' is the subset (cover) of the query terms in the passage, m is the number of query terms in the passage ($|q'|$), l is the length of the passage. For the track's problem description, we consider each sentence as a passage. As was done for $score_{LMD}$, sentences with $score_{cover} \leq 0$ were ignored and considered not relevant (as described in section 2.4).

4.1 Overview of Process

This subsection describes the procedure that was followed to generate runs UWMDSqlec2t25 and UMDSqlec4t50.

For each query (event/topic),

1. Generate a background language model B for the query. This language model would be used to generate expansion terms through pseudo-relevance feedback (PRF).
2. Get top 20 documents D_h in (current) hour h, for the query.
3. Get top expansion terms E_h for hour h, using the document set D_h.
4. Use top k terms from E_h to identify relevant sentences from documents in hour $h+1$ of the document stream.
5. Remove duplicate sentences.

The parameter k sets the number of terms chosen to expand the query. In addition, only those documents that have at least c (query + expansion) terms in them where chosen. This helps to prune documents with low number of query and/or expansion terms. It was the case that $c = 2$ and $k = 25$ for run UWMDSqlec2t25 and $c = 4$ and $k = 50$ for run UMDSqlec4t50.

We chose to generate expansion terms every hour in order to capture the possible changes in the content of the documents for that hour. For instance, for the training event "iran earthquake", the first few hours could be about the fact that the earthquake occurred, at a particular location, with a particular magnitude. The next few hours may focus on the updates on number of killed, injured or displaced people, updates on magnitudes of aftershocks and regions affected, along with the response from the government and rescue units. Later news may be about foreign aid, damages to property. Later still, we may find articles about the economic impact of the earthquake and government plans for rebuilding and resettlement. As important updates change every hour, the top expansion terms for each hour may be expected to change as well.

4.2 Query Term Expansion for Ranking Sentences

Pseudo-relevance feedback was performed with the top 20 documents (as scored by $score_{LMD}$) considered relevant (prescribed in Büttcher et al. [1], section 8.6.2). We tried various approaches for query expansion and term selection based on (i) choosing terms from the (pseudo-) Relevance Model (section 7.3.2 of Croft et al. [4]), (ii) choosing terms using KL divergence (Carpineto et al. [2], section 9.4 of Büttcher et al. [1]), and (iii) choosing terms based on IDF-like weights (a modification of the method described in section 8.6.1 of [1]).

We select a term t for expansion, if it has a high value for

$$n_{t,r} \cdot w_t \quad (4)$$

where, $n_{t,r}$ is the number of times the term appears in relevant documents and $w_t = log(l_C/l_t)$. l_C is the length of the collection up until the current hour in the time-ordered document stream and l_t is the number of times the term occurs in the collection up until the current hour.

4.3 The Background Language Model

Ideally, the collection model should be the background model against which we compute w_t. However, the document collection is time-ordered and as we go further along the collection, we should get better estimates of a term's importance. In the interest of computation time, we adopted an approximate background model B rather than the whole collection model. Given that the queries spanned 10 days, we started building our background model 20% time (2 days) in advance. For hour h, we scored all documents using query likelihood ($score_{LMD}$), and chose

the top 20 documents. The terms (and term counts) from these documents were added to B. l_B, the total number of terms in B was updated with the total number of terms in hour h. For hour $h+1$, the terms and terms counts were cumulatively added to B. This process resulted in a background model that grows with time which may produce better expansion terms as time progresses.

As we encounter new terms for every hour processed, to calculate the true l_t for each new term, we need to go back to start of collection and cumulatively add their hour-term-counts, because of our hour-wise index layout (section 2.3). In the interest of time, we approximated $w_t = log(l_B/l_{t_B})$, where l_{t_B} is the number of times the term t occurs in B. In hindsight, this may have been a poor approach and we should have used the complete collection model. It may be useful to compare our formulation w_t with the Robertson/Spärk Jones weighting formula or BM25 weights for terms, now that we have the test collection for the track.

query type	**seed query**
Generic All Attributes (GAA)	earthquake injuries locations deaths displaced financial impact
Generic Single Attributes	earthquake injuries earthquake locations earthquake deaths earthquake displaced earthquake financial impact
Specific All Attributes (SAA)	iran earthquake injuries locations deaths displaced financial impact
Specific Single Attributes	iran earthquake injuries iran earthquake locations iran earthquake deaths iran earthquake displaced iran earthquake financial impact

Table 2: Choice of queries to generate expansion terms.

4.4 Choosing a Query for Term Expansion

The problem definition for the track specifies upto five attributes of an event for which we need to find relevant sentences. Therefore, we are now in a position to choose a "seed" query that would help us to get good expansion terms through PRF. Table 2 lists the various possible seed queries for the training topic (query: "iran earthquake", type="earthquake"). For each of these queries, top 20 documents were identified, relevance models were created and expansion terms were generated.

We observed that top expansion terms changed every hour (See Tables 3 and 4 for an example on the changes in expansion terms for hours 2012-08-11-18 and 2012-08-11-21 respectively). Queries of type "specific all attributes" and "generic all attributes" were found to give better expansion terms across hours. Recall that we use the expansion terms obtained in hour h to score sentences in hour $h+1$ (section 4.1). The idea was to capture relevant sentences even as the content about the topic/event changes with time, i.e., each hour (or time period) may report about different aspects of the event and we wanted to capture terms relevant to the time period.

4.5 Variability in Expansion Terms

It was observed that the (top 20) documents obtained by PRF do not necessarily contribute good expansion terms when the seed query is of type "specific all attributes" (SAA). However, the query of type "generic all attributes" (GAA) does provide some terms related to the event across hours. Tables 3 and 4 show the top 10 terms for query term expansions with SAA and GAA queries for the training topic "iran earthquake".

In order to maintain a balance between the generic and the specific queries, we turned to Reciprocal Rank Fusion (RRF) (Cormack et al. [3]) to fuse the list of expansion terms generated by seed queries of type SAA and GAA. The third column in Tables 3 and 4 shows the top 10 terms of the RRF fused list of expansion terms.

Table 3 shows an example of the SAA query expansion terms showing specific locations (Haris, Varzaqan, Ahar) which are not top terms for expanded GAA query. Whereas, Table 4 shows an example where GAA query expansion terms may be useful. Terms like ANSS[4], RMSS[5], shakemap[6], may help provide further detailed information related to earthquakes in general.

[4]ANSS: Advanced National Seismic System `http://earthquake.usgs.gov/monitoring/anss/`

[5]"RMSS: root-mean-square travel time residual in seconds" - source `http://earthquake.usgs.gov/earthquakes/glossary.php`

[6]Shakemap: A map that presents information on the shaking of ground rather than epicenter and magnitude. `http://earthquake.usgs.gov/research/shakemap/`

Top 10 expansion terms for training query type		
GAA	**SAA**	**RRF-fused**
earthquake	injured	earthquake
quake	earthquake	injured
magnitude	magnitude	magnitude
injured	haris	quake
hundreds	varzaqan	killed
killed	ahar	hundreds
seismic	quake	haris
iran	killed	ahar
earthquakes	hundreds	varzaqan
northwestern	iran	iran

Table 3: **Hour 2012-08-11-18**: Expansion Terms for the training topic "iran earthquake", generated using seed queries of type Generic All Attributes (GAA) and Specific All Attributes (SAA).

Top 10 expansion terms for training query type		
GAA	**SAA**	**RRF-fused**
earthquake	earthquake	earthquake
magnitude	iran	seismogram
seismogram	villages	anss
anss	magnitude	nsmp
nsmp	least	rmss
rmss	northwestern	magnitude
recenteqsww	earthquakes	recenteqsww
crustal	tv	crustal
shakemap	injured	shakemap
seismologist	news	seismologist

Table 4: **Hour 2012-08-11-21**: Expansion Terms for the training topic "iran earthquake", generated using seed queries of type Generic All Attributes (GAA) and Specific All Attributes (SAA).

4.6 Weighting Term Importance with RRF Scores

To account for the revised ranking of terms we modified equation 3 as

$$score_{cover} = \sum_{t \in q'} \left(log \left(\frac{l_C}{l_t} \times rrf(t) \right) \right) - m \cdot log(l) \tag{5}$$

where, $rrf(t)$ is the RRF value for the term t computed as

$$rrf(t) = \sum_{i} \frac{1}{k + r_i(t)} \tag{6}$$

where, $1 \leq i \leq$ number of lists to fuse, $r_i(t)$ is the rank of the term t in list i, and prescribed parameter value of k is 60.

The revised formula 5 for $score_{cover}$ drives down the IDF-like weights for all terms, but does so based on the RRF score for the term. This ensures that terms having higher relevance/importance to the query contribute more to the sentence score. The original query terms were treated as having the highest possible RRF value (1/61), so that their importance is not devalued as compared to the expansion terms.

4.7 Building the Run

Many duplicate documents were found (and hence duplicate sentences) in the stream corpus and as such we need a deduplication module to eliminate duplicate sentences. We simulate deduplication (while adhering to the track's guidelines) by using the Unix commands *sort* and *uniq*. All updates were sorted first by sentence and then by time. Then exact duplicate sentences were eliminated using *uniq*. In a real system, one may keep all updates in a map or lookup table in order to avoid emitting exact duplicates of previous updates.

While working with the training topic, it was seen that there are many irrelevant sentences that score high on $score_{cover}$, due to the presence of high ranking expansion terms in them. To mitigate this problem, it was reasoned that a "relevant" document may have relevant material spread throughout its content. Therefore, $score_{cover}$ for a sentence could be weighted with the $score_{LMD}$ for the document from which the sentence was selected. In effect, the sentences from low ranking documents would receive a lower confidence rating and sentences from high ranking documents would receive a higher confidence rating, with $score_{cover} \times score_{LMD}$.

Finally, as described in section 4.1, the two parameters k and c need to be tuned. Table 5 shows the number of unique sentences obtained for various setting of the parameters. We opted to emit more number of sentences in order not to miss too many updates.

5 Runs rg1, rg2, rg3 and rg4

The *rg* runs (rg1, rg2, rg3, rg4), were generated from the set of documents scored by query likelihood model described in section 2.4. Sentences from document set S were used to generate the runs. In this section, the methods used to generate the runs are reviewed.

c	k	unique sentences
2	**25**	**7425**
2	50	7699
2	100	5495
4	25	6111
4	**50**	**7073**
4	100	5446
10	25	3488
10	50	5047
10	100	5700

Table 5: Number of unique updates for the topic "iran earthquake" obtained for values of parameters c and k. Rows in bold represent parameter values for submitted runs.

5.1 Scoring sentences

Okapi BM25 (eq.7) was used as the ranking function to score the sentences in the document with respect to the query.

$$score_{sentence} = \sum_{t \in q} q_t \times \frac{f_{t,d}(k_1+1)}{k_1((1-b)+b(l_d/l_{avg}))+f_{t,d}} \times w_t \quad (7)$$

where, $f_{t,d}$ is the frequency of term t in sentence d, w_t is the IDF of term t, l_d is length of sentence, l_{avg} is the average sentence length (found to be 34) and parameters $k_1 = 1.2, b = 0.75$ (as prescribed by Büttcher et al. [1]).

Since the query terms are fewer and the sentences are relatively short ($l_{avg} = 34$) compared to document lengths, query expansion techniques were used to improve the recall of the results. Hence, the technique used for finding expanded words forms one of the key components of the approach and is outlined in the next section.

5.2 Distributional Similarity Based Term Expansion (DSTE)

As seen earlier in section 1.1, a query can belong to one of the following event types: accident, bombing, earthquake, shooting, storm. For each of the event types, seed words (around 30 words per event type) were found manually from prior Wikipedia[7] articles of each event type. A list of training topics was also created, one for each event type.

Top K (=10000) sentences were then retrieved from S for each training topic, using the BM25 score and the seed words list. These top K sentences, along with the seed words, are given as input to an algorithm that uses distributional similarity to find expansion terms [7]. These newly identified expansion words along with the initial seed words and query terms, constitute the expanded words list (q in equation 7) for calculating the $score_{sentence}$. Table 6 shows examples for seed words and their expansions for the training topic ("iran earthquake").

[7] www.wikipedia.com

seed word	Top 10 expansion terms generated with DSTE
quake	earthquake tremor disaster magnitude aftershock temblor toll damage province death
damaged	destroyed killed left hit injured struck wounded leveled brought died
cities	counties areas towns regions provinces parts villages people states residents
assistance	aid help food relief money work medicine team sympathy water
disaster	earthquake quake emergency relief tremor crisis aftershock catastrophe development region

Table 6: Examples of seed and expansion terms for *rg* runs

5.3 Modified score function

Similar to the previously discussed runs (section 4.7), it was found that $score_{sentence}$ alone isn't sufficient to judge the relevance of a sentence with respect to the query. It was observed that there were quite a few sentences which contain the expanded words, but are from very low scoring documents which are not relevant to the query.

For example, there were earthquakes in some parts of China during the same time as the Iran earthquake, which is the training topic. With only the sentence scores, the sentences from documents related to the earthquake in China would be as relevant as the sentences from Iran earthquake, which is unlikely to be the case. Also, the document score of a document related to the Chinese earthquake is much smaller than that of a document related to the Iranian earthquake. In an attempt to combat this, we combine the $score_{sentence}$ and $score_{LMD}$ to ensure that we return relevant sentences from relevant documents. It was seen that the simple product of these two scores

would in general lead to better updates, i.e.,

$$score_{combined} = score_{sentence} \times score_{document} \quad (8)$$

This function needs to be evaluated with a series of experiments which we will pursue as a future work.

5.4 Sentence Selection Criteria

To avoid redundancy in updates, and to improve their quality, we need to shortlist sentence updates as well as avoid duplicate sentences.

A rigid cutoff for $score_{sentence}$ to shortlist sentences was not used due to the diversity in the documents returned every hour. Hence, an incremental cutoff score ($score_{cutoff}$) for $score_{sentence}$ was used every hour to decide whether a sentence should be included in the list of updates.

The following algorithm was used to generate the list of updates:

```
S_h = 5000; //max num of sentences per hour
score_cutoff = 0; //cutoff=0 for first hour
updatelist[]  //list of updates

for every hour between the start     \
                    and end timestamps
  count = 0;
  score[];  //sentence scores current hour

  for all the sentences in the hour:
    if score_sentence > score_cutoff:
      add sentence to updatelist;
      score[count] = score_sentence;
      count++;
    end if
  end for
  if count > S_h: //update score_cutoff
    sort score[] desc
    score_cutoff = score[S_h];
  end if
end for

return updatelist
```

For the first hour, all sentences were included in the update set. For subsequent hours, only sentences with $score_{sentence} > score_{cutoff}$, i.e. minimum score cutoff from previous hour, were added to the update set. Similarly, an incremental minimum cutoff score was computed for documents, which is dependent upon D_h number of documents every hour. For a sentence to be included in updates, both the sentence score cutoff and document score cutoff should be passed.

RunID	Sentences per hour (S_h)	Documents per hour (D_h)
rg1 and rg2	5000	3000
rg3 and rg4	1000	500

Table 7: S_h and D_h for *rg* runs

RunID	Expected Latency Gain	Latency Comprehensiveness
CosineEgrep	0.0104	0.0179
CosineEgrepIgnoreCase	0.0146	0.0130
NormEgrep	0.0011	0.0611
NormEgrepIgnoreCase	0.0010	0.0534
UWMDSqlec2t25	0.0173	0.5375
UWMDSqlec4t50	0.0176	0.5304
rg1	0.0205	0.5705
rg2	0.0218	0.5624
rg3	0.0261	0.5063
rg4	0.0275	0.5165
Average All Runs	0.0599	0.2996

Table 8: Average ELG and LC across Topics.

The deduplication step kicks in while adding a sentence to the update list. If more than 90% of the words in the current sentence are covered in any of the previous update sentences, then it was discarded as a duplicate.

Table 7 lists the the number of sentences and documents shortlisted for each *rg* run. The incremental minimum cut-off scores change every hour based on the score of the S_h^{th} sentence of the previous hour.

Run **rg2** differs from **rg1** in that, the length of the sentences was normalized. While testing on the training topic, it was observed that the sentence scores alone do not account for the verbosity of the sentence, even with the BM25 score (after changing b parameter). Due to this, the sentence score was multiplied by $log(1+\frac{l_{avg}}{l_d})$; as explained in document length normalization in Büttcher et al. [1]. Similar to **rg2**, **rg4** performs sentence length normalization, whereas **rg3** does not. Runs **rg2** and **rg4** try to increase the ELG of the results by reducing the penalty for verbosity.

6 Discussion of Results

While neither of the baseline runs (CosineEgrep and NormEgrep) performed particularly well, this is not overly surprising given that both are quite conservative in the number of sentences emitted per topic; approximately 12 on average for CosineEgrep and 151

for NormEgrep. This disparity makes it apparent as to why NormEgrep likely has a higher LC than CosineEgrep. However, even though both emit relatively few sentences per topic the ELG is quite low and likely stems from the simplistic sentence selection measure. It is interesting that introducing more sentences with `egrep`'s ignore case option decreases the performance of the norm-based similarity measure and likely has to do with the introduction of non-relevant sentences.

The performance of runs UWMDSqlec2t25 (0.5375) and UMDSqlec4t50 (0.5304) with respect to the LC metric, is above the average reported amongst all submitted runs to the track (0.2996). In addition, the maximum scores for these runs (0.979 and 0.9872 respectively) are close to the reported maximum of 0.999. Both these runs performed poorly with respect to the ELG metric. This could be in part because of the differences in the content of the relevant documents as time progresses. Comprehensive (news) articles (or documents) would score higher with $score_{LMD}$, however they would likely occur later in the time-ordered document stream. The confidence rating of $score_{cover} \times score_{LMD}$ would give more importance to sentences from such articles.

Table 8 shows that high recall (LC) is achieved in all *rg* runs, which can be attributed to the expanded words for the query which were obtained using the DSTE algorithm (section 5.2). On the other hand, the *rg* runs have performed poorly with respect to ELG, which could be because of the weak deduplication algorithm, and due to the penalty of missing good sentences in low scoring documents which are early. Improvements in these two areas may help in better performance as measured by ELG for the *rg* runs.

7 Conclusions

Participation in the Temporal Summarization Track has provided an opportunity to explore various approaches to the task of identifying relevant sentences in a timely manner. Many of the runs submitted performed quite well with respect to Latency Comprehensiveness (i.e. recall), which may be attributed to the use of standard retrieval techniques. However, all our runs performed poorly with respect to Expected Latency Gain and methods of improving this while not decreasing Latency Comprehensiveness remain to be investigated.

8 Acknowledgements

We thank Mark Smucker for his helpful feedback and advice. This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET:www.sharcnet.ca) and Compute/Calcul Canada, and was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), and in part by the Google Founders Grant, and in part by the University of Waterloo. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors.

References

[1] S. Büttcher, C. L. A. Clarke, and G. V. Cormack. *Information retrieval : Implementing and Evaluating Search Engines*. MIT Press, Cambridge, Mass., 2010.

[2] C. Carpineto, R. de Mori, G. Romano, and B. Bigi. An information-theoretic approach to automatic query expansion. *ACM Trans. Inf. Syst.*, 19(1):1–27, 2001.

[3] G. V. Cormack, C. L. A. Clarke, and S. Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 758–759, New York, NY, USA, 2009. ACM.

[4] B. Croft, D. Metzler, and T. Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley Publishing Company, USA, 1st edition, 2009.

[5] I. Ounis, C. Macdonald, J. Lin, and I. Soboroff. Overview of the trec 2011 microblog track. In *NIST Special Publication 500-296: The Twentieth Text REtrieval Conference Proceedings (TREC 2011)*, 2011.

[6] M. Sperling, R. Jin, I. Rayvych, J. Li, and J. Yi. Similar document detection and electronic discovery: So many documents, so little time. In *DESI V Workshop at ICAIL 2013*, June 2013.

[7] O. Vechtomova. A semi-supervised approach to extracting multiword entity names from user reviews. In *Proceedings of the 1st Joint International Workshop on Entity-Oriented and Semantic Search*, JIWES '12, pages 2:1–2:6, New York, NY, USA, 2012. ACM.