NTNU – Trondheim
Norwegian University of
Science and Technology

# Temporal Summarization of Time Critical Events

A system for summarizing events over time in
a continuous stream of documents

**Håvard Lund Eidheim**

Master of Science in Informatics
Submission date: June 2015
Supervisor: Kjetil Nørvåg, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

# Abstract

When a critical event occurs, it is often challenging to get a clear overview of the event. Many sources of information such as newspapers and Twitter often contains a lot of information that is not relevant and requires a lot of reading to gain an overview of the situation.

Real time temporal summarization systems is an effort to address this problem, by analysing available information in real time and producing a continuous summary as new information becomes available.

The result of this thesis is a real time temporal summarization system. The system has been tested according to the requirements of the TREC 2014 Temporal Summarization track. The results have been evaluated using standard evaluation criteria and compared to the results of the 2014 participants.

The system performed comparable to the best teams from 2014 and it is likely that the techniques used could be improved further by considering more aspects of the available information when summarizing an event.

# Sammendrag

Når en katastrofe infreffer er det ofte vanskelig å skaffe seg et godt overblik. Mange vanlige informasjonskilder slik som aviser og Twitter inneholder mye informasjon som ikke er relevant for hendelsen, og man må ofte lese mye for å få et riktig bilde av hva som skjer.

Sanntids, temporal summarization systemer sikter på å kunne løse dette problemet ved å kontinuerlig generere oppdateringer om en gitt hendelse ettersom ny informasjon blir tilgjengelig.

arbeidet med denne oppgaven har resultert i et slikt system. Systemet er testet etter kravene som er stillt for deltagelse i TREC 2014 Temporal Summarization. Resultatene som er oppnådd av systemet har blitt evaluert etter kriterier definert av TREC, og resultatene har blitt sammenlignet med resultatene oppnådd av deltakerne i 2014.

Systemet har oppnådd resultater som er sammenlignbare med de beste lagene fra 2014.Det er sannsynlig at systemet kan forbedres ytterligere ved å vurdere flere aspekter ved informasjonen som analyseres etter hvert som oppdateringer genereres.

# Preface

This thesis is the product of my final year at the Norwegian University of Science and Technology in Trondheim.

This has been a challenging year and it has required a lot of patience, experimentation and intuition in order to understand systems created by teams from all over the world.

I would like to thank my Supervisor Kjetil Nørvåg for helping me select a topic, as well as his guidance and input during the work on my thesis. I would also like to thank my Friends at friends at Fiol, Mari, Joakim and Audun for our many conversations about books, life and work.

# Contents

**1 Introduction** **1**
- 1.1 Motivation . . . . . 1
- 1.2 Goals and Problem Definition . . . . . 2
- 1.3 Contribution . . . . . 4
- 1.4 Outline . . . . . 4

**2 Introduction to Summarization and Temporal Summarization** **7**
- 2.1 Automatic Summarization . . . . . 7
- 2.2 Topic Detection and Tracking . . . . . 8
- 2.3 Introduction to Temporal Summarization . . . . . 8
- 2.4 Real Time Temporal Summarization Example . . . . . 9

**3 Related Technologies** **13**
- 3.1 Information Retrieval and the Vector Space Model . . . . . 13
- 3.2 Novelty and Redundancy Detection . . . . . 15
- 3.3 Query Expansion and Modelling . . . . . 16
- 3.4 Sentence Information Retrieval and Novelty Detection . . . . . 17

**4 TREC Temporal Summarization** **19**
- 4.1 TREC Temporal Summarization . . . . . 20
- 4.2 Temporal Summarization General Procedure . . . . . 20
- 4.3 Corpus . . . . . 21
- 4.4 Evaluation Events . . . . . 22
- 4.5 Results format . . . . . 23
- 4.6 Evaluation and Metrics . . . . . 24
  - 4.6.1 General Definitions and Functions . . . . . 25

4.6.2 Expected Gain . . . . . . . . . . . . . . . . . . . . . 26
4.6.3 Comprehensiveness . . . . . . . . . . . . . . . . . . 26
4.6.4 Latency Discount . . . . . . . . . . . . . . . . . . . 27
4.6.5 Verbosity Normalization . . . . . . . . . . . . . . . 27
4.6.6 Combined Metric . . . . . . . . . . . . . . . . . . . 28

**5 TREC Temporal Summarization 2013 and 2014 29**
5.1 2013 Results . . . . . . . . . . . . . . . . . . . . . . . . 29
5.2 2013 General Discussion . . . . . . . . . . . . . . . . . 30
5.3 2013 Highlight: ICTNET . . . . . . . . . . . . . . . . . 33
5.4 2014 Results . . . . . . . . . . . . . . . . . . . . . . . . 34
5.5 2014 General Discussion . . . . . . . . . . . . . . . . . 35
5.6 2014 Hightlight: CUNLP . . . . . . . . . . . . . . . . . 36
5.7 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . 37

**6 Temporal Summarization Test Framework 39**
6.1 System Overview . . . . . . . . . . . . . . . . . . . . . . 39
6.1.1 Preprocessor . . . . . . . . . . . . . . . . . . . . . . 39
6.1.2 Generic Text Corpus and TF-IDF Index . . . . . . . . 40
6.1.3 Corpus Reader . . . . . . . . . . . . . . . . . . . . . 40
6.1.4 Document Processors . . . . . . . . . . . . . . . . . 40
6.2 Preprocessing . . . . . . . . . . . . . . . . . . . . . . . 41
6.3 System Structure . . . . . . . . . . . . . . . . . . . . . 42
6.4 Document Stream . . . . . . . . . . . . . . . . . . . . . 43
6.5 Text Index and News Corpus . . . . . . . . . . . . . . . 44
6.6 Summarizer Interface . . . . . . . . . . . . . . . . . . . 45
6.7 Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . 46

**7 TREC Temporal Summarization Solutions 47**
7.1 Baseline Design . . . . . . . . . . . . . . . . . . . . . . 47
7.1.1 Parameters . . . . . . . . . . . . . . . . . . . . . . . 48
7.1.2 Pseudocode . . . . . . . . . . . . . . . . . . . . . . . 49
7.1.3 Results . . . . . . . . . . . . . . . . . . . . . . . . . 50
7.2 Event Model Design . . . . . . . . . . . . . . . . . . . . 52
7.2.1 Parameters . . . . . . . . . . . . . . . . . . . . . . . 54
7.2.2 Pseudocode . . . . . . . . . . . . . . . . . . . . . . . 54
7.2.3 Results . . . . . . . . . . . . . . . . . . . . . . . . . 54
7.3 Evolving Event Model Design . . . . . . . . . . . . . . . 55

7.3.1 Parameters . . . . . . . . . . . . . . . . . . . . . . . . . . 57
7.3.2 Pseudocode . . . . . . . . . . . . . . . . . . . . . . . . . 58
7.3.3 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59

**8 Evaluation of Results 61**
8.1 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . 63

**9 Conclusion 65**

**Appendices 71**

**A System Guide 1**
A.1 Preprocessing . . . . . . . . . . . . . . . . . . . . . . . . . . 1
A.2 Temporal Summarization System . . . . . . . . . . . . . . . . 2
A.3 Evaluation Script . . . . . . . . . . . . . . . . . . . . . . . . 3

# Chapter 1

# Introduction

Temporal summarization is the problem of summarizing information over time. The problem have sprung from work on topic detection and tracking which is the problem of identifying information topics and tracking them over time [21][2]. A problem with topic detection and tracking is that topics often grow large with time, making then unwieldy for practical purposes. In many cases it would be more practical to receive a summary of a certain topic rather than everything that have ever been said about it.

A sub-problem of temporal summarization is real time temporal summarization. This involves monitoring topics as they unfold and continually producing a summary. This problem that will be addressed in this thesis.

## 1.1 Motivation

During time critical events such as earthquakes or airplane crashes, it is often hard to get an updated overview of the event in a short amount of time. A good way to get up to date information would be to get a stream of sentence-length updates about the situation as it develops. If an earthquake occurred in Trondheim, an update feed could look something like the list below.

- Earthquake registered in Trondheim.
- The quake had a strength of 5.0 on the Richter scale.

- There have been reports of Power outage in the city centre.
- There is no reports of casualties people.
- According to sources in the Norwegian Seismic Array, there are no danger of subsequent quakes.

Most people would probably read their favorite online newspaper in search for news of the event but newspapers will most likely also contain articles that are not relevant to the event in question. To find the information the user is looking for she or he will have to find relevant articles and read substantial portions of this article to extract desired information. A relatively good approach would be to read just the headlines, as these often contain the most important information but this could cause you to miss important details. Many news papers also have a tendency to make vague titles to make as many users as possible click the actual article to increase ad revenue. Another important reason that newspapers and similar is not an optimal source of up to date relevant information is that the content will have to be produced by humans.Not all newspapers have access to direct sources or correspondents, meaning that they will have to wait for other newspapers to write about it first. This might introduce significant delays depending on the resources and sources of the newspaper.

Another approach to getting updates about an event is to follow relevant twitter hashtags or profiles. The problem with twitter hashtags in this context is that everyone can post about a topic, which generates noise and potentially ruins the value of the stream.

One way of solving this problem is a system that automatically process documents that are published and use them to generate a feed of information about an unfolding event. This system would be a real time temporal summarization system.

## 1.2 Goals and Problem Definition

This thesis has three goals.

1. Create a real time temporal summarization system based on research of other system and related technologies.

2. To test different approaches to the problem and evaluate and evaluate how the different approaches perform.

3. To compare the results of the system with the results of comparable systems to evaluate the methods used.

To compare a system with other system, a standardised problem, data set and evaluation criteria are needed. This part of the thesis is based on the TREC 2014 Temporal Summarization track. TREC stands for Text REtrieval Conference and is a yearly conference that publish a number of different tasks that participants try to solve in preparation for the conference. Each track define a problem and metrics for evaluating the systems submitted for evaluation. The Temporal summarization track is based on the problem defined in the paper *Updating Users About Time Critical Events*[11] by Qi Guo et al.

The temporal summarization track aims create and evaluate temporal summarization systems. The track provides guidelines, metrics, example events and a data set that can be used to create and evaluate these systems. 2013 was the first year for this track and it was run again in 2014. Because the submission deadline falls between academic years the results of this thesis will not be submitted for official evaluation. This should not be much of a problem however, because the evaluation criteria is based on standard TREC evaluation and publicly available, this way any system that follows the specified formats and uses the proper data sets can be compared to other systems following the same guidelines. The approaches used by the participants of the track and the results they achieved will be used as a foundation for this thesis. Because the results are published, it is possible to analyse the performance of the different approaches and use this as a guide when creating a system.

Some central questions that will be have to be considered through working with this thesis are:

1. What are important considerations when designing a real time temporal summarization system?

2. What are the advantages and disadvantages of different approaches?

## 1.3 Contribution

The results of the work and research done during this thesis is a real time temporal summarization system conforming to the requirements of the TREC temporal summarization track. The techniques used to create the system was evaluated and compared with the results of the participants of the 2014 run of the track.

## 1.4 Outline

The thesis is divided into 9 chapters. The first 5 chapters introduce the problem of real time temporal summarization and the practical aspects of the TREC Temporal Summarization track. Chapter 5 and 6 presents the solutions created for the TREC temporal summarization track and the results achieved with the different approaches. Chapter 8 and 9 seek to evaluate the results achieved in this thesis and to provide a summary of the answers to the central questions posed in the introduction.

Below is a list of the chapters in this thesis and a short explanation of what they will contain.

1. **Introduction** - Introduction to temporal summarization and presentation of the questions that will be answered in the thesis.

2. **Theoretical background** - Overview of techniques that are related to temporal summarization or techniques that can be used when constructing a temporal summarization system.

3. **Introduction to temporal summarization** -

4. **TREC temporal summarization 2014** - Detailed description of the TREC temporal summarization 2014 track and description of the data set and evaluation metrics used for the track.

5. **TREC temporal summarization past results** - Overview of the results achieved by participants of the 2013 and 2014 TREC temporal summarization tracks and discussion of the methods used.

6. **Temporal summarization framework** - A description of the overall framework and the different parts of the temporal summarization system created for the 2014 track.

7. **Temporal summarization algorithms and results** - A description of each of the approaches used to

8. **Evaluation of results** - Evaluation of the results achieved by the system that was created compared with the participants of the 2014 track.

9. **Conclusion** - Conclusion and summarised answers to the questions posed in the introduction.

# Chapter 2

# Introduction to Summarization and Temporal Summarization

Temporal summarization can be described as the problem of automatically summarizing information over time. An example can be to generate a summary of a debate that spanned some amount of time. This chapter gives a general description of temporal summarization with a real time example and discuss some work on temporal summarization outside the TREC track.

## 2.1 Automatic Summarization

Automatic summarization is the problem of automatically summarizing information, usually in the form of a document. One of the earliest works of temporal summarization attempted to create a system that could automatically generate abstracts based on a research paper [14].

Another form of automatic summarization is multi document summarization. Multi document summarization aims to create summaries of a large amount of documents covering some topic. A study by Kathleen McKeown et al. [17] showed that both automated summaries and handwritten summaries helped when people were asked to gather facts from several different news documents concerning a specific topic. Multi document summarization is more challenging than single document summarization because systems also have to consider what documents to use for the summary.

There are two ways of creating summaries, abstraction based and extraction based summarization. Abstraction based summarization is to create text that summarize other text. This is generally they way summaries are produced by humans, we read some information and write a shorter piece covering the essential points of the original information. Extraction based summarization is to extract information from a document or a set of documents that cover the most important points of information. Because text generation is a very difficult problem by itself, most summarization research is focused on generative summaries. Systems that use extraction based summaries must decide what information to extract in some way. To decide what to extract, systems use different heuristics to decide what information is important to include in a summary. A summary could consist of a subset of documents that have been deemed to be influential and have a lot of citations, or it could be a set of sentences extracted from a single or multiple documents.

Possible applications of automatic summarization technology include summarizing news stories, compiling information from a large set of documents into a single document or generate a summary of a topic.

## 2.2 Topic Detection and Tracking

Topic detection and tracking is the problem of detecting events in a continuous stream of documents and tracking them over time [1]. Topic detection and tracking is useful for following stories over time or to automatically categorize documents. It can also be used to detect the first story of an event.

## 2.3 Introduction to Temporal Summarization

Temporal summarization can be seen as a combination of the problem of multi document summarization and topic detection and tracking. A temporal summarization system needs to be able to track events as well as summarize the documents that belong to this event.

A system made for summarizing topics can be real time or retroactive. Real time temporal summarization systems produce a summary gradually as time goes on and more documents become available. Retroactive summarization systems produce a summary once at a given time and considers all previous documents at the same time. The later have the advantage of having full knowledge of the topic or event at the point of summarization. This lets your system take advantage of features like citation data or hotlinking from other documents.

This problem was first defined in a paper by James Allan et al. [2]. The authors define the problem of temporal summarization and metrics that should be used to evaluate such systems. This paper lays the foundation for the later TREC Temporal Summarization Track.

Outside of the TREC track there have been some experiments with different takes on the temporal summarization problem. In a paper by Ruben Sipos et al. [20] the authors experiment with solutions that does not produce a summary in the traditional sense. Instead, they attempt to identify landmark documents and the spread of novel ideas and author influence throughout a corpus.

## 2.4 Real Time Temporal Summarization Example

Below is an example of how a real time temporal summarization system could work. Our example system receives a query from a user that describes an ongoing event. The system then processes an incoming stream of documents and outputs updates that are relevant to the event as it unfolds.

In this example, we will use a simple notion of document relevance. If the title of the document contains all the words in an event query, we consider that document to be relevant to the event. The same criteria is used for identifying and extracting interesting updates from the documents. Sentences are scanned and sentences containing all words in the query are assumed to be

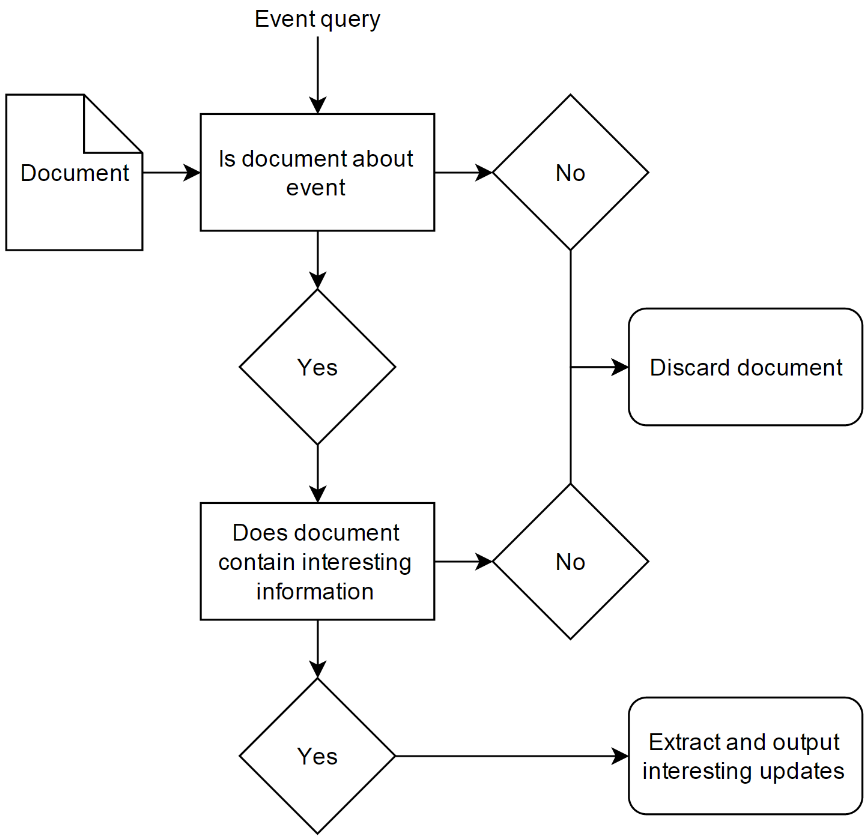
Event query
Document
Is document about event
No
Yes
Discard document
Does document contain interesting information
No
Yes
Extract and output interesting updates

Figure 2.1: Flowchart of a basic real time temporal summarization system

interesting updates by themselves.

Below is a run-through of the system where the system is set to produce updates about a tsunami that has occurred in Trondheim. The user submits a query to the system and it processes a stream of documents from one or more external sources. As it identifies relevant documents and processes them, it outputs updates that are deemed to contain interesting information.

---

**Query**
**query** = {trondheim tsunami}

**Document 1**
**title** = {Trondheim tsunami insurance}
**body** = {Buy your tsunami insurance now. Give our office in Trondheim a call at 93776583.}
**updates** = {}

---

**Document 2**
**title** = {Trondheim area hit by tsunami}
**body** = {The city centre of Trondheim was hit by a tsunami this morning. Eye witnesses reports that the wave was at least 4 meters high.}
**updates** = {{The city centre of Trondheim was hit by a tsunami earlier today}}

---

**Document 3**
**title** = {Millions of euros of damage caused by today's tsunami in Norway}
**body** = {The Norwegian authorities estimate that the tsunami has caused damage for 3 million Euro. This includes significant damage to the Train station of Trondheim as well as the harbor area that too the impact of the tsunami.}
**updates** = {{The city centre of Trondheim was hit by a tsunami earlier today}}

---

**Document 4**
**title** = {Trondheim in chaos after tsunami}
**body** = {The Norwegian prime minister promises that all available resources will be used to help the citicens after tsunami. The tsunami hit Trondheim earlier today and the damages so far has been estimated to more than 3 million euro.}
**updates** = {{The city centre of Trondheim was hit by a tsunami this morn-

ing}, {The tsunami hit Trondheim earlier today and the damages so far has been estimated to more than 3 million euro.}}

---

The title of the first document contains both “trondheim” and “tsunami”, but no sentences contain both of these words, so none of them are added to the updates.

The title of the second document contains both the words in the query, this is also the case for the first sentence, so we add this to the update feed. The second sentence would probably also be a good update but because of the simplistic way the system detects interesting sentences, it is discarded by the system.

The third document’s title contains the word “tsunami” but not the word “trondheim”, it is therefore discarded and no updates are produced. This is another case where the system ignores information that could have been interesting to a user. In this case, it happens because the system does not deem the document relevant, even though it clearly provides information about the event in question.

The fourth document is similar to document 3, but because of the different title, it is identified as relevant. Sentence 2 also contains both words in the query and it is extracted from the document. If document 3 looked a little different, the system could very well have selected sentences that are very similar from document 3 and 4. This is something that should be avoided because a user is not interested in receiving multiple updates with the same information.

This example illustrates many of the challenges that are present when constructing a real time temporal summarization system. Making a good system requires careful consideration of the different parts of the system to ensure that they work well together and that the end results is of good quality.

Some of the things that has to bee considered when creating such a system is, selecting the right documents, extracting the interesting information, submitting updates at the right time, avoiding redundant updates and making sure to cover the whole event without including irrelevant information.

# Chapter 3

# Related Technologies

Temporal summarization is closely related to a number of sub-fields within information retrieval and other topics. Some of these methods are highly relevant to temporal summarization and can be used for solving different sub-problems. This chapter will give a short introduction to some techniques that are related to temporal summarization.

## 3.1 Information Retrieval and the Vector Space Model

Information Retrieval is the problem of finding information that is relevant to some information need. Information retrieval is a large field with many different techniques that are used to search a set of documents for documents that are relevant to a query or improve the results of a search.

One of the most used techniques from information retrieval is the vector space model. The vector space model is a way of representing documents and calculating the similarity between them. In this model, documents are represented as vectors where each element if a feature of the document. Usually this means that each element in the document vector corresponds to a term in the document and the number represents the importance of the term in the document. The similarity between two documents are calculated by computing the angle between the two vectors and taking the cosine of the angle. For more in-depth description of different ways of implementing the vector space model and the TF-IDF weighting scheme, see [8].

Given two vectors $A$ and $B$, the similarity between the vectors are given by dividing the dot product of the two vectors by the length of the two vectors multiplied.

$$similarity(A, B) = \frac{A \bullet B}{||A|| \times ||B||} \tag{3.1}$$

To distinguish between important and unimportant terms in a document we use a measure called TF-IDF. TF-IDF stands for term frequency - inverse document frequency and is a common way to score terms in a document. The idea is that terms that occur many times in a document is important for this document and that rare terms are better suited to differentiate documents than common terms. When using TF-IDF weighting of terms, each document vector consists of the TF-IDF scores of each term in the document.

$$vector(d, D) = \{t \mapsto (tf(t, d) \times idf(t, D)) \ \forall \ t \in d\} \tag{3.2}$$

The term frequency $tf$ is defined as the frequency of a term $t$ in a document $d$. The frequency can be implemented as a simple boolean measure of term presence given a document or in some other way. A common definition of term frequency is dividing the number of occurrences of the term in a document by the number of occurrences of the most frequent term in the document. This is a way of preventing bias towards long documents and is referred to as double normalization. The most distinguishing term for a documents is the term in the document with the highest TF-IDF score

$$tf(t, d) = 0.5 + \frac{0.5 \times f(t, d)}{max\{f(w, d) : w \in d\}} \tag{3.3}$$

The Inverse document frequency is defined as the number of documents in the collection divided by the number of documents that contain a specific term. This ensures that rare terms are weighted higher than common terms. If no documents contain the term, the inverse document frequency is zero.

$$idf(t, D) = \begin{cases} 0 & \text{if } |\{d \in D : t \in d\}| = 0 \\ \log \frac{N}{|\{d \in D : t \in d\}|} & \text{otherwise} \end{cases} \tag{3.4}$$

To find the documents in a collection that is most similar to a query, we simply have to calculate the cosine similarity between the query and all documents, and rank them by this score.

## 3.2 Novelty and Redundancy Detection

Novelty or redundancy detection is the problem of detecting whether or not a piece of information is novel or redundant compared with some already known information.

Novelty and redundancy detection is commonly used by clustering systems to decide if a new sample is worth storing as a data point or if the data is already covered by other examples.

An example of novelty detection is refining the results of a documents search. If we have a set of results ranked by relevance, we might want to omit lower ranked results if they do not contain any new information compared to the highly ranked ones. This can make the overall results easier to process for the users but it might also exclude documents that the would be interesting to the user.

Most forms of novelty detection are based on comparing unseen documents to previously seen documents. If the new document is sufficiently different from the old documents they are assumed to contain new information. In many cases the comparison is done by comparing the terms in the documents with the terms in each previously seen document or the combined terms of selected documents. Comparing documents can be done pairwise or by means of some form of clustering, where documents are grouped together based on likeness to each other or some calculated or chosen exemplar for each cluster.

In a paper by Yi Zhang et al. [26]. The authors combine information filtering with novelty detection to eliminate redundant results from the filtered stream of documents. They use a variety of measures for detecting novelty and redundancy such as language models and cosine similarity. One of the conclusions is that simple similarity measures such as cosine similarity can perform very well for this kind of problems but that other methods might perform better if developed further.

## 3.3 Query Expansion and Modelling

Query expansion and query modelling are techniques used to modify a user's query with the goal of making it more suited to the information need of the user. Query expansion is some times referred to as relevance feedback. This comes from a way of performing query expansion where some results of a query is fed back into a query expansion module to expand the query. The feedback can be explicit or implicit. An example of explicit feedback is a user that selects a document that he or she believes to be relevant and the query is modified with terms from this document to produce more results like it. Implicit feedback can be automated without user feedback. A simple example of implicit feedback is to perform a query and select the top terms from the top results for this query and perform the search again. For more detailed discussion of query expansion and relevance feedback see [8].

An interesting approach to query modelling is presented in [18]. The work is based on the observation that documents are published in bursts as different events unfold. An example used by the authors is the query "grammys", the amount of documents published that are relevant to this query is highly concentrated around the time of the Grammy Awards each year. Based on the assumption that documents published during burst as are more likely to be relevant to a given query, we can use terms from the best documents during bursts to make a better query. The authors have reported significant improvements over various baselines.

The limitation of this approach is that it depends on our ability to reliably detect bursts in a stream. This is a simple task when we are free to examine the corpus retroactively, but detecting bursts in real time is naturally difficult. One of the reasons this is difficult for real time systems is the fact that bursts will naturally occur because of human sleeping patterns. Because of latency weighting of scores in the temporal summarization task, it is not acceptable to wait a day or even many hours to wait for bursts to become clear. It is possible to generate expansion terms as time progresses and more data becomes available, but there is a risk that these terms loose their relevance or introduce more noise.

## 3.4 Sentence Information Retrieval and Novelty Detection

There has been done some research on the topic of sentence IR and novelty detection on the sentence level. This problem of finding sentences that are relevant to some query is more challenging than finding documents because each sentence consists of much fewer terms and hence much less information.

In a paper [3] from 2003, the authors attempt to identify relevant and novel sentences from a ranked list of documents. This task is very similar to the temporal summarization task. The only difference is that the temporal summarization task requires temporal processing of the documents and places time constraints on selecting sentences.

The authors experimented with several different methods for sentence selection based on language models, TF-IDF and word counting. One of primary issues the authors wanted to address was the fact that most earlier methods were based on the assumption that all the evaluated documents was relevant.

The conclusion of the work is that working without the assumption of relevant documents makes sentence retrieval and novelty detection much harder. Many of the tested methods perform much worse when the assumption is lifted and this is especially true for the statistical and language models. The word counting methods also performed worse but the performance dropped more smoothly.

The problems covered by this paper is very similar to the temporal summarization task and it shows that great care must be taken when selecting documents to ensure that the results are good.

# Chapter 4

# TREC Temporal Summarization

To experiment with different techniques and to see how a real time temporal summarization system can be implemented, a defined task with set evaluation criteria that can be used to evaluate different techniques are needed. TREC has organized a track that focus on the problem of summarizing events in 2013 and 2014. This track provides the participants with a data set, rules, test events and evaluation criteria that are created with the goal of researching temporal summarization.

Participation ion the track is not a part of this thesis, but because the information is publicly available, It is possible to implement systems conforming to the requirements of the track and compare the results with teams that have participated officially.

This chapter is about organization of the TREC Temporal Summarization track and describes the premise, rules and requirements of the track. The content of this chapter is based on the information made available by the track organizers in [4].

## 4.1 TREC Temporal Summarization

The definition of a temporal summarization system used in this context is a system that processes a stream of documents to automatically produce updates related to a given set of events.

To make evaluation of the different systems easier, the track dictates that updates produced by a system can only be sentences that are present in documents found in the corpus. This means that all systems that are evaluated using the corpus and tools provided by the track are extractive summarizers.

This makes the task more simple and easier to evaluate for the organizers. It also restricts the participants from using techniques that generate their own updates. Sentences selected directly from documents have the advantage that they are sentences created by a human and that they most likely are complete and are easy to understand. Generating text is still a difficult problem and is not the focus of the track.

The 2013 run of the track featured an additional task compared to the 2014 run. This task was called value tracking and had the goal of tracking important numbers as an event unfolded, such as number of dead people or the financial impact of an event. Because most participants focused on the sequential update summarization task, this task was abandoned in 2014.

## 4.2 Temporal Summarization General Procedure

The track organizers have outlined a general high level procedure for a temporal summarization system. This outline contain no technical details and is of little use other than to illustrate how such a system might look like on a high level. The procedure is replicated in algorithm 1.

The key parts of this procedure is the part where documents are processed and the part where updates are extracted. There are no restrictions on how these parts can be implemented and it is possible to implement these parts of the system in many different ways.

**Algorithm 1** Temporal summarization general procedure

```
1: procedure SUMMARIZE
2:     $C \leftarrow$ time ordered corpus
3:     $e \leftarrow$ event
4:     $ts \leftarrow e.start$
5:     $te \leftarrow e.end$
6:     $R \leftarrow$ result
7:     for all documents d in C do
8:         $time \leftarrow d.time$
9:         process(d)
10:        if $ts \leq time$ and $time \leq te$ then
11:            $Ut \leftarrow$ extractUpdates()
12:            for all updates u in Ut do
13:                R.append(u)
14:    return R
```

Previous track runs have mostly used text indexing techniques to store information and separate relevant documents from irrelevant documents. For determining which sentences to use as event updates many different techniques have been used. Some examples include bag-of-words novelty detection approaches and cluster based query expansion approaches.

To make it easy to experiment with multiple approaches, the system should be separated into modules that handle sub-problems. Especially the modules responsible for choosing updates needs to be clearly defined and independent of other parts of the system so that it is easy to test widely different approaches.

## 4.3 Corpus

The track use a filtered version of the 2014 KBA Stream Corpus [1] from the TREC Knowledge Base Acceleration track. To make the size of the

[1] KBA Stream Corpus 2014, http://trec-kba.org/kba-stream-corpus-2014.shtml

corpus more manageable for the participants, the organizers provide a filtered version [2].

The corpus is serialized with Apache Thrift, encrypted with GnuPG and compressed with XZ. Documents span in publication time from December 2011 to April 2013 and are divided into folders by hour. Each hour worth of data is divided into multiple chunks where each chunk contain a certain type of documents. Document types in this context means different types of sources, for instance news documents, forum posts or posts on social media. The creators of the corpus format supply a set of tools that can be used to extract the documents from the corpus in various formats.

The unfiltered version of the corpus consists of just above 16TB worth of documents. To make the amount of data more manageable for the participants, the track organizers supply a filtered version. The filtered version is around 590GB in size, this is achieved by removing documents that are not within the time frame of at least one of the evaluation events. Documents that does not contain at least one word from the evaluation event queries are also removed, as they are very unlikely to be relevant to any of the test events.

Each document in the corpus is timestamped and should be processed in chronological order. As the name suggests, the stream corpus emulates a stream of newly published documents and should be treated as such. As can be seen in the temporal summarization general procedure 1 reading a document requires you to set your system time to a time no less than the time stamp from the read document. It is not allowed to "peek ahead" by reading documents without adjusting system time. This also means that you are not allowed to use TF-IDF scores or similar based on data that is published after the document.

## 4.4 Evaluation Events

The documents also contain a title and a link to a Wikipedia article describing the event but these are retrospective data not to be used during simulations.

[2]TREC Temporal Summarization 2014 corpus, http://s3.amazonaws.com/aws-publicdatasets/trec/ts/index.html

The timestamps are GMT UNIX-timestamps and are defined as the number of seconds passed since 00:00 1.January 1970.

Example events are provided by the track organizers and consists of:

- **Id:** The identifier for the event
- **Start:** A time stamp that defines when the event started
- **End:** A time stamp that defines when the event ended
- **Query:** A query representing a users description of an event
- **Type:** The type of event, is one of accident, bombing, earthquake, hostage, impact event, protest, riot, shooting, storm

The query of an event is what people would likely use when referring to an event in the present, not how they would refer to an event after it has passed. The query for the event “Early 2012 European cold wave” is for instance simply “European cold wave”.

## 4.5 Results format

The results of a run is a set of updates for each event. Each update consists of:

- **Query Id:** The id of the event that the update belongs to
- **Run Id:** The name of the solution that generated the update, this allows for a system run multiple solutions in a single batch.
- **Document Id:** The id of the document that contains the sentence that was selected as an update
- **Sentence Id:** The index of the sentence in the document
- **Decision timestamp:** The system time at the time the update was generated

In addition to these fields each update contains the name of the team that created the solution and a confidence value that was used by the track organizers when evaluating submissions. None of these are relevant for this thesis.

The results of a run is evaluated using a script that calculates the scores of the systems using different evaluation metrics.

## 4.6 Evaluation and Metrics

To accurately compare different systems using different approaches, the organizers of the temporal summarization track have defined a set of common evaluation metrics that measure different aspects of a temporal summarization system creating according to the specifications of the task. The metrics are based on common information retrieval metrics, but has been tailored to the real time temporal summarization domain by combining these with latency and verbosity of updates. The complete definition of these metrics can be found in [5].

Scoring different systems is at the basic level done by comparing updates produced by the systems to a set of gold standard updates chosen by the track organizers for each event. These gold standard updates are called nuggets and are extracted from Wikipedia edit history of pages concerning the events for the task. Submitted systems are evaluated based on how well their updates cover the information provided by the nuggets. The evaluation criteria are based on traditional information retrieval metrics but also take into account that a temporal summarization system is not a standard information retrieval system. In addition to equivalents of precision and recall, the systems are evaluated by how early the produced updates based on the Wikipedia edit and how verbose or concise they are.

Below is a list of the main evaluation criteria for the track.

- **Expected Gain** - How much information is provided by the updates
- **Comprehensiveness** - How well the updates cover the event
- **Latency** - How soon the updates are delivered
- **Verbosity** - How compact the updates are

- **Combined Metric** - A combination of the different metrics to produce a single score

## 4.6.1 General Definitions and Functions

The output of a system is a set of updates where every update is a sentence from one of the documents in the corpus. The set of updates generated by a system id referred to as $S$.

$$S = \textit{the updates of a system} \tag{4.1}$$

Metrics are calculated separately for each event that is summarized by the system. The subset of updates produced for a particular event is referred to as $S_e$.

$$S_e = \textit{the updates of a system for a particular event} \tag{4.2}$$

The set of nuggets that are relevant to an event is referred to as $N_e$.

$$N_e = \textit{the defined nuggets for a particular event} \tag{4.3}$$

Updates are compared with nuggets to calculate the various metrics used to evaluate a system. The earliest matching update of a nugget is defined as the earliest update generated by a system that matches the content of a given nugget.

$$M(n, S_e) = min\{u \in S : n \approx u\}u.time \tag{4.4}$$

The set of nuggets that have an update $u$ as it's earliest matching update is referred to as $M'$.

$$M'(u, S_e) = \{n \in N_e : M(n, S_e) = u\} \tag{4.5}$$

### 4.6.2 Expected Gain

The expected gain of a temporal summarization system is the expected information gain for an update and is defined as the sum of every update gain divided by the number of updates produced by the system. This metric is similar to precision in information retrieval system in that it says something about how valuable each result is for the user. A high expected gain tells us that each update provides relevant information and that the amount of noise and irrelevant results are low.

The gain of an individual update is calculated based on how many nuggets are covered by the update and how relevant to the event these nuggets are.

$$gain(u, S_e) = \sum_{n \in M'(u,S_e)} relevance(n) \tag{4.6}$$

The expected gain of a system is the sum of the gain generated by each update divided by the total number of updates.

$$EG(S_e) = \frac{1}{|S_e|} \sum_{u \in S_e} gain(u, S_e) \tag{4.7}$$

### 4.6.3 Comprehensiveness

The comprehensiveness of a temporal summarization system is how well the updates of the system cover the defined nuggets for an event. The comprehensiveness is a measure of how many of the nuggets that are covered by every update on average. A system is rewarded for covering as many nuggets as possible and punished for leaving out information.

$$C(S_e) = \frac{1}{\sum_{n \in N_e} R(n)} \sum_{u \in S_e} gain(u, S_e) \tag{4.8}$$

### 4.6.4 Latency Discount

To penalize systems that produce late updates and reward system that produce early updates, the gain of a specific update for a given set of matching nuggets is discounted by a latency factor. The latency factor is given by a function that compares the time stamp of the update with the time stamp of the nugget. Depending on how many hours before of after the nugget the update was generated, the factor increases or decreases. The function is an inverse tangent where the function varies from 2 to 0, with updates that are generated at exactly the same time as the nugget will have a latency factor of 1. At positive or negative 10 hours, difference the factor will be approximately 0.3 or 1.7.

To incorporate the latency factor into the overall evaluation of the system, the latency factor is applied to the gain of each update. The latency discounted gain function looks like this:

$$gain(u, S_e) = \sum_{n \in M'(u,S_e)} relevance(n) \times latency(u, n) \tag{4.9}$$

### 4.6.5 Verbosity Normalization

Verbosity normalization is applied to the expected gain function to punish systems that are too verbose and reward systems that manage to convey meaningful information in a compact manner. This normalization is applied to the overall verbosity of the system and not each individual update. This is to ensure that systems are not punished for longer updates that cover many nuggets. The normalization works by taking the number of words in the updates that do not match nuggets and divide it by the average number of words in a nugget for the given event.

The verbosity normalized version of the expected gain divides the sum of the combined gain from each update by the combined verbosity of the updates instead of the number of updates.

$$EGV(S_e) = \frac{1}{\sum_{u \in S_e} verbosity(u)} \sum_{u \in S_e} gain(u, S_e) \tag{4.10}$$

The verbosity of an update is defined as 1 plus the number of words in an update that does not match a nugget, divided by the average nugget length for the particular event. An update that does not contain any words not found in a nugget will have a verbosity score of 1, the score will increase with the number of unrelated words.

$$V(u, e) = 1 + \frac{|u| - |words\ from\ u\ matching\ nugget|}{avg_{n \in N_e |n|}} \tag{4.11}$$

### 4.6.6 Combined Metric

The overall metric for the system is the harmonic mean between the expected gain and the comprehensiveness of the system. This is the equivalent of an F-measure in information retrieval where expected gain corresponds to precision and comprehensiveness corresponds to recall. The function is a combination of comprehensiveness, expected gain, latency and verbosity. This ensures that different systems are scored according to multiple criteria of a good temporal summarization system.

$$F(S) = \frac{EGV(S) \times C(S)}{EGV(S) + C(S)} \tag{4.12}$$

# Chapter 5

# TREC Temporal Summarization 2013 and 2014

The TREC temporal summarization track was ran in 2013 and 2014. Both of these runs featured roughly the same task description, evaluation criteria, test events and data set. This chapter presents the results of the participants of the 2013 and 2014 track.

## 5.1 2013 Results

2013 was the first year of the temporal summarization track. The teams chose widely different approaches and even after the track, there are no definitive answers as to what is the best approach for this problem.

The 2013 track did not use a combined metric to evaluate the results but rather looked at gain and comprehensiveness separately. In order to properly compare the results with the results from 2014 I calculated this score myself using the formula used for the 2014 track and the supplied gain and comprehensiveness scores. As mentioned in the track overview, the formula is a standard harmonic mean between gain and comprehensiveness.

Below is a list of the different participants and their most important technologies as well as the results of each submitted run. The Highest achieved score for each metric category is **bolded**.

- **ICTNET** - title filtering, Wikipedia based trigger words, simhash redundancy detection
- **HLTCOE** - bag of words event models based on unigrams; named entities and verbs with Wikipedia expansion, feature based sentence selection based on document and sentence relevance; novelty; salience and number presence
- **PRIS** - query expansion, document and sentence relevance, latent Dirichlet allocation based keyword mining
- **UOG** - search engine, sentence ranking, Gibbs sampling
- **ZZISTI** - sentence importance by entity identification, cosine similarity based novelty detection
- **UWaterlooMDS** - ranking hourly chunks of documents using query likelihood with Dirichlet Smoothing, various sentence ranking schemes

## 5.2 2013 General Discussion

In general there was a big difference in how the different runs performed by different metrics. Some runs performed great in the comprehensiveness metric but these where often the poorest performers when it came to the update gain metric. The best runs where the runs with a good update gain, as their comprehensiveness was often just half of the comprehensiveness score of the most comprehensive runs, while the expected gain of the runs with good comprehensiveness was often many times worse. There was also a large difference in overall score achieved by the different teams,

All of the submitted systems submitted used an approach where documents first were judged for relevance and sentences were selected only after determining if the document was relevant or not.

The way the participants determined if a document was relevant or not can be separated into two categories. The first approach was to filter documents

| **Run Id** | **Gain** | **Comp.** | **F-Measure** |
|---|---|---|---|
| ICTNET-run2 | 0.127 | 0.251 | **0.1686** |
| ICTNET-run1 | 0.125 | 0.253 | 0.1674 |
| hltcoe-TuneBasePred2* | 0.114 | 0.244 | 0.1554 |
| hltcoe-TuneExternal2* | 0.117 | 0.203 | 0.1484 |
| PRIS-cluster5 | **0.136** | 0.126 | 0.1308 |
| PRIS-cluster3 | 0.103 | 0.176 | 0.1300 |
| PRIS-cluster2 | 0.074 | 0.260 | 0.1152 |
| hltcoe-BasePred | 0.067 | 0.368 | 0.1134 |
| PRIS-cluster1 | 0.067 | 0.292 | 0.1090 |
| PRIS-cluster4 | 0.067 | 0.288 | 0.1088 |
| hltcoe-Baseline | 0.063 | 0.381 | 0.1082 |
| uogTr-uogTrNMTm1MM3 | 0.069 | 0.216 | 0.1046 |
| uogTr-uogTrNSQ1 | 0.060 | 0.184 | 0.0904 |
| hltcoe-EXTERNAL | 0.054 | 0.413 | 0.0956 |
| uogTr-uogTrNMM | 0.045 | 0.254 | 0.0764 |
| uogTr-uogTrNMTm3FMM4 | 0.049 | 0.170 | 0.0760 |
| uogTr-uogTrEMMQ2 | 0.040 | 0.259 | 0.0692 |
| ZZISTI-wim-GY-2013-SUS1 | 0.036 | 0.148 | 0.0580 |
| UWaterlooMDS-rg4 | 0.028 | 0.516 | 0.0532 |
| UWaterlooMDS-rg3 | 0.026 | 0.506 | 0.0494 |
| UWaterlooMDS-rg2 | 0.022 | 0.562 | 0.0424 |
| UWaterlooMDS-rg1 | 0.021 | **0.571** | 0.0406 |
| UWaterlooMDS-UWMDSqlec4t50 | 0.018 | 0.530 | 0.0348 |
| UWaterlooMDS-UWMDSqlec2t25 | 0.017 | 0.537 | 0.033 |
| UWaterlooMDS-CosineEgrep | 0.010 | 0.018 | 0.0128 |
| UWaterlooMDS-NormEgrep | 0.001 | 0.061 | 0.0020 |

Figure 5.1: Gain, Comprehensiveness and F-Measure scores of the 2013 runs [7]

based on some criteria. Examples of this include the solution created by the ICTNET team[13] that filtered documents based on the presence of the query terms in the title of the document, and the solution by the ZZISTI team[22] that considered a document relevant if it contains the query terms. The other solutions[24][25][23][15] all used some form of similarity based comparison to decide if a document is related to a particular event. Some participants used

TF-IDF scoring and cosine similarity while others relied on various search engines or clustering techniques to achieve the same goal.

The three best performing teams all used some form of external corpus such as a Wikipedia dump to extend their various forms of event models. What made these teams stand out was their very high gain score compared to the other participants, indicating that their way of selecting sentences is a good one. Many participants also used some form of query expansion, either for expanding event models or for the initial identification of relevant documents.

HLTCOE used a solution where sentences where selected based on various features, not only the presence of certain terms. Most of these features were based on their generated bag-of-words representations of each event, but they also included features such as the presence of numbers and the novelty of a sentence compared with the current bag-of-words representation of the event.

The WARTERLOOMDS team submitted several different runs and all of them got poor latency gain score but the team had the best scores by the comprehensiveness metric. The runs with the highest comprehensiveness score expanded the original query by the top expansion terms each hour to track the evolution of the event. Relevant sentences were also selected based on the expansion terms. These runs where also among the runs that generated the highest number of updates. The low gain score combined with the high comprehensiveness could indicate that while the event expansion method is good at finding more relevant documents, it also causes the system to produce more irrelevant updates or updates with no new information. This could also mean that redundancy detection is not enough to ensure that produced updates provide relevant information, at least if the number of believed relevant documents is high. The ICTNET team relied on simple similarity based redundancy detection to ensure novelty, but in their case, the pool of possible sentences were much lower because of their restrictive document relevance judgement.

## 5.3 2013 Highlight: ICTNET

The team with the highest combined score in the 2013 track was the ICTNET team. This team employed a real time multi stage filtering approach where documents and sentences are gradually filtered and the sentences remaining after the filtering are selected as updates.

The filtering process can be summarized as follows:

- Filter out irrelevant documents
- Select sentences that contain trigger words learned from Wikipedia
- Remove redundant sentences

This approach differs from a couple of other approaches in that their approach is entirely real time and does not introduce delays by collecting documents or sentences at any stage throughout the process. As long as the system is able to identify good updates, this could yield considerable bonus points for generating updates in a timely fashion.

The first stage of their filtering process is to discard documents that are believed to be irrelevant. This is done by discarding documents where the title does not contain all terms in the query. Using the title of the document proved to yield better results than using the body of the document. The second stage of the filtering is selecting sentences that contains certain trigger words. These terms include words such as “killed”, “victims” etc.

It is unclear from the report weather or not these trigger words are event specific, event type specific or generic.

Additional words were learned by using an old snapshot of Wikipedia to identify terms frequently used in updates concerning the different types of events. The last step in the filtering process is to remove redundant updates. This is done by computing the similarity between selected sentences and previous updates using a simhash algorithm and discarding redundant sentences while the remaining sentences are added to the results.

Compared to many of the other participants, the methods used were very simple. The only part of the system that involves any actual analysis of data is the selection of trigger words. The fact that the system is very restrictive in choosing relevant documents and sentences may be one of the reasons it performed relatively well compared to other systems. Choosing documents based on their titles is very restrictive, but if this diminishes the likelihood of choosing irrelevant documents it might be worth it. Choosing sentences based on trigger words might discard relevant sentences, but is also less likely to choose irrelevant sentences. The fact that many other runs loose out to this simple but conservative system could indicate that these other systems have problems with noise. This will have to be tested to be verified, but it is important to consider the effects of irrelevant documents when selecting sentences as updates.

## 5.4 2014 Results

The 2014 track is mostly the same as the 2013 version. Guidelines, metrics and corpus are all the same, the only thing that was changed from 2013 was the events used for evaluation. Some participants from 2013 make their return, but there are also new participants. Some of the participants significantly altered their approach, but some techniques return in some way or form.

Below is a list of the main techniques used by the different participants for the 2014 run and the results achieved by each submitted solution[12][9][10][16][27][19].

- **CUNLP** - affinity propagation, Gaussian processes, semantic similarity
- **BJUT** - query expansion, classification, clustering
- **UOG** - filtering of various sentence features
- **IRT** - machine learning, generic event model
- **ICTNET** - latent Diriclet allocation, support vector machine learner
- **BUPT** - unclear, query expansion and improvement

| **Run Id** | **Gain** | **Comp.** | **Lat** | **F-Measure** |
|---|---|---|---|---|
| CUNLP-2APSal | 0.0631 | 0.322 | 0.2304 | 0.1162 |
| BJUT-Q1 | 0.0657 | 0.4088 | 0.3238 | 0.1110 |
| BJUT-Q0 | 0.0632 | 0.3979 | 0.2421 | 0.1091 |
| BJUT-Q2 | 0.0632 | 0.3979 | 0.2421 | 0.1091 |
| uogTr-uogTr2A | 0.0467 | 0.4453 | 0.1424 | 0.0986 |
| uogTr-uogTr4AC | 0.0347 | 0.4539 | 0.1076 | 0.0793 |
| uogTr-uogTr4ARas | 0.0387 | 0.3691 | 0.1300 | 0.0772 |
| IRIT-KW30H5NW3600 | 0.0383 | 0.3521 | 0.0621 | 0.0723 |
| IRIT-KW30H5NW300 | 0.0378 | 0.3538 | 0.0614 | 0.0714 |
| uogTr-uogTr4A | 0.0281 | 0.4733 | 0.0752 | 0.0677 |
| average | 0.0327 | 0.3615 | 0.1031 | 0.0620 |
| IRIT-KW80H5NW3600 | 0.0289 | 0.3764 | 0.0440 | 0.0604 |
| CUNLP-1APSalRed | 0.0325 | 0.3058 | 0.0601 | 0.0602 |
| IRIT-KW30H10NW300 | 0.0298 | 0.378 | 0.0504 | 0.0602 |
| IRIT-KW80H5NW300 | 0.0285 | 0.3806 | 0.0435 | 0.0596 |
| ICTNET-run3 | 0.0531 | 0.1081 | 0.2794 | 0.0530 |
| BUPT-PRIS-Cluster4 | 0.0155 | 0.2692 | 0.0314 | 0.0508 |
| IRIT-KW80H10NW300 | 0.0225 | 0.4012 | 0.0361 | 0.0503 |
| BUPT-PRIS-Cluster3 | 0.0115 | 0.338 | 0.0208 | 0.0407 |
| CUNLP-3AP | 0.0174 | 0.4265 | 0.0318 | 0.0403 |
| ICTNET-run2 | 0.0418 | 0.0934 | 0.2269 | 0.0311 |
| BUPT-PRIS-Cluster2 | 0.0059 | 0.3728 | 0.0101 | 0.0222 |
| ICTNET-run4 | 0.0079 | 0.407 | 0.0096 | 0.0178 |
| ICTNET-run1 | 0.007 | 0.409 | 0.0084 | 0.0160 |
| BUPT-PRIS-Cluster1 | 0.0033 | 0.4369 | 0.0059 | 0.0127 |

Figure 5.2: Gain, Comprehensiveness, latency and F-Measure score of the 2014 runs [6]

## 5.5 2014 General Discussion

In general there was no improvement in the scores compared to the 2013 track, although there was some interesting new approaches. The trend from 2013 where runs with high gain score much better than runs with high comprehensiveness, this is also noted by the organizers in the overview paper[6] and they conclude that further research should be focused on improving the precision of temporal summarization systems.

## 5.6 2014 Hightlight: CUNLP

One of the best performers from the 2014 run was the team from Columbia University. Their solution used methods that was not used in 2013 and stand out in comparison to the other participants.

Their solution is primarily based on affinity propagation and Gaussian processes. Affinity propagation is a clustering technique based on message passing. Clusters are defined by exemplars that are actual data points, and data points are assigned clusters based on. Affinity propagation is initialized with a matrix of similarity between each data point and a preference vector where each data point have a preference value that express an a priori interest in using each point as an exemplar.

The similarity between the data points are a form of semantic similarity and the preference values is a numeric representation of how likely this sentence is to be selected as an update. This likelihood is estimated by training a Gaussian process model to recognize the structure of a gold nugget update from the 2013. The features used to model each nugget are non-semantic to allow for recognition of good updates regardless of content and what event it would be relevant to.

The team divide the corpus into hourly chunks and evaluates one hour at a time, this means that each update will will have a delay of one hour which will affect the score but this is also necessary with the methods chosen by the team. Documents are filtered so that only documents containing at least on query word is selected. Each hour worth of documents are divided into sentences and each sentence gets a preference value and their similarity computed. After the preferences and similarity matrix is computed, affinity propagation is run to identify clusters. The exemplars of each cluster is chosen as an update.

The team experienced some problems with their system selecting sentences that contained category text such as a news site navigation bars as cluster exemplars. The reason this happened was that navigation often use words that describe categories and that added together they would be similar to many different topics. To compensate for this they trained their system to detect such sentences and discard them. This problem seems to be a consequence of their choice of clustering algorithm, other participants have not reported similar problems.

The team submitted 3 different runs, where one of them received a noticeably better score than the other two. The run with the best score was also the one with the least amount of updates. This was also observed in last year's run, likely reasons for this are discussed in the previous section.

The team achieved a good score in this years run in comparison to many of the other participants. Despite the good score it is not better than the top performing teams from last years run. It seems that in general, the scores were about the same as last year, however the top performers from last year still performed better than the best performers this year. It is not clear whether this has to do with the evaluation events or if the top performer from last year was simply better.

## 5.7 Summary

Most submissions employ some form of query expansion or others means of identifying terms that are important to an event. some use generic event models generated based on a corpus, while others continually update several models of an event. Either way, query expansion is an important part of identifying sentences that are suitable as updates. Some participants use query expansion for identifying relevant documents, but this seems less necessary. In some cases this leads to poor performance because too many documents are selected.

The reason some form of query expansion is needed is that each event comes with very little data. Selecting sentences based on the presence of query terms alone can cause a system to miss many good updates. This can be intuitively explained by the following example: Suppose you want to receive updates about the 2008 olympic games, your query would probably look something like "2008 olympic games" or "Beijing olympic games". With a functional information retrieval system we would find documents that are about the 2008 Beijing olympic games. The problem is that sentences that would be considered good updates often are nothing like the original query. An example of a good update could be “Shelly-Ann Fraser-Pryce takes gold in 100m sprint” or “china finishes as the nation with the most gold medals at 51 gold medals”, but none of these sentences contains any terms from the original query. To produce updates like these, we have to use different techniques than simple query similarity.

An approach used by HLTCOE in 2013 involved using many different features that are not only based on the presence of trigger words generated by query expansion or similar. This technique was not very common among the submissions, but the performance of the HLTCOE submission make this an interesting approach. Simply increasing the likelihood of selecting a sentence that contains numbers will probably not increase the performance of a system by a large margin, but if good features can be identified it might be a supplement to the keyword approach. A potential difficulty with this is that different features might not be well suited to all kinds of events.

All of the successful submissions also included some form of novelty or redundancy detection when selecting sentences. Because the corpus contains republications of many articles, having no form of redundancy or novelty detection will severely hurt the performance of a system.

# Chapter 6

# Temporal Summarization Test Framework

This chapter will discuss the overall design of the TREC temporal summarization system and it's general components.

## 6.1 System Overview

In it's most simple form, the temporal summarization system reads from a stream of documents and produces updates related to one or more event that is has been set to monitor.

Below, each major part of the system is Described briefly to give an overview of the design and responsibilities of each component. The different parts are discussed in more detail in their own sections.

### 6.1.1 Preprocessor

The temporal summarization corpus supplied by the track organizers comes serialized in the Apache Thrift format and is semantically tagged by the Serif tagger. This makes the corpus very large and it contains a lot of data that is not needed by a typical solution to the task. To reduce that size of the corpus and make the format easier to work with, the corpus will have to be preprocessed in a way that is tailored to the implementation of the temporal summarization system.

### 6.1.2 Generic Text Corpus and TF-IDF Index

The rules of the track states that a solution may not use data that became available after a given document was published to produce updates from said document. This means that any statistical data, TF-IDF scores or similar must be compiled before the first document in the corpus or generated on the fly by the system. Since the temporal summarization corpus comes prefiltered by the track organizers, it is not suitable as a generic corpus. Some participants have used Wikipedia dumps for this purpose with good results. A generic corpus can be used to calculate reliable TF-IDF values that can be used for various purposes by a text index.

The index must be able to compare text vectors, discover related terms, or identifying important terms in a document, depending on what is needed by the different solutions. The index can be thought of as a semantic backbone that allows the solutions to make informed decisions when evaluating text.

### 6.1.3 Corpus Reader

Since the corpus is too large to fit in main memory, it has to be read from disk before it van be processed. The function of the corpus reader is to read chunks of the corpus, parse it into individual documents and feed them to the rest of the system in a format that is suitable for processing.

### 6.1.4 Document Processors

The document processors are the main parts of the system, they analyze documents to produce updates. The document processors should use a common interface to ensure that they are interchangeable and easy to implement and test. They must also be allowed and able to store any information they need to process future documents.

## 6.2 Preprocessing

The data set provided by the track organizers comes serialized in the Apache Thrift format. Because this format require a specific tool set to work with and because the format is a little cumbersome, most participant chose to extract the desired data and store it in a more compact and easily manageable format. Many participants simply extracted the raw content and stored the documents as .txt files. The format also stores the content in multiple forms such as raw HTML and cleaned versions, making the corpus many times larger than it needs to be.

Only English news articles are considered when generating updates. The main reason for this is that the nuggets and queries are all English, allowing other languages is a potential source of noise, but is unlikely to contain any interesting information. It is also likely that a user monitoring an update stream would like the stream to be in his or her language, making it a reasonable decision to only look at documents in a specific language. For the sake of keeping the collection of documents small, non-English documents are removed by the preprocessor.

The preprocessor reads the documents from the corpus and extracts the desired data before it stores them as JSON-files. Only the Id, timestamp and full text of the original documents are kept. Additionally, the title of each document is extracted by scanning the HTML version of the document for the title tag.

Each JSON file contains one hour worth of documents and is compressed before written to disk. Each compressed chunk is about 3-10 MB in size.

The preprocessor(2) is written in python and is not part of the temporal summarization system, it is run separately and produces files that are used by the actual system.

**Algorithm 2** Preprocessor pseudo code

```
1: procedure PREPROCESS
2:     $Hours \leftarrow ordered\ list\ of\ hours$
3:     for all hour h in Hours do
4:         $Chunks \leftarrow list\ of\ chunks\ in\ hour$
5:         $HourChunk \leftarrow output$
6:         for all chunk c in $Chunks$ do
7:             for all document d in chunk do
8:                 if $d.language = english$ and $d.source = news$ then
9:                     $d' \leftarrow empty\ document$
10:                    $d'.id \leftarrow d.id$
11:                    $d'.timestamp \leftarrow d.timestamp$
12:                    $d'.title \leftarrow extractTitle(d)$
13:                    $d'.body \leftarrow d.cleanBody$
14:                    $HourChunk.append(d')$
15:            $wiriteToDisk(HourChunk)$
```

## 6.3 System Structure

The general system structure is based on the temporal summarization general procedure that is provided by the track organizers and reproduced in chapter 4. The figure 3 below illustrate how the system operates and what components are used throughout the process.

When the process is started, it starts by reading the events to be evaluated as well as initializing the text index and indexing the text corpus. When this is complete, summarizers are initialized based on the process parameters and the events that will be summarized.

When the setup is done, the system starts reading from the stream corpus with the document stream reader. Each document is processed by the active summarizers until the entire corpus is processed. Because of the general summarizer interface the method for generating updates can be changed without changing other parts of the system. When this is done, the updates from each summarizer are written to disk for evaluation.

**Algorithm 3** General Framework pseudo code

```
procedure SUMARIZE(sumarizerType, tuning)
    $Corpus \leftarrow readCorpus()$
    $Index \leftarrow createIndex()$
    $Events \leftarrow readEvents()$
    $DocumentStream \leftarrow initializeDocumentStream()$
    $Sumarizers \leftarrow empty\ set$
    $updates \leftarrow empty\ list$
    for all document d in Corpus do
        $Index.index(d)$
    for all event e in Events do
        $Sumarizers.put(createSumarizer(e, summarizerType, tuning))$
    for all document d in DocumentStream do
        for all sumarizer s in Sumarizers do
            $s.process(d)$
    for all summarizer s in Summarizers do
        $Updates.add(s.getUpdates())$
    $writeUpdates(Updates)$
```

## 6.4 Document Stream

The document stream is responsible for reading the stream corpus from disk and parsing documents. The corpus is much larger than the main memory of most machines and have to be read gradually. When the corpus has been processed by the preprocessor, it is stored in hourly chunks. Most chunks are between 8 and 30 MB and can be read fairly quickly. Each chunk is a compressed JSON file, it is read using a stream reader that can read compressed formats and parsed from JSON after it has finished reading. The stream reader itself behaves as an iterator. When an element is assessed, the reader reads a chunk from disk and fills a buffer with documents. When the chunk has been exhausted, a new chunk from disk to refill the buffer.

## 6.5 Text Index and News Corpus

The system contain a text index that is used for generating TF-IDF scores and comparing text vectors. The text index is used by different solutions for various purposes.

It would have been possible to use an open source search engine for this purpose, but while many open source search engines such a Lucene are very good and easy to use, they often hide the more primitive features of a search engine such as the TF-IDF vectors. This is a problem because if I wish to do more than just finding relevant documents, this may require modifications of the framework which could be very difficult. Because of this it was decided that a simple in-memory text index should be implemented for use by the temporal summarization system.

To generate TF-IDF scores and compare vectors, a corpus for generating term statistics are needed. Because the temporal summarization track focus on news stories, the New York Times corpus was chosen for this purpose. The corpus consists of over 1.8 million English news documents published between 1987 and 2007. A downside to this corpus is that all documents are written in American English, while the documents in the temporal summarization corpus consists of documents from all over the world. The stories are however well covered by media all over the world and even if the system inherit a slight bias for American English documents it should be OK. The corpus also have the advantage of being readily available and comes with a set of tools to read and extract the documents that is easy to use.

A very important use of the index is finding terms related to a query or another document. This can be used as a form of query expansion and is very useful. The algorithm(4) first searches the index to find the top N most relevant documents for the query. These documents are assumed to be relevant and contain terms that are related to the query. To find the terms that are the most related to the query, each therm in each document is added to a map where the value of a term is the sum of it's TF-IDF value from each document. Another way of doing this is to simply count how many documents a term occurs in or count the number of occurrences throughout the documents, but this could lead to frequent unimportant term such as stop-words to be selected. By summing the TF-IDF values, we ensure that both frequency and importance is considered when selecting top terms.

**Algorithm 4** Query Expansion pseudo code

```
1: procedure ExpandQuery(query, numTopDocuments, numTopTerms)
2:     $TopDocuments \leftarrow empty\ sorted\ list$
3:     $Terms \leftarrow empty\ map$
4:     $TopTerms \leftarrow empty\ list$
5:     for all documentVector d in index do
6:         $score \leftarrow similarity(query, d)$
7:         if $TopDocuments.length < numTopDocuments$ then
8:             $TopDocuments.add(\{d, score\})$
9:         else if $score > TopDocuments.lowest().score$ then
10:            $TopDocuments.removeLowest()$
11:            $TopDocuments.add(\{d, score\})$
12:    for all documentVector d in TopDocuments do
13:        for all (term, tfidfScore) p in d do
14:            if $Terms.containsKey(p.term)$ then
15:                $Terms.add(p.term, p.tfidfScore)$
16:            else
17:                $Terms.incrementKey(p.term, p.tfidfScore)$
18:    for all (term, tfidfSum) p in Terms do
19:        $TopTerms.add((p.term, p.tfidfSum))$
20:    $sortReverse(TopTerms)$
        return $TopTerms.subList(0, numTopTerms)$
```

## 6.6 Summarizer Interface

The solution interface defines how a specific implementation of a solution to the temporal summarization problem looks like to an external module. The interface is simple and states that a solution implementation must be able to process a document and generate results. The inner workings of a solution is not defined and will look different based on what solution is being employed. Solutions are also tunable with a set of parameters that are specific to each solution.

The next chapter will discuss the implementation of each summarizer and evaluate the results achieved with different approaches.

## 6.7 Evaluation

Evaluation of the results are done by running a script provided by the track organizers. The results are written to disk using a specific format and each summarizer have a unique name that separates it's updates from other summarizers. The result format was discussed in chapter 4.

When the evaluation script is run it produces a list of results for all the summarizers in the results file as well as detail statistics on individual events and different versions of the measures used to evaluate a solution.

# Chapter 7

# TREC Temporal Summarization Solutions

This chapter covers the design and implementation of the different solutions that was implemented for the TREC temporal summarization track task. The layout for these solution is outlined in the previous chapter. This chapter deals with the inner workings of these solutions and how they perform their task of generating updates.

## 7.1 Baseline Design

The first solution that was implemented is a simple solution based roughly on the solution implemented by the ICTNET team for the 2013 track[13]. This was the best solution of the year, but it also relied on simple techniques that can be modified in many ways.

The solution works by sending the data through multiple filters and returning sentences that pass every stage of the filtering process. The first stage in this process is to discard documents that are believed to be irrelevant. A document is determined to be irrelevant if the title does not contain every work of the query. When a document is determined to be relevant, sentences are extracted based on the presence of trigger words. If a sentence contain one or more trigger words, the sentence is considered a potential update.

Sentences are added to the update stream if they are adequately different from all earlier updates. A sentence is considered adequately different if it is similar but contains new numbers or if it is sufficiently different. The similarity between sentences is computed using the Jaccard coeficcient.

The collection of trigger words can be referred to as an event model, a collection of terms that are believed to be significant when identifying sentences that are suited as updates. To have a basis for comparing future solutions and to decide if the overall structure of the solution was suited to the problem, this first version was supplied with a manually created event model for update identification. A manually generated event model so not very interesting by itself, but can be used as a comparison for other event models and Will still let other parts of the system be tested as normal.

Trigger words were chosen specific for each event type based on intuition and expected subclasses of the event type. The trigger words were not tailored to the test events but rather chosen to cover as many event sub types as possible. The trigger words for the storm category were chosen to fit many kinds of weather events, not just storms and cold waves, which was the types of events included in the test events.

### 7.1.1 Parameters

Below is an overview of the parameters used by this solution.

- *sl* - maximum sentence length - Sentence length limit in number of words. This is used to ensure that overly long sentences are not selected as updates.

- *tt* - trigger word threshold - How many words from the event model a sentence must contain to be considered a potential update.

- *rt* - sentence redundancy threshold - The threshold score for a sentence to be considered redundant with another sentence. Used to eliminate redundant updates.

### 7.1.2 Pseudocode

**Algorithm 5** Evolving Individual Event Model pseudo code

```
1: procedure Process(DocumentStream, Event, Parameters)
2:     Results ← empty list
3:     SystemTime ← timestamp
4:     EventModel ← manually grenerated event model
5:     for all document d in DocumentStream do
6:         candidateSentences ← empty list
7:         if Document.titleContainsAll(Event.query) then
8:             SystemTime ← d.timestamp
9:             for all sentence s in d do
10:                if sentence.numWords() < sl and sentence.numberOfWordsContained(EventModel) >= tt then
11:                    candidateSentences.add(s)
12:        updates ← empty list
13:        for all sentence s in candidateSentences do
14:            duplicate ← false
15:            for all sentence r in Results do
16:                similarity ← jaccardCoefficient(r, s)
17:                if similarity > rt then
18:                    duplicate ← true
19:            if duplicate = false then
20:                updates.add(s, SystemTime)
21:        Results.add(updates)
       return Updates
```

This solution was implemented in multiple steps, the first version featured a generic event model that attempted to model sentences that contained new information about an event. The terms used for constructing this event model is generic terms that can be expected to show up in the coverage of a large variety of catastrophic events. Examples of such terms include “killed”, “confirmed”, “casualties”, “damage” and so on. This attempted to capture sentences that contains some information about what has happened during an event. The backside of this solution is that these generic terms might not be able to capture more specific sentences that use vocabulary related to the event class or information about this particular event such as location. Such sentences are often very interesting.

The second version of the solution replaced the generic event model with event models custom made for each event type. This allowed the solution to identify more interesting updates and less updates that were unrelated to the event.

The results below show the difference between the generic and the type specific event models.

### 7.1.3 Results

| **Parameters** | **Updates** | **Gain** | **Comp.** | **F-Measure** |
|---|---|---|---|---|
| generic sl75 rt0.75 tt2 ms 10 | 100.3077 | 0.0601 | 0.0774 | 0.0629 |
| generic sl75 rt0.75 tt2 ms 20 | 1067.3333 | 0.0371 | 0.2406 | 0.0613 |
| generic sl75 rt0.75 tt2 ms 30 | 1389.3333 | 0.0372 | 0.2684 | 0.0604 |
| generic sl75 rt0.75 tt2 ms 50 | 2565.0667 | 0.0283 | 0.2897 | 0.0481 |

Figure 7.1: Average values of performance measures for the manual generic event model

| **Parameters** | **Updates** | **Gain** | **Comp.** | **F-Measure** |
|---|---|---|---|---|
| type sl100 tt1 rt0.75 | 1434.8667 | 0.0375 | 0.2024 | 0.0559 |
| type sl50 tt2 rt0.75 | 207.6000 | 0.1421 | 0.0733 | 0.0701 |
| type sl75 tt2 rt0.75 | 232.8000 | 0.1516 | 0.0949 | **0.0837** |
| type sl100 tt2 rt0.75 | 243.1333 | 0.1475 | 0.0956 | 0.0808 |
| type sl75 tt3 rt0.75 | 47.8000 | 0.1004 | 0.0352 | 0.0446 |

Figure 7.2: Average values of performance measures for the manual event spesific model

As the results show, the version with event models tailored to each event type performed much better than the generic version. One of the problems with the generic event model is the selection of sentences that are not related to the event. This often occurred when documents contained sentences that were not related to the main article. This is not uncommon because the documents of the stream corpus often contain navigational elements and similar unrelated text from websites. This problem often occurred when documents containing unrelated headlines relating to crime or similar. The vocabulary of crime is overlapping with the vocabulary of disasters to some degree, and were some times selected because they contained terms from the event model. They also often contain words that are different from the words used in documents describing disasters, and are therefore likely to be considers novel as well as relevant. Some events would only produce updates that had nothing to do with the event in question, causing the solution to receive a very low score for these events. This severely reduced the overall performance of the system.

This problem is present in every solution that is based on event models, but is especially pronounced when the event model is generic or imprecise. If a document contains sentences that are unrelated to to the main document, and the terms used for identifying potential updates are of a generic nature, they have a much greater chance of being selected than if the terms are more specific to the event in question.

Adjusting the *tt* (trigger word threshold) parameter also contributes to reducing this problem when using a non-generic event model because the more terms are required to match, the less likely it is that an unrelated sentence will contain multiple terms from the event model.

Generic event model also have a problem with scaling, the more event type a model is designed to cover, the more generic it has to be and it will perform worse.

When testing the event type model, increasing the trigger word threshold to 2 instead of 1 improved the solution, this also eliminated some of the problems where updates were selected from irrelevant documents, as sentences from unrelated stories are much less likely to contain two trigger words as opposed to one.

Adjusting the *tt* greatly affected the score of the system. Because of the size of a normal sentence, the usable range for this parameter is not very big unless the event model contains a large amount of terms. A value of 1 caused the solution to select over 1000 updates per event on average which is a vary large amount of updates. A value of 3 dropped the number of updates very low and the comprehensiveness score of the system dropped by a considerable amount. Because of the verbosity normalization of scores, short relevant sentences are very attractive, and a too high *tt* value makes it very unlikely that such updates are selected. A value of 2 significantly reduces the amount of irrelevant sentences because the likelihood of an unrelated sentence containing two event related terms is much less than one. Increasing this value from 1 to 2 increased the score of the system with a large amount.

## 7.2 Event Model Design

This solution is based on event type models and follows the same structure as the manual solution. This solution is however completely automatic. Instead of using an manually created event model, an event model is generated by finding terms that are related to the terms in the event query and using these terms as an event model. The process of finding terms related to a given vector is described in chapter 6.

There are many ways to automatically generate an event model, but the rules of the task puts some restrictions on how we can approach this problem. A good way of doing this would be to use a classifier that learned the rules of sets of terms that can discover good updates. Because of the real time requirement for this application and lack of training data this is not applicable in this case. Another way of generating an event model is using query expansion. Query expansion is used for finding terms that are related to a query in the hope that adding these terms to the original query will give more or better results. Query expansion requires an initial query for finding related terms. There are two available pieces of data for each event that can be used for this purpose. The first is using the event query, the other is using the event type.

Generating the event model using the event type will essentially attempt to automatically generate the event models used by the manual solution. A potential advantage of using the event type for generating a model is that if the type of event is precisely defined, information from similar events could be used to generate a model with strong event type specific vocabulary. Unfortunately, many of the event types for the track are very generic. The event type “storm” for instance covers both hurricanes and cold waves. The vocabulary for a storm type event might be terms such as “wind” and “rain”, while cold waves would most likely not contain these terms. This negates any advantage that could be gained from a specific event type. Another downside compared to using the query is that it might not capture information that is specific to the location or time of the particular event in question.

Using the query for the event can make the vocabulary more specific because the queries used are often more specific. The argument for using the query is that every event is different in some way and is not necessarily best modelled by a model made to fit all events of the same type. Queries often also include some form of location for the event, which is very useful. The theoretical downside to using the event query is that the combinations of terms in the query might not be related to any similar events. This could occur if we try to expand the query “Oslo tornado” but there is no record of a tornado happening in Oslo before in the corpus used for expansion. Documents about Oslo or tornadoes might still be selected and provide terms related to these separate topics, but the results might have been better by simply expanding the event type “tornado” and avoiding potential noisy expansion terms related to “Oslo”.

Below are examples of event models generated for the event “2011-13 Russian protests” using the query and event type respectively. The parameters used to generate this event model is *nd* 40 and *ms* 8.

*eventModel(russian protests, 40, 8) =*
{*russian, protests, nyt, moscow, russia, russias, vladimir, government*}

Figure 7.3: Event model generated for the query “russian protests”

*eventModel(protest, 40, 8) =*
{*protest, protesters, antiwar, opposition, president, party, said, rally*}

Figure 7.4: Event model generated for the event type “protest”

### 7.2.1 Parameters

- sl - maximum sentence length - Sentence length limit in number of words. This is used to esure that overly long sentences are not selected as updates.
- tt - trigger word threshold - How many words from the event model a sentence must contain to be considered a potential update.
- rt - sentence redundancy threshold - The threshold score for a sentence to be considered redundant with another sentence. Used to eliminate redundant updates.
- ms - event model size - Number of terms used in event model.
- nd - top terms document pool - The number of documents to consider when generating event model.

### 7.2.2 Pseudocode

Since this solution is the same except for the generation of the event model, the pseudo code below 7 only includes a demonstration of how this event model is generated using the methods discussed. The implementation of the *expandQuery* method is listed in chapter 6. In this example, the event query is used for generating the event model.

**Algorithm 6** Event model generation

```
1: procedure Process(DocumentStream, Index, Event, Parameters)
2:     EventModel ← Index.expandQuery(Event.query, nd, ms)
```

### 7.2.3 Results

To see if it was possible to increase the performance of the system by using another form of redundancy detection, a solution based on TF-IDF and

| **Parameters** | **Updates** | **Gain** | **Comp.** | **F-Measure** |
|---|---|---|---|---|
| sl75 tt2 rt0.75 ms30 nd6 | 808.7692 | 0.0608 | 0.1630 | 0.0771 |
| sl75 tt2 rt0.75 ms40 nd8 | 825.0000 | 0.0667 | 0.1926 | **0.0861** |
| sl75 tt2 rt0.75 ms50 nd8 | 728.4615 | 0.0670 | 0.1677 | 0.0858 |
| sl75 tt2 rt0.75 ms60 nd8 | 673.0000 | 0.0665 | 0.1681 | 0.0847 |
| sl75 tt2 rt0.75 ms70 nd10 | 685.5000 | 0.0603 | 0.1438 | 0.0758 |
| sl75 tt2 rt0.75 ms80 nd10 | 731.0714 | 0.0564 | 0.1571 | 0.0731 |
| sl75 tt2 rt0.75 ms100 nd50 | 895.3571 | 0.0537 | 0.1829 | 0.0765 |

Figure 7.5: Average values of performance measures of event model

cosine similarity for comparison of sentences was tested. This way of comparing sentences to detect duplicates severely reduced the performance of the system. A likely explanation for this is that TF-IDF scoring puts more weight on the more important terms. This can cause sentences that are different but share trigger words to be discarded because they are believed to be redundant. Because sentences are selected based on the presence of trigger words, they are more likely than other terms to be present in multiple selected sentences. Even with a similar number of updates per event, the performance drop was considerable. Because of this, it was decided that the use of the Jaccard coeficcint is reasonable and that effort should be put into making better event models.

The event model based on the event query consistently outperformed the event model based on the event. This was the expected behavior. Even if the query for an ongoing event is not very precise, the queries for the test events are still more specific that the event types. The query “european cold wave” is s clearly about a cold wave, even if “european” is a very broad term. The event type “storm” on the other hand could be any type of weather related event.

## 7.3 Evolving Event Model Design

The evolving individual event model is based on the same fundamental techniques as the event type model. It functions in the same way except the event model is expanded and adapted as the event in question unfolds.

The argument for this type of solution is that every event is different in some way and even if earlier events are very similar, creating an accurate event model from past events is not perfect. The ideal solution would be to tailor the event model specifically to every event as we start to summarize it. Because we do not want to manually create this event model we need some automatic way to fit the models individually without having tom know how the event will unfold before it does. Another reason to implement an evolving solution is that only a few participants have used this approach in the past.

To ensure that event models are well suited to a particular event and remains this way for the lifespan of the event, an initial event model is generated and is evolved as the event is summarized. This will make the event model more specialized over time and could also follow the development of an event if the event changes over time.

The way this is implemented is that each summarizer starts out with an event model generated in the same way as in the event model solution. Selecting documents are still done by matching the title of each document. When a relevant document is identified and one or more sentence is determined to be a good update according to the event model, the model is modified based on the important terms from these sentences.

When the summarizer is initialized, the terms in the event model is given an initial score. This score is a measure of the importance of each term, and the terms most related to the event type are given a higher score.

The summarizer keeps a map of every related term and as more terms are selected from chosen sentences, we increment the score of these terms by the TF-IDF value of the term in each sentence. This way, terms that occur in many updates will gradually receive a higher score. The event model is defined as the N terms with the highest score from this set of terms. The initial terms start out with a score that is higher than the TF-IDF score of an important term in a sentence, but if they are not found in any updates they will soon be replaced with frequent and important terms from the selected updates.

In theory, if unrelated documents or updates are chosen early in the summarization, it could make the model drift but this is offset by the score of the initial terms in the event model and the fact that the selection of documents are not based on the event model. This makes it unlikely that any large drift could occur, as it would require multiple irrelevant documents and updates to be selected in succession.

### 7.3.1 Parameters

Below is an overview of the parameters used by this solution.

- sl - maximum sentence length - Sentence length limit in number of words. This is used to esure that overly long sentences are not selected as updates.
- tt - trigger word threshold - How many words from the event model a sentence must contain to be considered a potential update.
- rt - sentence redundancy threshold - The threshold score for a sentence to be considered redundant with another sentence. Used to eliminate redundant updates.
- ms - event model size - Number of terms used in event model.
- nd - top terms document pool - The number of documents to consider when generating event model.
- nt - number of expansion terms - The number of terms to extract from a selected sentence for evolving event model.
- it - initial event model term score - The initial score of the term in the event model most closely related to the event type.

### 7.3.2 Pseudocode

**Algorithm 7** Evolving Individual Event Model pseudo code

```
 1: procedure PROCESS(DocumentStream, Index, Event, Parameters)
 2:     $Results \leftarrow$ empty list
 3:     $SystemTime \leftarrow timestamp$
 4:     $ImportantTerms \leftarrow$ Index.expandQuery(Event.query, nd, ms)
 5:     $EventModel \leftarrow$ selectTopTerms(ImportantTerms, ms)
 6:     for all document d in DocumentStream do
 7:         $candidateSentences \leftarrow$ empty list
 8:         if $Document.titleContainsAll(Event.query)$ then
 9:             $SystemTime \leftarrow d.timestamp$
10:             for all sentence s in d do
11:                 if $sentence.numWords() < sl$ and $sentence.numberOfWordsContained(EventModel) >= tt$ then
12:                     $candidateSentences.add(s)$
13:         $updates \leftarrow$ empty list
14:         for all sentence s in candidateSentences do
15:             $duplicate \leftarrow false$
16:             for all sentence r in Results do
17:                 $simmilarity \leftarrow$ jaccardCoefficient(r, s)
18:                 if $simmilarity > rt$ then
19:                     $duplicate \leftarrow true$
20:             if $duplicate = false$ then
21:                 $updates.add(s, SystemTime)$
22:         for all sentence u in updates do
23:             for all (term, score) p in Index.topTermsInDocument(sentence, nt) do
24:                 if $ImportantTerms.containsKey(p.term) = false$ then
25:                     $ImportantTerms.incrementKey(p.term, p.score)$
26:                 else
27:                     $ImportantTerms.add(p.term, p.score)$
28:             $EventModel \leftarrow$ selectTopTerms(ImportantTerms, ms)
29:         $Results.add(updates)$
        return $Updates$
```

### 7.3.3 Results

| **Parameters** | **Updates** | **Gain** | **Comp.** | **F-Measure** |
|---|---|---|---|---|
| v1 sl75 tt2 rt0.75 ms8 nd40 nt1 it10 | 69.6923 | 0.0729 | 0.0913 | 0.0768 |
| v1 sl75 tt2 rt0.75 ms8 nd40 nt1 it15 | 79.0000 | 0.0782 | 0.1115 | 0.0880 |
| v1 sl75 tt2 rt0.75 ms8 nd40 nt1 it20 | 77.3077 | 0.0718 | 0.1262 | 0.0851 |
| v2 sl75 tt2 rt0.75 ms8 nd40 nt1 it10 | 112.1538 | 0.0897 | 0.1261 | **0.1025** |
| v2 sl75 tt2 rt0.75 ms8 nd40 nt1 it15 | 137.8462 | 0.0825 | 0.1261 | 0.0955 |
| v2 sl75 tt2 rt0.75 ms8 nd40 nt1 it20 | 135.9231 | 0.0771 | 0.1191 | 0.0904 |
| v3 sl75 tt2 rt0.75 ms8 nd40 nt1 it10 | 217.3077 | 0.0891 | 0.1601 | **0.1090** |

Figure 7.6: Average values of performance measures of Evolving event model

The results from this solution was a very noticeable improvement from the static event model. The evolution of each individual model helps the summarizer better catch the nuances of each event and ensure that

The first version of this solution expanded the event model by selecting a number of top terms from the document sentences were selected from. This was later changed so that the summarizer instead extracts a fewer number of terms from the individual chosen sentences.

This improved the results, as can be seen in the results table below, the first and second version is labeled *v1* and *v2* respectively. This increase in performance can be explained by the reduced risk of selecting noisy expansion terms. If the selected terms are all from sentences with good match with the trigger words, these sentence are more likely to be on subject than other sentences from the same document, making it less likely that we select terms that are very specific to the document but maybe not to the event at hand.

The third version was a late fix where some text filters were removed from the query expansion module. The text filter was meant to cover an error that turned out was not relevant and could be safely removed. This improved the performance across the board by a small margin, as the filtered terms were terms that was likely to be selected by the query expansion module.

The *it* (initial score of expansion terms) parameter turned out to be important for how the summarizer performed. The tuning of this parameter is somewhat determined by the normal TF-IDF score of the top terms in a given selected sentence. This is because value dictates how quickly the initial score is overtaken by fresh terms. The average TF-IDF score of the top term in a sentence seems to be around 4-6 depending on the sentence. To make sure that the least important terms are replaced first, the *it* parameter only dictates the score of the best term, the others are decremented by the rank of the term. With an initial term score of 10, the best terms are overtaken after about 3-4 hits in selected sentences, while the lower ranked terms are replaced are overtaken after 1-2. It is important to remember that these terms are not removed and if they occur in later sentences their score will increase again and they may be used in the event model. A low *it* value makes the event model more responsive to changes in the coverage of the event, while a high *it* value makes for a more rigid model. The results seems to indicate that a responsive model performs better.

# Chapter 8

# Evaluation of Results

In general the results achieved with the evolving event model based solution performed close to the best participants of 2014. There is a clear progression from the initial baseline, to the query expansion based event model to the evolving event model. Showing that the solution is at least competitive. A strength of the solution not shown by this score is that it is easy to combine with other solutions, an that there are potential to make this approach work event better by refining one or more of the stages in the process.

Below is a table with the results from the 2014 run and the results from the different solutions from the previous chapter.

| **Run Id** | **Gain** | **Comp.** | **F-Measure** |
|---|---|---|---|
| CUNLP-2APSal | 0.0631 | 0.322 | 0.1162 |
| BJUT-Q1 | 0.0657 | 0.4088 | 0.1110 |
| BJUT-Q0 | 0.0632 | 0.3979 | 0.1091 |
| BJUT-Q2 | 0.0632 | 0.3979 | 0.1091 |
| **evolving sl75rt0.75tt2nd40ms8nt1it10** | **0.0891** | **0.1601** | **0.1090** |
| uogTr-uogTr2A | 0.0467 | 0.4453 | 0.0986 |
| **event sl75rt0.75tt2nd8ms40** | **0.0697** | **0.1876** | **0.0859** |
| uogTr-uogTr4AC | 0.0347 | 0.4539 | 0.0793 |
| uogTr-uogTr4ARas | 0.0387 | 0.3691 | 0.0772 |
| IRIT-KW30H5NW3600 | 0.0383 | 0.3521 | 0.0723 |
| IRIT-KW30H5NW300 | 0.0378 | 0.3538 | 0.0714 |
| uogTr-uogTr4A | 0.0281 | 0.4733 | 0.0677 |
| average | 0.0327 | 0.3615 | 0.0620 |
| IRIT-KW80H5NW3600 | 0.0289 | 0.3764 | 0.0604 |
| CUNLP-1APSalRed | 0.0325 | 0.3058 | 0.0602 |
| IRIT-KW30H10NW300 | 0.0298 | 0.378 | 0.0602 |
| IRIT-KW80H5NW300 | 0.0285 | 0.3806 | 0.0596 |
| ICTNET-run3 | 0.0531 | 0.1081 | 0.0530 |
| BUPT-PRIS-Cluster4 | 0.0155 | 0.2692 | 0.0508 |
| IRIT-KW80H10NW300 | 0.0225 | 0.4012 | 0.0503 |
| BUPT-PRIS-Cluster3 | 0.0115 | 0.338 | 0.0407 |
| CUNLP-3AP | 0.0174 | 0.4265 | 0.0403 |
| ICTNET-run2 | 0.0418 | 0.0934 | 0.0311 |
| BUPT-PRIS-Cluster2 | 0.0059 | 0.3728 | 0.0222 |
| ICTNET-run4 | 0.0079 | 0.407 | 0.0178 |
| ICTNET-run1 | 0.007 | 0.409 | 0.0160 |
| BUPT-PRIS-Cluster1 | 0.0033 | 0.4369 | 0.0127 |

Figure 8.1: The results from the 2014 track, runs from this thesis are in **bold**

Below is the results for each individual event for the evolving event model based solution created for this thesis.

| **Event Id** | **Updates** | **Gain** | **Comp.** | **F-Measure** |
|---|---|---|---|---|
| 11 | 487.0000 | 0.0104 | 0.0256 | 0.0148 |
| 12 | 8.0000 | 0.1168 | 0.0405 | 0.0602 |
| 13 | 78.0000 | 0.0613 | 0.0972 | 0.0752 |
| 14 | 255.0000 | 0.0000 | 0.0000 | 0.0000 |
| 15 | 17.0000 | 0.0076 | 0.0057 | 0.0065 |
| 16 | 56.0000 | 0.2312 | 0.4325 | 0.3014 |
| 17 | 7.0000 | 0.0123 | 0.0076 | 0.0094 |
| 18 | 147.0000 | 0.0952 | 0.2069 | 0.1304 |
| 19 | 13.0000 | 0.1158 | 0.0826 | 0.0964 |
| 20 | 20.0000 | 0.0879 | 0.0567 | 0.0689 |
| 21 | 42.0000 | 0.0400 | 0.0276 | 0.0327 |
| 22 | 40.0000 | 0.0095 | 0.0084 | 0.0089 |
| 25 | 3.0000 | 0.0000 | 0.0000 | 0.0000 |

Figure 8.2: Results of the best run for every event

The scores achieved by the participants of the 2014 track were lower across the board than the scores achieved by the teams in 2013. One possible reason for this could be that some of the events of the 2014 track was harder than the ones in the 2013 runs. The results from the evolving event model shows that some events had a very low score compared to the rest, but this could be a coincidence. Because the results published does not include results for every event, it is unknown if the other participants had similar results or not.

The event model solutions achieved a very high score in the gain metric, but a low score in the comprehensiveness metric. A likely explanation for this result is that the strict selection of documents prevents the system from selecting unrelated documents, but also leaves out documents that could be relevant.

## 8.1 Future Work

An advantage to the evolving event model solution is that each step in the process is independent. This means that each step can be modified without having to modify the other stages in the solution.

Because the systems did not score very well in the comprehensiveness metric, it could be a good idea to improve the way documents are selected so that it is not as restrictive. One approach to this is to use the evolving event model for selecting documents as well as sentences. This would have to be tuned to avoid selecting too many documents but would also likely improve comprehensiveness by selecting more diverse documents.

Another possible improvement is to use more sentence features than term presence when selecting sentences. Sentence extraction often use features such as sentence position and some teams from the 2013 track used presence of named entities or numbers to better differentiate between good and bad updates. This can relatively easily be combined with an evolving event model, the challenge is to identify features that provide meaningful information and does not reduce comprehensiveness.

# Chapter 9

# Conclusion

This thesis had three main goals.

1. Create a real time temporal summarization system based on research of other system and related technologies.
2. To test different approaches to the problem and evaluate and evaluate how the different approaches perform.
3. To compare the results of the system with the results of comparable systems to evaluate the methods used.

The main goal of this thesis was to implement a real time temporal summarization system and to compare the results with similar systems to evaluate the techniques used.

A temporal summarization system was created based on the requirements of the 2014 TREC Temporal Summarization track. Based on the experience gained with each cycle of implementation, better solutions were created. The final product is a solution that use an evolving event model to follow the development of events as they are summarized.

The results of the different versions were tested according to standard evaluation criteria and compared with each other as well as the results of the participants of the 2014 Temporal Summarization track. The results show that the solutions got better each iteration and that the results of the final version was comparable to the top teams of 2014.

# Bibliography

[1] J. Allan, J. G. Carbonell, G. Doddington, J. Yamron, and Y. Yang. Topic detection and tracking pilot study final report. In *Proceedings of the Broadcast News Transcription and Understanding Workshop (Sponsored by DARPA)*, 1998.

[2] J. Allan, R. Gupta, and V. Khandelwal. Temporal summaries of news topics. In *SIGIR 2001: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, September 9-13, 2001, New Orleans, Louisiana, USA*, pages 10–18, 2001.

[3] J. Allan, C. Wade, and A. Bolivar. Retrieval and novelty detection at the sentence level. In *SIGIR 2003: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, July 28 - August 1, 2003, Toronto, Canada*, pages 314–321, 2003.

[4] J. Aslam, F. Diaz, M. Ekstrand-Abueg, R. McCreadie, V. Pavlu, and T. Sakai. TREC 2014 temporal summarization draft guidelines. In *The Twenty-Third Text REtrieval Conference (TREC 2013) Proceedings*, 2014.

[5] J. Aslam, F. Diaz, M. Ekstrand-Abueg, R. McCreadie, V. Pavlu, and T. Sakai. TREC 2014 temporal summarization metrics. In *The Twenty-Third Text REtrieval Conference (TREC 2013) Proceedings*, 2014.

[6] J. Aslam, F. Diaz, M. Ekstrand-Abueg, R. McCreadie, V. Pavlu, and T. Sakai. TREC 2014 temporal summarization track overview. In *The Twenty-Third Text REtrieval Conference (TREC 2014) Proceedings*, 2014.

[7] J. Aslam, F. Diaz, M. Ekstrand-Abueg, V. Pavlu, and T. Sakai. TREC 2013 temporal summarization. In *The Twenty-Second Text REtrieval Conference (TREC 2013) Proceedings*, 2013.

[8] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval - the concepts and technology behind search, Second edition*. Pearson Education Ltd., Harlow, England, 2011.

[9] L. Chen, H. Zhang, S. Li, Z. Ji, Q. Liu, Y. Liu, D. Wu, and X. Cheng. ICTNET at temporal summarization track TREC 2014. In *The Twenty-Third Text REtrieval Conference (TREC 2014) Proceedings*, 2014.

[10] L. Chen, H. Zhang, S. Li, Z. Ji, Q. Liu, Y. Liu, D. Wu, and X. Cheng. IRIT at TREC temporal summarization 2014. In *The Twenty-Third Text REtrieval Conference (TREC 2014) Proceedings*, 2014.

[11] Q. Guo, F. Diaz, and E. Yom-Tov. Updating users about time critical events. In *Advances in Information Retrieval - 35th European Conference on IR Research, ECIR 2013, Moscow, Russia, March 24-27, 2013. Proceedings*, pages 483–494, 2013.

[12] C. Kedzie, K. McKeown, and F. Diaz. Columbia university at trec 2014: Temporal summarization. In *The Twenty-Third Text REtrieval Conference (TREC 2014) Proceedings*, 2014.

[13] Q. Liu, Y. Liu, D. Wu, and X. Cheng. ICTNET at temporal summarization track TREC 2013. In *The Twenty-Third Text REtrieval Conference (TREC 2013) Proceedings*, 2013.

[14] H. P. Luhn. The automatic creation of literature abstracts. *IBM Journal of Research Development*, 2(2):159–165, 1958.

[15] R. McCreadie, M.-D. Albakour, S. Mackie, N. L. C. Macdonald, I. Ounis, and B. T. Dinçer. University of glasgow at TREC 2013: Experiments with terrier in contextual suggestion, temporal summarisation and web tracks. In *The Twenty-Third Text REtrieval Conference (TREC 2013) Proceedings*, 2013.

[16] R. McCreadie, R. Deveaud, M.-D. Albakour, S. Mackie, C. Macdonald, I. Ounis, T. Thonet, and B. T. Dinçer. University of glasgow at TREC 2014: Experiments with terrier in contextual suggestion, tempo-

ral summarisation and web tracks. In *The Twenty-Third Text REtrieval Conference (TREC 2014) Proceedings*, 2014.

[17] K. McKeown, R. J. Passonneau, D. K. Elson, A. Nenkova, and J. Hirschberg. Do summaries help? In *SIGIR 2005: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Salvador, Brazil, August 15-19, 2005*, pages 210–217, 2005.

[18] M. Peetz, E. Meij, and M. de Rijke. Using temporal bursts for query modeling. *Inf. Retr.*, 17(1):74–108, 2014.

[19] Y. Qi, Q. Wang, C. Huang, B. Tang, and W. Xu. The information extraction systems of BUPT_PRIS at TREC2014 temporal summarization track. In *The Twenty-Third Text REtrieval Conference (TREC 2014) Proceedings*, 2014.

[20] R. Sipos, A. Swaminathan, P. Shivaswamy, and T. Joachims. Temporal corpus summarization using submodular word coverage. In *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pages 754–763, 2012.

[21] R. G. Vikash Khandelwal and J. Allan. An evaluation corpus for temporal summarization. In *Proceedings of the First International Conference on Human Language Technology Research, 2001*, 2001.

[22] Y. Xi, B. Li, J. Zhou, and Y. Tang. ZZISTI at TREC2013 temporal summarization track. In *The Twenty-Third Text REtrieval Conference (TREC 2013) Proceedings*, 2013.

[23] T. Xu, P. McNamee, and D. W. Oard. HLTCOE at trec 2013: Temporal summarization. In *The Twenty-Third Text REtrieval Conference (TREC 2013) Proceedings*, 2013.

[24] Z. YANG, F. YAO, H. SUN, Y. ZHAO, Y. LAI, and K. FAN. BJUT at TREC 2013 temporal summarization track. In *The Twenty-Third Text REtrieval Conference (TREC 2013) Proceedings*, 2013.

[25] C. Zhang, W. Xu, F. Meng, H. Li, T. Wu, and L. Xu. The information extraction systems of PRIS at temporal summarization track. In

*The Twenty-Third Text REtrieval Conference (TREC 2013) Proceedings*, 2013.

[26] Y. Zhang, J. P. Callan, and T. P. Minka. Novelty and redundancy detection in adaptive filtering. In *SIGIR 2002: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 11-15, 2002, Tampere, Finland*, pages 81–88, 2002.

[27] Y. Zhao, F. Yao, H. Sun, and Z. Yang. BJUT at TREC 2014 temporal summarization track. In *The Twenty-Third Text REtrieval Conference (TREC 2014) Proceedings*, 2014.

# Appendices

# Appendix A

# System Guide

This short manual is a short guide to how the various parts of the temporal summarization system can be started.

The corpus data is not included in the source delivery for size reasons. Every part of the system is located in the `temporal-summarizer` folder. Contact Kjetil Nørvåg gain access to this folder.

## A.1 Preprocessing

WARNING! Do not perform preprocessing unless absolutely necessary. The procedure takes a couple of days and it is much faster to simply copy the clean version.

The KBA stream corpus is very large and contains a lot of data that is not of any use for the temporal summarization task.

To use the preprocessor navigate to the `temporal-summarizer/preprocessor` folder and run the command below. The `preprocessor.py` file is also the source code of the preprocessor.

```
python preprocessor.py ../corpus ../clean-corpus
```

This will process the stream corpus located in `temporal-summarizer/corpus` and write a cleaned version to the `temporal-summarizer/clean-corpus` folder.

To be able to run the preprocessor the system, must install the streamcorpus tools[1] and the system must have a key for the KBA stream corpus [2]. The key is located at `temporal-summarizer/trec-kba.rsa` and must be added to the host machines gpg keychain for the preprocessor to be able to decrypt the corpus.

## A.2 Temporal Summarization System

To start the temporal summarization system navigate to the `temporal-summarizer` folder and run the following command.

```
sh launch.sh
```

This file contains the following script to start the system.

```
java -jar -Xmx12000M -Xms4000M temporal-summarizer.jar parameters.txt
```

The `parameters.txt` file contains the launch parameters for the system and can be modified to run different solutions with different parameters.

Below is a list of what parameters are used for the different solutions. The explanation of these parameters can be found in chapter 7. The first parameter dictates what solution are initiated.

1. **Event Model**(event) - sl, rt, tt, nd, ms
2. **Evolving Event Model**(evolving) - sl, rt, tt, nd, ms, nt, it

Processing the corpus and generating updates for all example events will take around 3 hours depending on how fast the host computer can read the corpus and how many solutions are tested in a single run.

The system will generate results for each solution/event combination and write the results to the `temporal-summarizer/result-processing/ts-results.tsv` file.

[1]http://streamcorpus.org/
[2]http://trec-kba.org/kba-stream-corpus-2014.shtml

Because of a late change to the system, the results generated by the current version of the system might be slightly better compared to some of the results presented in chapter 7. The results in chapter 8 are all final and should be identical to the results generated.

## A.3 Evaluation Script

The evaluation script is provided by TREC and can be run by navigating to the `temporal-summarizer/result-processing` folder and running the following command.

```
sh evaluate.sh
```

This will initiate the command below to read file with updates from the temporal summarization system and evaluate them according to the evaluation criteria of the track.

```
python tseval.py ts-results.tsv > results.txt
```

The results can be observed by reading the `results.txt` file. The format of this file is explained in [4]. The values used when evaluating the systems in chapter 7 and 8 is nE[Latency Gain], Latency Comp. and HM(nE[LG],Lat. Comp.).