Design and Evaluation of Temporal Summarization Systems

by

Rakesh Guttikonda

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2014

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Temporal Summarization (TS) is a new track introduced as part of the Text REtrieval Conference (TREC) in 2013. This track aims to develop systems which can return important updates related to an event over time. In TREC 2013, the TS track specifically used disaster related events such as earthquake, hurricane, bombing, etc. This thesis mainly focuses on building an effective TS system by using a combination of Information Retrieval techniques. The developed TS system returns updates related to disaster related events in a timely manner.

By participating in TREC 2013 and with experiments conducted after TREC, we examine the effectiveness of techniques such as distributional similarity for term expansion, which can be employed in building TS systems. Also, this thesis describes the effectiveness of other techniques such as stemming, adaptive sentence selection over time and de-duplication in our system, by comparing it with other baseline systems.

The second part of the thesis examines the current methodology used for evaluating TS systems. We propose a modified evaluation method which could reduce the manual effort of assessors, and also correlates well with the official track's evaluation. We also propose a supervised learning based evaluation method, which correlates well with the official track's evaluation of systems and could save the assessor's time by as much as 80%.

Acknowledgements

Firstly, I would like to thank my advisor, Dr. Olga Vechtomova, for providing continuous support and guidance throughout my Master's program for almost 2 years. I am also grateful to her for giving me the freedom to choose the research problem and also offering me necessary guidance in my approach to solve it.

I would like to express my gratitude to Prof. Gordon Cormack and Prof. Charles Clarke for taking the time to read my thesis.

Special thanks to my close friend and lab mate, Gaurav, for all the insightful discussions in work and on personal front. I am also indebted to my close friends: Shrinu, Sandy, Sardaar, Shivam and others, for making my stay at Waterloo a memorable one.

Also, many thanks to my friends in the lab: Adam, Bahar, Ashif and others, who were always ready to help.

Finally, I would like to thank my parents and my lovely little sister, for supporting me all these years while pursuing my dreams. I would like to specially thank my fiancée Radhika for her love and unwavering support during the stressful times in the last few months.

Dedication

In loving memory of my grandfather, and to my family and Radha.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Retrieving important information related to a crisis or a natural disaster, especially when it is happening or has just happened, has own set of challanges when compared to traditional Information Retrieval (IR) systems. During the crisis, users urgently require important information related to the crisis, especially if they are going to face it [Aslam et al., 2014]. For example, during the hurricane Sandy[1] which affected eastern part of United States in 2012, it would help the users especially in these affected areas to keep track of important information like the current direction of the storm, or any important safety regulations issued by government etc., as soon as they are issued. There is a need for automatic systems, which can continuously update the user in real time with important information related to the event but at the same time, ensure the quality and novelty of the information presented.

1.2 The Problem

The Temporal Summarization (TS) track has been introduced as part of the Text REtrieval Conference (TREC) in 2013, to solve the problem outlined above. According to the track organizers Aslam et al. [2014], "The goal of TS track is to develop systems for efficiently monitoring the information associated with an event over time". The track has two main

[1] http://en.wikipedia.org/wiki/Hurricane_Sandy

tasks. In the first task, the systems are expected to return short and relevant sentences (or updates) for a particular event over time. Whereas, the second task involves tracking values of pre-defined attributes such as number of injured, or number of dead, etc. related to the event. In this thesis, we mainly focus on the first task, i.e., building a TS system that returns relevant updates for disaster related events over time.

The track's problem statement for the first task can be formulated as follows:

Input:

1. Time ordered stream of documents, i.e., KBA-2013 stream corpus[2].
2. Query in the form of xml, with the following attributes:
   (a) start time, i.e., minimum timestamp of a document from which the system can retrieve an update.
   (b) end time, i.e., maximum timestamp of a document from which the system can retrieve an update.
   (c) query.
   (d) query type $\in$ {*accident, bombing, earthquake, shooting, storm*}.
   (e) attributes $\in$ {*deaths, displaced, financial impact, injuries, locations*}.

Output: The following in a tab-separated file format for all updates:

1. Query Identifier.
2. Team Identifier.
3. Run Identifier.
4. Document Identifier (as in the KBA Corpus).
5. Sentence Identifier (i.e., Base zero index of the sentence in the document above).
6. Decision timestamp (i.e., time at which the system has made the decision, must be greater than or equal to the document timestamp).
7. Confidence Value (a strictly positive number which measures the system's confidence in this sentence being a reasonable update).

[2] http://trec-kba.org/kba-stream-corpus-2013.shtml

```
<event>
<id>TRAIN-1</id>
<title>2012 East Azerbaijan earthquakes</title>
<start>1344687797</start>
<end>1345551797</end>
<query>iran earthquake</query>
<type>earthquake</type>
<locations/>
<deaths/>
<injuries/>
</event>
```

Figure 1.1: Training Query: "iran earthquake" used in TST 2013.

For example, Figure 1.1 shows the training query released for the TS track. The query starts at 1344687797 (UNIX timestamp in GMT), i.e., 2012-08-11-12 (in yyyy-mm-dd-hh format), and spans for a period of 10 days i.e. 240 hours. All the updates (i.e., sentences) retrieved by a system should be from the documents within this period.

In addition, participants are required to adhere to the following rules:

1. Participants cannot use statistical models trained on the data that exists after the event end time.

2. The document statistics like TF-IDF, if used, should be calculated only from the documents that are available before the decision timestamp.

3. External corpus can be used for training purposes, but only if the documents exist before the query start time.

1.3 Our Approach

Unlike most IR systems where the retrieval unit is a document and the document collection is static, the TS problem deals with a different set of conditions. In temporal summarization systems, the retrieval unit is a sentence which has to be retrieved from documents in a time ordered fashion, and the document collection is dynamic (increasing with time). Also, there is no "ranking" for the sentences returned by the TS systems, unlike the ranking of

results (documents) in IR systems. So, evaluation metrics like P@k, MAP, or nDCG (see Büttcher et al. [2010, chap. 12]), are not directly applicable to this track.

One of the main challenges in the TS track is to return sentences with low-latency, and also at the same time, the system needs to maintain the quality of output in the presence of the dynamic corpus. So, the decision for inclusion of a sentence to the list of updates is important and is almost real-time, i.e., as soon as the sentence is seen.

We follow a streamlined approach to build our system using a combination of IR techniques, along with certain heuristics for sentence selection and deduplication. In our approach, we first retrieve the documents for the given query, using Query Likelihood model (i.e. language modeling approach) [Ponte and Croft, 1998]. Then, we work on the set of documents (D) which score higher than the cutoff score, so that we return sentences only from good scoring documents (which are likely to be relevant for the given query), and also reduce processing time.

In order to select and score the sentences from this reduced set of documents D, we use query expansion techniques to expand the query and then score the sentences using Okapi-BM25 [Robertson et al., 1996] with respect to the expanded query. We specifically use Lin's score based distributional similarity algorithm [Lin, 1998] for expanding the query. After scoring the sentences, we use an adaptive cutoff score based sentence selection algorithm to decide whether a sentence should be returned in the output. Also, before returning the sentences in the output, we use deduplication algorithm to check if the current sentence is a near-duplicate to any of the already returned sentences.

Chapters 3 and 4 of this thesis explain the approach in detail and also compare the performance of our system to other baseline systems. Chapter 3 explains our approach used to build the system while participating in TREC 2013 and discusses the performance of our system in TREC 2013. Chapter 4 explains the improvements made to our system after the TREC conference. Chapter 4 also explains the performance of different techniques by comparing to the baseline systems on the evaluation metrics explained in the next section.

1.4 Evaluation

Temporal Summarization track uses nugget-based evaluation, which was first introduced in TREC 2003 as part of the Question Answering (QA) track [Voorhees, 2004]. Voorhees [2004] defines nugget as, ".. a fact for which the assessor could make a binary decision as to whether a response contained that nugget". For the TS track, Aslam et al. [2014] define

nugget as, ".. a very short sentence, including only a single sub-event, fact, location, date, etc., associated with topic relevance".

In order to evaluate the systems in the TS track, for every test query (i.e. an event) the assessors pool a list of nuggets from Wikipedia event pages related to the event, along with their timestamps obtained from the revision history of the page. The TS track defines metrics, namely Expected Latency Gain (ELG) and Latency Comprehensiveness (LC), which are computed based on the nuggets covered in the system's response (i.e. list of updates).

Notation:
Using the notation as followed in Aslam et al. [2014], let N be the set of nuggets pooled by assessors from wikipedia event pages, and let S be the set of updates returned by the system. Let n ($\in N$) denote a single nugget, and u ($\in S$) denote a single update of the system.

An update matching function, $M(n, S)$, is defined as follows [Aslam et al., 2014]:

$$\mathbf{M}(n, S) = argmin_{\{u \in S : n \approx u\}} u.t$$

i.e., nugget n is matched to the earliest update u which contains the nugget n. If there is no matching update u for nugget n, then the set is empty (ϕ). Similarly, the set of all nuggets for which u is the earliest update can be defined as [Aslam et al., 2014]:

$$\mathbf{M}^{-1}(u, S) = \{n \in N : M(n, S) = u\}$$

Now, the two main metrics used for evaluating the systems are defined as follows [Aslam et al., 2014]:

$$\textit{Expected Gain (EG)} = \frac{1}{|S|} \sum_{n \in N : M(n,S) \neq \phi} g(M(n, S), n)$$

$$\textit{Comprehensiveness (C)} = \frac{1}{\sum_{n \in N} R(n)} \sum_{u \in S} \sum_{n \in M^{-1}(u,S)} g(u, n)$$

where,

$$\text{Gain: } g(u, n) = R(n) \times \text{ discount factor (i.e. latency/verbosity penalty of } u)$$

$$R_{graded}(n) = \frac{e^{n.i}}{e^{max_{n' \in N} n'.i}}$$

$$R_{binary}(n) = 1 \text{ if } n.i > 0; \text{ or 0 otherwise;}$$

$$\text{Latency penalty: } L(u,n) = 1 - \frac{2}{\pi} arctan\left(\frac{u.t - n.t}{3600 * 6}\right)$$

$$\text{Verbosity penalty: } V(u) = 1 + \frac{|\text{all words}|_u - |\text{matching words}|_{u,\forall n}}{avg.|\text{words}|_{\forall n}}$$

Expected Latency Gain (ELG) and Latency Comprehensiveness (LC) are calculated from EG and C respectively by including the latency penalty L in Gain g as shown above.

As one can observe, defining the matching function $\mathbf{M}$ is one of the critical steps of evaluation. In the current evaluation method of the track, the assessors manually match the nuggets N to the set of updates S. Given the size of updates ($\approx$ 7816 unique sentences pooled from all the runs for the 9 test queries) and the nuggets($\approx$ 1077 for the 9 test queries), matching is a time consuming process involving around 15 to 20 man hours per query (as per information gathered from the track organizers).

In this thesis, we propose a modified approach for evaluation, which reduces the number of matches to be checked by assessor considerably, thereby improving the productivity. We also then extend the idea to a supervised learning based evaluation method, where there is little human involvement in matching the nuggets to the updates. We show that the ranking of systems using the new evaluation mechanism correlates well with the official track's evaluation, and could save the assessor's time by almost 80%. Chapter 5 of this thesis explains our method in detail along with comparisons to the official track's evaluation and POURPRE [Lin and Demner-Fushman, 2005].

1.5 Contributions

The main contributions of the thesis are the following:

1. We present a novel approach using a combination of IR techniques to build a TS system which can track updates for disaster related events.

2. We show that distributional similarity techniques are useful for finding term expansions and can be used as a query expansion technique in TS systems.

3. We propose a method to automatically identify seed words for the Lin's distributional similarity algorithm [Lin, 1998], and propose a method to automatically merge the related words lists.

4. We propose an adaptive cutoff score based sentence selection algorithm and show that it is useful for building TS systems, where latency is an important factor.

5. We investigate different deduplication techniques for TS system, and show that Cosine similarity is better than other deduplication algorithms like Charikar's simhash [Charikar, 2002], and a baseline percentage match.

6. We propose a modified approach for matching nuggets to sentences by assessors. This method reduces the manual effort required from assessors, thereby improving productivity and facilitating larger scale of evaluation.

7. We propose a supervised learning based evaluation mechanism (using distributional similarity & machine learning approach) for evaluating TS systems, which correlates well with the official track's evaluation. We show that with the new evaluation mechanism assessor's time could be saved by 80% when compared to the original track's evaluation.

Chapter 2

Related Work

This chapter starts with a brief overview of the Text REtrieval Conference (TREC) Question Answering (QA) track which has some similarity to TST. Afterwards, we present an overview of the standard retrieval models and query expansion techniques with methods such as distributional similarity which can be used in building and evaluation of Temporal Summarization systems. In the end, the chapter concludes with the overview of nugget-based evaluation methods and existing automatic nugget based evaluation systems such as POURPRE which we use for comparisons in Chapter 5.

2.1 TREC: QA track

The Text REtrieval Conference (TREC)[1] hosts a series of tracks every year focusing on different research areas and applications in information retrieval. Some of the tracks of TREC-2013 are :

1. Contextual Suggestion track: "... investigates search techniques for complex information needs that are highly dependent on context and user interests."[Dean-Hall et al., 2014]

2. Web track: ".. to explore and evaluate retrieval approaches over large-scale subsets of the Web."[Collins-Thompson et al., 2014]

[1] http://trec.nist.gov/

This year marks the beginning of Temporal Summarization Track (TST)[2] which this thesis mainly focuses upon. The objective and the problem statement for TST were discussed in Chapter 1. TST bears some similarities with the QA track organized by TREC which we explain below.

TREC started the QA track[3] for the first time in 1999 (TREC 8) to change the model of an IR system from retrieving traditional ranked document lists to retrieving answers for the query [Voorhees and Tice, 1999; Voorhees, 2001]. This user model assumes that the users would prefer getting answers rather than search for the answers in the documents returned. Recent tracks like 1CLICK [Sakai and Joho, 2011; Kato et al., 2013] organized at NTCIR[4], focus on this model of presenting information retrieved from web search engines instead of showing URLs of documents. For these tasks, systems are expected to return textual information gathered from various documents directly upon only one click on the search button [Sakai et al., 2011]. Such systems are evaluated on the amount of "information units" (nuggets) the textual output contains.

The QA track conducted from 1999 till 2007 covered the following question types: Factoid based questions (1999-07), List based questions (2003-07), Definition questions/Other (2003-07) and Complex interactive questions (2006-07).

QA track originally started with the factoid based question types. For example, "How many calories are there in a Big MAC?" is a factoid based question [Voorhees, 2004]. The answer is a fact or a short string. By 2006, the factoid answers were expected to be temporally correct [Dang et al., 2007]. For example, "Who is the president of United States?" is one such question, where the current president "Barack Obama" should be the answer but not "Bill Clinton" or any of the previous presidents.

QA track started the List and Definition questions in 2003. List questions are similar to the factoid based questions but have multiple short answers. For example, "List the names of chewing gums." is one such question. For answering such questions, systems need to collect answers from multiple documents [Voorhees, 2004].

Definition questions on the other hand, ask about interesting information of a person or thing and expect more descriptive answers than factoid based questions [Voorhees, 2004]. For example, "Who is Albert Ghiorso?" or "What is a golden parachute?" are definition questions. For definition questions, systems are expected to return snippets extracted from documents that can answer the question [Voorhees, 2004]. Systems are allowed to

[2] http://www.trec-ts.org/
[3] http://trec.nist.gov/data/qamain.html
[4] http://research.nii.ac.jp/ntcir/index-en.html

return multiple snippets for a question, but they should not repeat the information while answering the question [Voorhees, 2004].

The TS track conducted this year bears similarity to the QA track, especially with the definition based questions. Mainly the similarities are:

1. Retrieval unit for both TS track and QA track is textual string instead of document. In TS track, systems are expected to return sentences extracted from documents, where as in QA track systems should return answer strings which can be an entire sentence, or part of a sentence, or multiple sentences together.

2. Both tracks follow nugget based evaluation model. A nugget is a short string of text which covers a important facet of information about the target in the query. Both tracks judge the systems' performance based on the nuggets covered in the response. While TST incorporates graded importance for nuggets, for official track scores TST uses binary importance scores for nuggets similar to *vital* and *okay* categories in the QA track [Voorhees, 2004].

However, both tracks differ in the following:

1. TST uses temporal statistics while evaluating a system's response. A system is expected to return important updates related to the query as soon as possible along with maintaining novelty in the updates. QA track doesn't use such temporal constraints for evaluation.

2. TST track uses a time ordered collection of documents (KBA corpus) and for a given query with time period, only documents before the query start time and during the query time can be used to return the sentences. QA track uses a static collection (AQUAINT corpus [Graff, 2002]) and answer strings in the system's response can be from any document within the corpus.

2.2 Retrieval Models: Probabilistic Retrieval and Language Modeling

This section presents an overview of the standard techniques for retrieving ranked lists of documents from a document collection given a query. In this section, we summarize the retrieval methods explained in Chapters 8 and 9 of Büttcher et al. [2010]. Büttcher et al. [2010] explain the derivations in a detailed manner for the reader to understand easily.

2.2.1 Probabilistic Retrieval

Using the notation followed in Büttcher et al. [2010, page 259], suppose we are given three random variables (r.v.) namely: D for documents (sample space: collection of documents), Q for query (sample space: any textual string accepted by search engine as query) and R for relevance (binary r.v.). The probability that document d is relevant for a query q is given by (as shown in Büttcher et al. [2010, page 259, Eq 8.1]),

$$p(R = 1|D = d, Q = q) \tag{2.1}$$

According to Probability Ranking Principle [Robertson, 1977, page 295], "If an IR system's response to each request is a ranking of the documents in the collections in order of decreasing probability of usefulness to the user who submitted the request, then the overall effectiveness of the system to its users will be the best that is obtainable on the basis of the data", which is equivalent to ranking the documents by 2.1.

Applying Bayes theorem to equation 2.1, we get (as shown in Büttcher et al. [2010, page 260, Eq 8.5]):

$$p(R = 1|D, Q) = \frac{p(D, Q|R = 1)p(R = 1)}{p(D, Q)} \tag{2.2}$$

Now maximizing the probability p, is equivalent to maximizing the odds ratio $\frac{p}{1-p}$, which is equivalent to maximizing $\log(\frac{p}{1-p})$ [Büttcher et al., 2010, page 260]. So, if we apply this transformation to equation 2.1, the relative ordering of documents will not change. "Thus log-odds and probability are *rank-equivalent* (i.e., ranking by one produces the same ordering as ranking by the other)." [Büttcher et al., 2010, page 260].

Hence, transforming equation 2.2 using above, and after expansion of joint probabilities, we get (see Büttcher et al. [2010, page 261, Eq 8.13]):

$$\log \frac{p(D, Q|R = 1)p(R = 1)}{p(D, Q|R = 0)p(R = 0)} \textit{ is rank equivalent to } \log \frac{p(D|Q, R = 1)}{p(D|Q, R = 0)} \tag{2.3}$$

which is the core equation for the probabilistic retrieval.

Robertson et al. [1996] suggested the following equation 2.4, which can be obtained from transformation of above equation as shown in Büttcher et al. [2010],

$$\sum_{t \in q} \left(q_t \times \frac{f_{t,d}(k_1 + 1)}{k_1((1 - b) + b(l_d/l_{avg})) + f_{t,d}} \times w_t \right) \tag{2.4}$$

where, $f_{t,d}$ is the frequency of the term t in document d, w_t is the Inverse Document Frequency (IDF) weight, i.e., $log\frac{N}{N_t}$, where N is the total number of documents and N_t is the number of documents which contain the term t, l_d is length of document, l_{avg} is the average document length and parameters $k_1 = 1.2, b = 0.75$ [Büttcher et al., 2010, page 272].

This equation 2.4, also known as "BM25", is adapted to retrieve sentences for our TS system.

2.2.2 Language Modeling

The language modeling approach was first suggested by Ponte and Croft [1998].

The equation 2.3 discussed in the earlier section, after denoting $R = 1$ as r, and $R = 0$ as $\overline{r}$, can be modified as shown in Büttcher et al. [2010, page 287]:

$$\begin{aligned} \log \frac{p(D,Q|r)p(r)}{p(D,Q|r)p(\overline{r})} &= \log \frac{p(Q|D,r)p(D|r)p(r)}{p(Q|D,\overline{r})p(D|\overline{r})p(\overline{r})} && (2.5)\\ &= \log \frac{p(Q|D,r)p(r|D)}{p(Q|D,\overline{r})p(\overline{r}|D)} && (2.6)\\ &= \log p(Q|D,r) - \log p(Q|D,\overline{r}) + logit(p(r|D) && (2.7) \end{aligned}$$

The last term in the above equation can be ignored, as it is independent of query q and just indicates the prior probability of relevance of document $D = d$. The condition on $D = d$ with non relevant-documents ($\overline{r}$) doesn't correlate well with the user requirements. Hence, the first term $\log p(Q|D,r)$ alone can be used for ranking document $D = d$ with respect to query $Q = q$ [Büttcher et al., 2010, page 288].

Assuming independence between the query terms, the probability can then be calculated as follows (as shown in Büttcher et al. [2010, page 292]):

$$\begin{aligned} p(q|d) &= \prod_{i=1}^{n} p(t_i|d) && (2.8)\\ &= \prod_{t \in q} p(t|d)^{q_t} && (2.9) \end{aligned}$$

Now using the document language model as the Maximum likelihood model, and with Dirichlet smoothing [Chen and Goodman, 1999] over the collection, the probability of a term t appearing in document d becomes (see Büttcher et al. [2010, page 291]):

$$p(t|d) = \frac{f_{t,d} + \mu M_c(t)}{l_d + \mu} \tag{2.10}$$

Hence, substituting the above in equation 2.9, we get:

$$p(q|d) = \prod_{t \in q} \left(\frac{f_{t,d} + \mu M_c(t)}{l_d + \mu} \right)^{q_t} \tag{2.11}$$

After applying the logarithm and simplifying the parameters, we finally arrive at the following equation as shown in Büttcher et al. [2010, page 295, Eq 9.32]):

$$\sum_{t \in q} q_t \cdot log \left(1 + \frac{f_{t,d}}{\mu} \cdot \frac{l_C}{l_t} \right) - n \cdot log \left(1 + \frac{l_d}{\mu} \right) \tag{2.12}$$

where, t is the term, q is the query term vector, q_t is the term frequency of t in the query, $f_{t,d}$ is the term frequency of t in document d, l_C is the length of the collection (the total number of terms in the collection), l_t is the number of times the term t appears in the collection, and μ is the Dirichlet smoothing factor.

We use Equation 2.12 for retrieving documents from the collection for a given query as described in the subsequent chapters.

2.3 Sentence Retrieval

Sentence Retrieval is one of the key components of Temporal Summarization system and hence in this section we provide a brief overview of the existing sentence retrieval methods developed by researchers. Sentence Retrieval was studied earlier by researchers for various tasks such as Question Answering track of TREC, Summarization, Novelty[5] track of TREC. Murdock [2006] provides an excellent survey and background of the aspects involved in Sentence Retrieval.

[5] http://trec.nist.gov/data/novelty.html

Standard document retrieval models such as TF-IDF, BM25, Language Models etc., have been adapted for sentence retrieval by researchers. Allan et al. [2003] adapted a simple approach of TF-IDF at sentence level (i.e. TF-ISF), where the relevance of sentence s for a given query q is given by (as shown in Allan et al. [2003, page 3]),

$$R(s|q) = \sum_{t \in q} log(tf_{t,q} + 1) log(tf_{t,s} + 1) log(\frac{n+1}{0.5 + sf_t})$$

where, t is the term, q is the query term vector, $tf_{t,q}$ is the term frequency of t in q, $tf_{t,s}$ is the term frequency of t in s, n is the number of sentences in collection, sf_t is the number of sentences which contain term t.

Allan et al. [2003] show that finding relevant sentences from relevant documents for the novelty task is much more difficult than detecting novel sentences from relevant sentences. While finding relevant sentences, the authors find that TF-ISF technique performed better (though not statistically significant) than language modeling with KLD and query modeling (with two stage smoothing)[Zhai and Lafferty, 2002] for the TREC 2002 novelty track.

Merkel and Klakow [2007] compare the Language Model retrieval method [Ponte and Croft, 1998] with other methods such as TF-IDF and Okapi-BM25 for the Question Answering track of TREC 2004. The authors show that Language Modeling approach with dirichlet priors performs better than other approaches such as TF-IDF and Okapi-BM25. Metzler et al. [2005] also show that the Query Likelihood model outperforms methods such as TF-IDF and word overlap for identifying relevant sentences for the topics used in Question Answering track. Balasubramanian et al. [2007] find that advanced language modeling techniques such as translation model (Model S) [Murdock, 2006], mixture model [Jeon et al., 2005], relevance models [Lavrenko and Croft, 2001], etc. perform better than the simple Query Likelihood model on the dataset prepared by Murdock [2006] from TREC 2003 QA track.

Various query expansion techniques such as Pseudo-relevance feedback, co-occurrence based expansion of query terms from external corpus, WordNet have also been suggested by researchers to improve the sentence retrieval module in the Novelty track of TREC 2002 [Collins-Thompson et al., 2002; Schiffman, 2002; Zhang et al., 2003].

In this thesis, we don't try to compare different retrieval models or propose a new retrieval model for documents/sentences in TS system, but mainly focus on other techniques/aspects involved in building a Temporal Summarization system for a particular retrieval model. We use the standard retrieval model of Okapi-BM25 as the baseline method for retrieving sentences.

2.4 Automatic Query Expansion

The vocabulary mismatch problem [Furnas et al., 1987] suggests that words used in a query by the user of a web search engine often mismatch with the vocabulary used in the document. So, using any of the standard retrieval models such as Okapi-BM25 or Language Modeling approach, the documents which do not use exactly the same words due to the use of synonyms (e.g. "buy" and "purchase"), plurals (e.g. "colours" instead of "colour"), abbreviations ("U.S." instead of "United States") etc. will not be retrieved or might appear low in the ranked list due to fewer words matching the query.

In order to deal with these vocabulary problems, various techniques have been proposed to improve the recall (covering more relevant documents of all the relevant documents) of IR systems. However, the downside of these methods might be that the precision of systems might go down due to extraneous documents retrieved in the process of expanding the query. A number of techniques have been proposed since early years, like, relevance feedback in the vector space model [Rocchio, 1971], using co-occurence statistics for terms [Harper and Van Rijsbergen, 1978], stemming [Porter, 1980], statistical analysis of term distributions [Doszkocs, 1978] for automatic query expansion.

A detailed survey on AQE is beyond the scope of the thesis, but survey papers such as Carpineto and Romano [2012]; Bhogal et al. [2007]; Vechtomova [2009] and IR books such as Baeza-Yates et al. [1999] and Manning et al. [2008] cover the topic in detail.

Vechtomova [2009] identifies three main sources for the query expansion terms:

1. Hand built dictionaries, thesauri and ontologies, for example, WordNet.
2. Documents used in the retrieval process, for example, top K documents for query.
3. External collections, such as Wikipedia.

Below we explain some of the Query Expansion techniques which could be used for TS systems.

2.4.1 WordNet

WordNet[6][Miller, 1995] is a lexical database of english words, where words are grouped into sets of synonyms called "synsets". Each synset represents a concept, and WordNet also

[6]Available for free at http://wordnet.princeton.edu/

maintains various lexical relationships between different synsets. For example, WordNet stores hypernym/hyponym (e.g. bird is hypernym for pigeon and pigeon is hyponym for bird) relationships between noun synsets.

The following noun synsets exist for the word "hero":

1. **hero** (the principal character in a play or movie or novel or poem).
2. champion, fighter, **hero**, paladin (someone who fights for a cause).
3. bomber, grinder, **hero**, hero sandwich, hoagie, hoagy (different names of sandwiches).
4. **Hero**, Heron, Hero of Alexandria (Greek mathematician who devised a method for calculating area of a triangle).

In order to find expanded words for the query terms using WordNet, one has to first identify the correct synset for the word (depending on the context of the word in the query). After finding the best synset, one can choose the words present in synset along with the words present in any related synset as the expanded terms [Carpineto and Romano, 2012]. And finally while using the expanded words in any of the retrieval methods, appropriate weights can be assigned to the expanded words.

WordNet was used by Voorhees [1994], where after several experiments it was hypothesized to be ineffective for usage in IR, due to the loss in precision (unless the correct synset is chosen). Later some of the problems in WordNet such as lack of proper names in WordNet, and existence of polysemous words (words having multiple synsets) have been handled by Mandala et al. [1998]. WordNet was effectively used by Pasca and Harabagiu [2001] as a promising method for query expansion in the QA track of TREC.

2.4.2 Relevance Feedback and Pseudo-Relevance Feedback

The main idea behind Relevance Feedback is to get feedback from the user for document relevance, and then expand the query by selecting important terms from the identified relevant documents [Büttcher et al., 2010, page 273].

The method for Relevance Feedback involves the following steps (see Büttcher et al. [2010, page 274]):

1. Retrieve documents for the user's query.

2. User browses the documents and identifies/marks the relevant documents.

3. System ranks the words from the identified relevant documents and selects the top K words as expansion terms.

4. Execute the final query with adjusted weights (generally $\frac{1}{3}$ of original weight for expanded terms) and present the user with final list of documents.

Pseudo-Relevance Feedback (PRF) on the other hand, eliminates the user interaction step of identifying the relevant documents [Büttcher et al., 2010, page 275]. PRF assumes that top N documents retrieved for the user query are relevant and then proceeds with the steps 3 and 4 shown above. The performance of PRF highly depends on the number of relevant documents retrieved in the top N documents which are used for finding expanded terms [Büttcher et al., 2010, page 275].

2.5 Distributional Similarity

Statistical based approaches such as term-term co-occurrence measures using entire corpus were suggested by researchers for AQE [Qiu and Frei, 1993; Xu and Croft, 1996; Schütze and Pedersen, 1997; Park and Ramamohanarao, 2007]. The methods involve either corpus specific global techniques or query specific local techniques, calculating co-occurrence measures for query terms at either document level, topic level or even at sentence or paragraph level in the corpus collection [Carpineto and Romano, 2012]. Different similarity measures like Dice coefficient, Jaccard, Cosine, Average Conditional Probability, and Normalized Mutual Information could be used for selecting query expansion terms, and were evaluated by Kim and Choi [1999], where the authors show that Dice, Jaccard and Cosine perform better than the others.

One of the main disadvantages of the statistical approaches based on the co-occurrence of the terms is that they do not check for the linguistic meaning in which the terms are used. Also, for the co-occurrence based methods only the terms which occur along with the query terms within the vicinity of either document level, paragraph level or topic level are found as expansion terms. Distributional Similarity methods overcome these limitations by considering words that occur in similar contexts (even from different documents) to be related [Vechtomova, 2012].

Distributional similarity measures suggested by Lin [1998]; Weeds and Weir [2003] are some of the standard measures to calculate similarity between terms [Vechtomova, 2012].

Vechtomova and Robertson [2012] adapted BM25 ranking function for calculating similarity between context features of the words, and found it to be competitive compared to other distributional similarity measures. Below we only describe the Lin's distributional similarity metric which is used as a query expansion technique in subsequent chapters.

Given a list of seed words (related to the query), and a corpus of well formed sentences or documents which are seemed to be relevant to the query, Lin's similarity measure can identify related words for each of the seed words. Lin's distributional similarity method uses grammatical dependency relations as features, and the Mutual Information is used as the weight of the feature [Lin, 1998].

The method for calculating Lin's similarity is as follows [Lin, 1998]:

1. A standard NLP parser like Stanford NLP[7] could be used to extract dependency triples from the text corpus. A dependency triple consists of two words and a grammatical relationship between them in the sentence.

2. Let $f(w, r, w')$ denote the frequency of the dependency triple containing words w and w' with the relationship r. Wild card character is denoted by $*$, and the $f(w, r, *)$ denotes the sum of the frequency counts of all the dependency triples containing word w and relationship r.

3. Mutual Information $I(w, r, w') = \log \frac{f(w,r,w') \times f(*,r,*)}{f(w,r,*) \times f(*,r,w')}$

4. Each seed word (s) is represented by a feature vector V consisting of pairs $< r, w >$ with $I(s, r, w) > 0$.

5. The similarity measure between seed word (s) and a candidate word (c) is given as:

$$sim(s, c) = \frac{\sum_{<r,w> \in V(s) \cap V(c)} (I(s, r, w) + I(c, r, w))}{\sum_{<r,w> \in V(s)} I(s, r, w) + \sum_{<r,w> \in V(c)} I(c, r, w)}$$

2.6 Nugget-based Evaluation in the QA track

In TREC 2003, nugget based evaluation was designed to evaluate a system's response for definition questions. "There were 50 definition questions for TREC 2003, of which 30 were seeking information about a person (e.g. Vlad the Impaler, Benhur), 10 about an

[7] http://www-nlp.stanford.edu/software/lex-parser.shtml

1	vital	Agreement between companies and top executives
2	vital	Provides remuneration to executives who lose jobs
3	vital	Remuneration is usually very generous
4	okay	Encourages execs not to resist takeover beneficial to shareholders
5	okay	Incentive for execs to join companies
6	okay	Arrangement for which IRS can impose excise tax

Table 2.1: List of nuggets for "What is a golden parachute?" question in TREC 2003 QA track.

organization (e.g. Freddie Mac, Bausch & Lomb) and 10 questions about some other thing (e.g. golden parachute, TB, etc.)" [Voorhees, 2004].

As a response to definition questions, systems return an unordered set of *[document id, answer string]* pairs for every question. Every answer string is supposed to contain a facet of information about the target in the question. As such there were no limits placed on the length of each individual answer string or number of such pairs, but systems were penalized for retrieving extraneous or duplicate information [Voorhees, 2004].

The evaluation of the systems for this task is performed as follows [Voorhees, 2004]:

1. The assessor is presented with a list of answer strings obtained from all the systems' responses for every question. The assessor then builds a single list of "information nuggets" along with their importance level (*"vital"* or *"okay"*) from the list of answer strings for every question. According to Voorhees [2004], "an information nugget is defined as a fact for which the assessor could make a binary decision as to whether a response contained the nugget".

   Table 2.1 shows the list of nuggets found and classified by the assessor for the definition question "What is a golden parachute?" in 2003 QA track.

2. The assessor then manually matches the nuggets to the answer strings returned by a system. For each answer string returned by the system, the assessor checks if the answer string covers the same information as covered in the nugget. If a particular system response covered the nugget more than once, then only one of the answer strings is matched to the nugget. According to Voorhees [2004], assessors ignored the wording differences and considered only the conceptual matches between nuggets and systems' responses, and hence the manual evaluation was necessary.

   Table 2.2 shows the list of nuggets matched to a sample system's response for the "What is a golden parachute?" question in 2003 QA track.

2, 3	a.	The arrangement, which includes lucrative stock options, a hefty salary, and a "golden parachute" if Gifford is fired,
1	b.	Oh, Eaton also has a new golden parachute clause in his contract.
	c.	But some, including many of BofA's top executives, joined the 216 and cashed in their "golden parachute" severance packages.
	d.	The big payment that Eyler received in January was intended as a "golden parachute"
	e.	Cotsakos' contract included a golden parachute big enough to make a future sale of the company more likely
	f.	syndication, the golden parachute for production companies
6	g.	But if he quits or is dismissed during the two years after the merger, he will be paid $24.4 million, with DaimlerChrysler paying the "golden parachute" tax for him and the taxes on the compensation paid to cover the tax.
	h.	If he left, On leaving, O'Neill could would be able to collect a golden parachute package providing three years of salary and bonuses, stock and other benefits.
4	i.	After the takeover, as jobs disappeared and BofA's stock tumbled, many saw him as a bumbler who had sold out his bank, walking away with a golden parachute that gives him $5 million a year for the rest of his life.
	j.	And after BofA disclosed that he had a golden parachute agreement giving him some $50 million to $100 million if he left following the merger, he sent a voice mail message to bank employees that he intended to stay.

Table 2.2: List of nugget matches created by assessor for a system response to "What is a golden parachute?" (as shown in Voorhees [2004]).

Let,

n = Number of vital nuggets returned in a response
a = Number of okay nuggets returned in a response
N = Total number of vital nuggets in the assessors list
l = Number of non-whitespace characters in the entire answer string summed over all answer strings in the response;

Then,

$$\text{Recall of the system } (R) = \frac{n}{N}$$

$$\text{Allowance } (\alpha) = 100 * (n + a)$$

$$\text{Precision } (P) = \begin{cases} 1, & \text{if } l < \alpha \\ 1 - \frac{l-\alpha}{l}, & \text{otherwise} \end{cases}$$

$$\text{F-score : } F(\beta) = \frac{(1+\beta^2) * P * R}{\beta^2 * P + R}$$

$$\text{For TREC 2003 : } \beta = 5,$$
$$\text{For TREC 2004, 2005 : } \beta = 3$$

Figure 2.1: Evaluation metric for a system's response for definition questions.

3. Recall of a system is defined to be the ratio of the number of vital nuggets discovered in system's response to the total number of vital nuggets found by the assessor.

   According to Voorhees [2004], researchers found evaluation of the precision of the system to be tricky because of the inconsistency while calculating the total number of nuggets (*vital* & *okay*) discovered in each system's response by the assessors. So for the evaluation of precision, verbose system's response is penalized just as in the evaluation of summarization systems [Harman and Over, 2002].

   F-score $F(\beta)$ is calculated for each system's response as described in Figure 2.1. For the 2003 QA track, the value of $\beta = 5$ and for 2004, 2005 tracks the value of $\beta = 3$ was used [Voorhees, 2004; 2005; 2006]. This shows that the track organizers felt the recall of a system was three to five times more important than the precision for the QA track.

2.7 Automatic nugget-based evaluation systems: POURPRE, Nuggeteer

In the evaluation method described in the previous section, even though the answer key (list of nuggets) needs to be created only once for every question, assessors need to manually match these nuggets to the answer strings returned by every system. This human involvement in judging a system's response is a bottleneck for the evaluation process. In future, if there are more systems to evaluate the current manual evaluation is not easily scalable and also the current evaluation which requires assessor's judgements cannot be directly used on a new system. Lin and Demner-Fushman [2005] proposed an automatic evaluation system, POURPRE, which can automatically match nuggets to answer strings without human involvement. It was found that the POURPRE's automatic scoring metric correlates well with the QA track's evaluation.

2.7.1 POURPRE

Papineni et al. [2002] first successfully implemented the BLEU metric (also known as IBM BLEU), which automatically evaluates the machine translation output of a system. BLEU uses an n-gram co-occurrence statistics to compare user translated reference outputs with the system's output. The system's output is then scored upon the number of matches it has with the reference outputs. It was found that the BLEU's metric of a system correlates highly with the human judgement of the system for the machine translation. The same idea was extended to evaluate the document summarization output by ROUGE, a system developed by Lin and Hovy [2003].

POURPRE works on the same ideas as BLEU or ROUGE. However, in POURPRE instead of n-gram matches, the authors use unigram matches [Lin and Demner-Fushman, 2005]. The authors hypothesize that using n-grams ($n \geq 2$), will not be directly useful because n-gram matches essentially check for the fluency of the sentences which is necessary in machine translation, whereas in matching concepts (i.e. nuggets to answer strings) it is not much useful.

So essentially in POURPRE, for matching a nugget to an answer string, the system checks for the percentage of words in the nugget that overlap with the sentence, also called as the match score. The nugget is matched to the answer string in the output to which it has the highest match score, and the other answer strings are assumed to not contain the nugget. Once the match scores for all the nuggets have been found, the recall of the system is defined to be the sum of the match scores of all vital nuggets divided by the

RUN	Percentage match score	IDF match score
TREC 2004 ($\beta = 3$)	0.833	0.812
TREC 2003 ($\beta = 3$)	0.886	0.876
TREC 2003 ($\beta = 5$)	0.878	0.875

Table 2.3: Kendall's τ correlation of POURPRE with official track rankings [Lin and Demner-Fushman, 2005].

total number of vital nuggets in assessors list. Precision was also calculated in the similar manner using the same formula used by manual evaluation in Figure 2.1.

POURPRE also uses a modified version of percentage match of nugget to sentence, by taking into account the Inverse Document Frequency (IDF) scores. This is a meaningful modification because matching common words like "in", "the", "year", is not same as matching not so common words like "parachute" (in the nugget example). So, the modified match score is the sum of the IDF scores of the matched words in the nugget divided by the total IDF of all the words in the nugget.

Lin and Demner-Fushman [2005] have shown that POURPRE method of automatically matching nuggets to answer string correlates well with the human judgements and also at the same time better than directly using BLEU or ROUGE system for automatic evaluation. The correlation results of POURPRE on the QA tracks of TREC is outlined in Table 2.3.

2.7.2 Nuggeteer

Nuggeteer is another automatic evaluation system proposed by Marton and Radul [2006] for nugget based evaluation methods. Nuggeteer is build upon the premise that "If a system response was ever judged by a human assessor to contain a particular nugget, then other identical responses also contain that nugget" [Marton and Radul, 2006]. Marton and Radul [2006] build a binary classifier for every nugget, using n-gram weight, informativeness measure for each n-gram and a matching threshold between a response and a nugget as shown below.

RUN	POURPRE	Nuggeteer
TREC 2005 (O) ($\beta = 3$)	0.709	0.858
TREC 2004 (O) ($\beta = 3$)	0.833	0.898
TREC 2003 (D) ($\beta = 3$)	0.886	0.879
TREC 2003 (D) ($\beta = 5$)	0.878	0.849

Table 2.4: Kendall's τ correlation of POURPRE and Nuggeteer with official track rankings, D represents definition task and O represents other questions task.

According to Marton and Radul [2006],

for each n-gram $(w_{i+1}w_{i+2}...w_{i+n}) \in$ response $(S = w_1w_2...w_l)$,

$$\text{n-gram weight } (W) = \sum_{k=1}^{k=n} IDF(w_{i+k})$$

$$i(g, w_{i+1}w_{i+2}...w_{i+n}) = \begin{cases} 1, & \text{if } count(g, w_{i+1}w_{i+2}...w_{i+n}) > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\text{Informativeness } (I) = 1 - \frac{\sum_{g' \in G} i(g', w_{i+1}w_{i+2}...w_{i+n})}{|G|}$$

$$\text{Recall } R(g, S) = \sum_{k=0}^{n-1} \sum_{i=0}^{l-k} W(g, w_i...w_{i+k}) * I(g, w_i...w_{i+k})$$

where, $count(..)$ function tells the number of times n-gram appears in a system response containing nugget g.

The thresholds for deciding whether a nugget g is contained in system response S, is based on the bayesian model generated on the training data for all the nuggets [Marton and Radul, 2006]. The Kendall's τ correlation results shown in Table 2.4 were achieved for the TREC QA tracks conducted in various years.

Overall, Nuggeteer shows comparable correlation values when compared to POURPRE for the automatic evaluation of Definition and Relationship question types. Nuggeteer's main advantage is that it uses already judged response instead of the nugget for better comparison while judging a new response.

Towards the end of the thesis we propose a similar supervised learning based evaluation mechanism for the TS track which can considerably reduce the manual effort and assessor's time, making the evaluation scalable and easily applicable to judge new systems.

Chapter 3

TREC 2013: Temporal Summarization Track

This chapter explains in detail our approach used to build a TS system for TREC 2013. At the end of this chapter, we discuss the performance of our system by comparing it to other systems in the track, and discuss the shortcomings and scope for improvements in our approach. We submitted four runs to the track, out of a total of 26 runs submitted. University of Waterloo submitted a total of eight runs to the track, which includes our four runs. The problem statement of TS track of TREC 2013 is discussed earlier in Section 1.2 of Chapter 1. The evaluation metrics were also discussed in the same chapter earlier.

Three students from University of Waterloo (myself, Gaurav Baruah and Adam Roegiest) participated in the TS track submitting individual runs. Since all of us were working on the same problem, we worked together to build a common base framework by downloading the documents, indexing the content and scoring the documents with respect to the test queries, which could then be used to build the runs with individual methods. The experimental setup detailed in Sections 3.1 and 3.2, is a joint work which is also described in [Baruah et al., 2014].

3.1 Preliminaries and Experimental Setup

The TS track of TREC 2013 uses KBA-2013 stream corpus[1] as the time-ordered document collection, which was downloaded locally into our system. The data inside the corpus

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

is serialized with thrift format, so the C++ thrift sample from https://github.com/trec-kba/streamcorpus was used to extract the content from thrift format [Baruah et al., 2014].

The Named Entity (NE) tags were ignored while converting the document to TREC format, as it was found that some of the NE tags like place, person, etc. were wrongly tagged. The main reason for the conversion to TREC style format is for the ease of processing the documents in further stages [Baruah et al., 2014].

3.2 Indexing & Scoring Documents

We did not use search engine frameworks like Lucene, Indri, or Wumpus for indexing the document collection, even though they incorporate many standard Information Retrieval techniques for retrieving and scoring the documents for a given query.

The main reasons for not using the already available search frameworks, like Wumpus, are [Baruah et al., 2014]:

1. The document collection stream used for this track is temporal in nature, i.e., the documents should be indexed according to time in the order they appear (i.e., increasing order of timestamps), and given the query start and end time stamps, only documents from these timestamps should be retrieved. Hence, to use Wumpus, significant amount of changes need to be done in the indexing, and also to process the results for removing the documents which are not in the query time period.

2. We noticed duplicate document identifiers in the corpus, and it would require some preprocessing steps (to remove the documents), before the Wumpus could index the collection.

In light of the above, and the submission deadline time constraints, we felt it was better to index the document collection and score the documents using our own method.

3.2.1 Indexing: Hour-wise index files

The document collection is time ordered, and for each query, we need to use only the documents that appear before the query start time for statistics like TF-IDF. So, building

one standard inverted index file for the whole document collection is not appropriate for our requirement. We could have built different index files for each query, using only the documents that appear before that query's start time, but for a new query, indexing should be done again, which doesn't make the system scalable as the documents need to be processed again.

So, it was intuitive that indexing the documents hour-wise would be the most appropriate method, as there are a total of 11,948 hours of documents in the collection, and even for a query whose start time is around 10,000th hour, only 10000 index files need to be merged to get collection statistics, which is easier compared to going through the entire document collection for 10,000 hours to build index.

For every hour in the document collection (ranging from 05-Oct-2011 00 hrs, to 13-Feb-2013 23 hrs), we built the following files for documents within every hour [Baruah et al., 2014]:

1. *hourTF.gz*: Contains <Word : Term-Frequency> tuples for all the terms appearing in the documents in that hour. Number of terms in the collection (l_C) up until that hour and the number of terms in that hour (l_H), are also stored in the file.

2. *features.gz*: Contains Term-Frequency feature vectors of documents in one file, with each line containing features of one document (docId: list of <word : freq_count>).

3. *meta.gz*: Metadata in a file with each line containing doc-id, source `.gz` file of the document, document timestamp and document length.

With these files, the whole document collection was reduced to approximately 600 GB (compressed), and as mentioned earlier, this helps us to compute the term statistics up to the query start time easily [Baruah et al., 2014]. To tokenize the documents into terms, we used whitespace tokenizer with only alphabetic characters retained.

3.2.2 Scoring documents: Using Query Likelihood model

Language Model with Dirichlet smoothing (LMD) was used to score the documents with respect to the query (described in Equation 2.12 earlier), as shown below:

$$score_{LMD} = \sum_{t \in q} q_t \cdot log\left(1 + \frac{f_{t,d}}{\mu} \cdot \frac{l_C}{l_t}\right) - n \cdot log\left(1 + \frac{l_d}{\mu}\right) \tag{3.1}$$

where, t is the term, q is the query term vector, q_t is the term frequency of t in the query, $f_{t,d}$ is the term frequency of t in document d, l_C is the length of the collection (the total number of terms in the collection), l_t is the number of times the term t appears in the collection, and μ is the Dirichlet smoothing factor. We set the value of $\mu = 1000$ [Baruah et al., 2014].

All the documents within the query duration (between start time and end time of the query) were scored using the Equation 3.1, the feature vector file (*features.gz*) which has the frequency counts of all the terms appearing in each document, and the *hourTF.gz* file which has the collection length l_C. The cumulative frequency l_t for all the query terms is calculated using the *hourTF.gz* files from the document collection start until the current hour.

The following algorithm was used to compute document scores, i.e. $score_{LMD}$ (as shown in [Baruah et al., 2014, page 3]):

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

The scores for all the documents within a particular hour are written to a separate file, *query.score*, which will be used in later stages. After the documents are scored, only sentences from the top scoring documents are considered for further processing (as they are more likely to be relevant).

We used an arbitrary fixed cutoff score of 0 for document selection, i.e., all documents whose $score_{LMD} > 0$ were extracted from the corpus for further processing (denoted by document set D) [Baruah et al., 2014]. However, later in Section 3.4, we propose an adaptive cutoff score algorithm, which chooses the document cutoff score for hour $h + 1$ dynamically based on the scores of the documents in hour h.

Note that, a cutoff score is used instead of using top K documents because: As per the track regulations, at a particular time instance we cannot use the future documents within that hour. If at all top K documents in the hour need to be found, the decision timestamps for the sentences in the run should be at least the end of the hour, which would in turn increase the latency of the update and the penalty on gain metric.

With this basic framework built until now, we used separate algorithms to process the document set D and prepare our runs.

3.3 Query Expansion: Distributional Similarity

From the training topic, it was observed that the query is very short in length (e.g. "iran earthquake"), and we already know that the query size used in major search engines like Google is less than 3 words[2]. After initial processing of the documents within the first few hours of the collection, we observed that the average sentence length is ≈ 34 words. Since the query is very small in length and also the sentence length is very small, intuitively it seems that chances of finding query terms in the sentence will be low. Also, it was found from the list of nuggets released for the training topic that 78 relevant nuggets (out of a total of 103 nuggets) did not contain any of the query terms ("iran" or "earthquake").

So, to improve the recall (coverage of relevant nuggets) of our system we use query expansion techniques (expanding the query with relevant words), and score the sentences with respect to the expanded query. After expanding the query with related terms, we used the following formula (adapted from Equation 2.4) to score the sentences.

$$score_{sentence} = \sum_{t \in q} \left(q_t \times \frac{f_{t,s}(k_1 + 1)}{k_1((1-b) + b(l_s/l_{avg})) + f_{t,s}} \times w_t \right) \tag{3.2}$$

where, $f_{t,s}$ is the frequency of term t in sentence s, l_s is length of sentence, l_{avg} is the average sentence length (found to be 34 earlier) and parameters $k_1 = 1.2, b = 0.75$ (as prescribed by [Büttcher et al., 2010, page 272]).

The above equation is a direct adaptation of Okapi-BM25 function, where the sentence is now treated as a document. So, the weight of the term w_t should now be the Inverse Sentence Frequency (ISF) weight i.e. $log(\frac{N}{N_t})$, where N = Total number of sentences, and N_t =number of sentences containing term t. Blake [2006] shows an interesting observation

[2] http://www.hitwise.com/us/about-us/press-center/press-releases/2009/google-searches-jun-09/

that there is very high correlation between the ISF and the Inverse Term Frequency (ITF), i.e. $log(\frac{l_C}{l_t})$, where l_C is length of the collection and l_t is the number of times t appears in collection. Momtazi et al. [2010] also confirms that both ISF and ITF could be used as effective query term weights for the sentence retrieval. So, we used $w_t = log(\frac{l_C}{l_t})$ as an approximation for the ISF weight of the term while scoring sentences.

Kindly note that we did not weight the expansion terms with respect to the query terms, unlike general query expansion techniques described in Chapter 2. This is because the sentences to be ranked are already selected from the top scoring documents which are likely to be relevant to the query topic, and hence the expansion terms if found in the sentences of these documents must be used in the context relevant to the query. Under this assumption, we use the Equation 3.2 to score the sentence, without any additional weight for the expansion terms with respect to the query terms.

3.3.1 Finding Expansion terms

As seen earlier in Section 1.2 of Chapter 1, a query can belong to one of the following event types: {accident, bombing, earthquake, shooting, storm}. The following steps were performed, to find the expansion terms.

Step 1: For each of the event types, seed words ($\approx$ 30 words per event type) were found manually from Wikipedia articles of each event type. In particular, we ensured that the Wikipedia article for an event type occurred prior to the all the events specified in the test queries. The seeds words picked from the articles were not specific to that particular event but tend to occur more commonly in all articles of that event type.

For example, for the event type of earthquake, we used the Wikipedia articles on major earthquakes[3] before 2012 to find the following seed words:
{earthquake, damaged, leveled, killed, injured, dead, died, wounded, survive, assistance, cities, displaced, aftershock, aftermath, seismic, magnitude, quake, death, toll, recovery, victims, homeless, destroyed, ambulance, disaster, shelter, panic, stranded}.

Step 2: A list of training topics was also created, one for each event type. It was ensured that the training topics appeared before the track's test topics in time, for each of the event types. This is in accordance with the track's guidelines, which allows the use of documents from earlier time but not from the future.

For example, the training topic "iran earthquake" was used for earthquake event type, with query end time as 1345551797 unix timestamp, i.e. Aug 21, 2012. The test queries for

[3] http://en.wikipedia.org/wiki/List_of_21st-century_earthquakes

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

Table 3.1: Examples of seed and expansion terms for earthquake event type.

this event type included "guatemala earthquake", whose query start time is 1352306147, i.e. Nov 7, 2012, which is later than the training event.

Step 3: Top K (= 10,000) sentences were then extracted from the document set D retrieved for each training topic (e.g. "iran earthquake" for earthquake event type). The sentences were scored using the BM25 function in Equation 3.2 using the seed words list as the query terms.

Step 4: The top K sentences retrieved in the above step, and the seed words, are given as input to the Lin's distributional similarity algorithm (refer Section 2.4 of Chapter 2) to find the expansion terms.

Table 3.1 shows examples of seed words and their related words found for the training topic "iran earthquake". One can observe from Table 3.1 that the distributional similarity algorithm performs quite well in retrieving expansion words related to the seed word of the query topic.

These expansion words along with the initial seed words and query terms, constitute the expanded query (q in Equation 3.2) for calculating the sentence score ($score_{sentence}$) in Equation 3.2. For the TREC submission, the expansion terms were carefully hand-picked, which was possible because the number of event types (= 5) and expansion terms for each event type was small. For example, in Table 3.1, words such as "brought", "left", "parts", "work", etc. were not used in the final expanded query.

Clearly, the manual selection of words in this method is a hinderance for the scalability

of the system to new event types. However, for any new query belonging to one of the five event types the same expansion terms list can be used, since the expansion terms found are related to the event type and are not specific to the query. We also present an automatic expansion method without manual selection, developed after TREC, which is explained in detail in Chapter 4.

3.4 Sentence Selection Criteria & De-duplication

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

In order to select sentences, we could return top K (for some fixed value of K) sentences every hour. But since the decision for selection of a sentence is made at the end of the hour here, there would be a latency penalty that would be applied due to the delay in update (which might have appeared early in the hour).

And also a rigid cutoff score, such as 0 used for $score_{LMD}$ of documents, could not be used for $score_{sentence}$ to shortlist sentences, due to the diversity in the documents returned every hour (sentences might have lower scores in earlier hours, but are more important due to the low latency factor). Hence, an incremental cutoff score ($score_{cutoff}$) different for every hour was used to decide whether a sentence should be included in the list of updates or not.

The following algorithm was used to select the sentence/update into the list of updates:

```
S_h = 5000; //max num of sentences per hour (parameter for algorithm)
D_h = 3000; //max num of documents per hour (parameter for algorithm)
score_cutoff = 0; //cutoff = 0 for first hour
dscore_cutoff = DBL_MIN;
updatelist[]  //list of updates

for every hour 'h' between the start and end timestamps:
  i = 0;
  score[];  //list of sentence scores for current hour
  dscore[]; //list of doc scores for current hour

  for every document 'd' in hour 'h':
    if doc_score(d) < dscore_cutoff:
```

```
      continue;

    dscore[].add(doc_score(d));

    for every sentence 's' in document 'd':
      if score_sentence(s) > score_cutoff:
        add 's' to updatelist[];
        score[].add(score_sentence(s));
      end if
    end for

  end for

  if score[].length() > S_h: //update score_cutoff
    sort score[] desc
    score_cutoff = score[S_h];
  end if

  if dscore[].length() > D_h: //update dscore_cutoff
    sort dscore[] desc
    dscore_cutoff = dscore[D_h];
  end if
end for

return updatelist[]
```

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

The two parameters, S_h and D_h, in the above mentioned algorithm select the number of sentences returned in the update. We submitted four different runs for different parameter values, which is explained in detail in the next section.

The deduplication step kicks in while adding a sentence to the update list. If more than 90% of the words in the current sentence are covered in any of the previously re-

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

Table 3.2: S_h and D_h for *rg* runs

turned sentences, it was discarded as a duplicate. For the TREC submission, we followed the mentioned deduplication algorithm, but after TREC, we compared different deduplication techniques which could be used in TS systems. Experiments conducted after TREC conference which investigate various techniques are detailed in Chapter 4.

Also, for all the four runs submitted to TREC, the confidence score for the update returned in our run was defined to be $score_{sentence} * score_{document}$ [Baruah et al., 2014]. This was particularly done to ensure that the high scoring sentences from high scoring documents receive greater confidence score.

3.5 Submitted runs

We submitted four different runs, namely: **rg1, rg2, rg3,** and **rg4**, to the TS track of TREC. Other runs from the University of Waterloo are: CosineEgrep, NormEgrep, UWMDSqlec2t25 and UWMDSqlec4t50.

The four **rg** runs differed in the number of sentences and documents selected per hour. Table 3.2 lists the number of sentences and documents shortlisted for each *rg* run. The incremental minimum cut-off scores change every hour based on the score of the S_h^{th} sentence of the previous hour, as shown in the algorithm in the previous section.

The reason for submitting different runs is to test the right cutoff for the number of sentences selected per hour. The track's evaluation metrics, Expected Gain and Comprehensiveness, are similar to standard IR metrics, Precision and Recall respectively. While Expected Gain (EG) tells us about the gain received by the system for covering relevant nuggets in all the updates returned, Comprehensiveness (C) metric indirectly measures the number of relevant nuggets covered out of all the relevant nuggets.

The formulae for the metrics as explained in Section 1.4 of Chapter 1 are:

$$\mathbf{EG} = \frac{1}{|S|} \sum_{n \in N : M(n,S) \neq \phi} g(M(n,S), n) \tag{3.3}$$

$$\mathbf{C} = \frac{1}{\sum_{n \in N} R(n)} \sum_{u \in S} \sum_{n \in M^{-1}(u,S)} g(u,n) \tag{3.4}$$

Hence, intuitively one can expect that, as the number of updates increases, there is a possibility that the system covers new nuggets and hence Comprehensiveness (C) metric (i.e. Recall) might increase. But at the same time, with an increase in the number of updates, the Expected Gain of the system might decrease due to the faster growth of $|S|$ in denominator of Equation 3.3. So, to achieve the right balance between the two metrics, the parameter values for S_h and D_h need to be varied, and hence the four runs.

While testing on the training topic, it was observed that the sentence scores are not accounting for the verbosity of sentences, even with the BM25 score (after changing b parameter). Verbosity of sentences is important because, ELG (Expected Latency Gain) metric assigns penalty for verbose sentences. So we prepared two runs by multiplying the sentence score ($score_{sentence}$ in Eq 3.2) with $log(1 + \frac{l_{avg}}{l_d})$; (see document length normalization in [Büttcher et al., 2010, page 301]). This was tried for the TREC submission as an experimental change to Okapi-BM25 scoring function for TST. Even though runs *rg1* and *rg2* have the same parameters, the scoring formulae for both differ. Runs *rg3* and *rg4* differ in the same way.

So, while runs **rg1** and **rg3** use Eq 3.2 for scoring the sentence, for runs **rg2** and **rg4**, the scoring formula is:

$$score_{sentence} = \sum_{t \in q} \left(q_t \times \frac{f_{t,d}(k_1 + 1)}{k_1((1-b) + b(l_d/l_{avg})) + f_{t,d}} \times w_t \right) \times log(1 + \frac{l_{avg}}{l_d}) \tag{3.5}$$

3.6 Results & Discussion

Table 3.3 shows the results of various runs submitted to the Temporal Summarization task. The track's metrics, Expected Latency Gain and Latency Comprehensiveness, are reported for different runs.

Our run (**rg1**) achieved the highest value (0.571) in Latency Comprehensiveness (LC) metric in the competition. Also, this value is nearly twice the average value of the Latency Comprehensiveness achieved by all systems. Similarly, our other runs, namely rg2, rg3 and rg4 achieved very high values in the LC metric. The main reasons for this high value of

RunID	ELG	LC
PRIS-cluster5	**0.136 (0.090)**	0.126 (0.164)
ICTNET-run2	0.127 (0.075)	0.251 (0.169)
ICTNET-run1	0.125 (0.075)	0.253 (0.169)
hltcoe-TuneExternal2*	0.117 (0.073)	0.203 (0.156)
hltcoe-TuneBasePred2*	0.114 (0.117)	0.244 (0.188)
PRIS-cluster3	0.103 (0.050)	0.176 (0.203)
PRIS-cluster2	0.074 (0.031)	0.260 (0.217)
uogTr-uogTrNMTm1MM3	0.069 (0.053)	0.216 (0.203)
PRIS-cluster4	0.067 (0.026)	0.288 (0.262)
PRIS-cluster1	0.067 (0.026)	0.292 (0.270)
hltcoe-BasePred	0.067 (0.057)	0.368 (0.272)
hltcoe-Baseline	0.063 (0.046)	0.381 (0.261)
uogTr-uogTrNSQ1	0.060 (0.031)	0.184 (0.171)
ALL-	0.058 (0.061)	0.288 (0.288)
hltcoe-EXTERNAL	0.054 (0.027)	0.413 (0.291)
uogTr-uogTrNMTm3FMM4	0.049 (0.028)	0.170 (0.143)
uogTr-uogTrNMM	0.045 (0.023)	0.254 (0.231)
uogTr-uogTrEMMQ2	0.040 (0.024)	0.259 (0.254)
wim GY 2013-SUS1	0.036 (0.029)	0.148 (0.149)
UWaterlooMDS-rg4	0.028 (0.019)	0.516 (0.339)
UWaterlooMDS-rg3	0.026 (0.015)	0.506 (0.323)
UWaterlooMDS-rg2	0.022 (0.018)	0.562 (0.349)
UWaterlooMDS-rg1	0.021 (0.016)	**0.571 (0.358)**
UWaterlooMDS-UWMDSqlec4t50	0.018 (0.016)	0.530 (0.325)
UWaterlooMDS-UWMDSqlec2t25	0.017 (0.016)	0.537 (0.322)
UWaterlooMDS-CosineEgrep	0.010 (0.015)	0.018 (0.027)
UWaterlooMDS-NormEgrep	0.001 (0.002)	0.061 (0.117)
BJUT-Q0*	0 (0.0)	0 (0.0)

Table 3.3: μ and σ (in parenthesis) of task metrics namely Expected Latency Gain (ELG) and Latency Comprehensiveness (LC) over all queries, sorted by Expected Latency Gain. *run not pooled. Table is as reported in [Aslam et al., 2014, page 13].

LC metric and low ELG are due to the large number of updates returned in our runs and due to the query expansion.

The following observations can be made from the results of the track:

1. The track's metrics, Expected Latency Gain and Latency Comprehensiveness, are nearly inversly related just like Precision and Recall. We can observe from the table that, with decrease in ELG metric, the LC of the run increases. Since both metrics are important for a system, a good system should maintain a right balance between these two values. However, in the track, there is no single metric (combining these two metrics) with which the systems could be ranked.

2. With decrease in the number of updates returned by the system per hour, we observe that the ELG metric increases and LC metric decreases. Intuitively this is understandable, because of the greater decrease in $|S|$ in the denominator (in Equation 3.3) compared to the decrease in gain in the numerator while calculating EG metric. Similarly, with a larger number of updates returned per hour, the chances of covering new nuggets increase, and hence the gain of the system increases but the denominator remains the same for Comprehensiveness metric (in Equation 3.4), so overall there is an increase in LC value with the increasing number of updates.

   For example, runs rg1 and rg3 use the same methodology for ranking sentences. However, they differ only in the cutoff (S_h) value for selecting the number of sentences per hour. Run rg1 returns a maximum of 5000 sentences per hour, whereas rg3 returns a maximum of 3000 sentences per hour. From the table, ELG(rg1) $<$ ELG (rg3) and LC(rg1) $>$ LC(rg3). Similar observations can be made for runs rg2 and rg4.

3. We see that the sentence length normalization, i.e. multiplying the sentence score by $log(1 + \frac{l_{avg}}{l_d})$ as explained in Equation 3.5, shows very marginal improvement or nearly indifferent in both the metrics (which include penalty for verbose sentences).

Based on the above observations, we attribute the low scores in Expected Latency Gain metric of our system to the following factors:

1. Large number of sentence updates (around 3000 or 5000) returned per hour.

2. Weak deduplication algorithm, which checks for duplicate sentences based on the percentage of words covered in an earlier update, with cutoff of 90% of words.

In Chapter 4, we try to overcome the above problems with better deduplication algorithms and by tuning parameters for the sentence selection cutoffs. We also make the system more robust and scalable, by proposing an automatic method for expanding the terms, in lieu of the current manual selection of seed and expansion terms.

Chapter 4

Experiments after TREC

This chapter investigates various techniques that could be applied while building a Temporal Summarization system. In particular, this chapter investigates the effectiveness of adaptive sentence cutoff selection algorithm, stemming, various deduplication algorithms and finally query expansion using Lin's distributional similarity [Lin, 1998]. We study the effectiveness of these techniques so that researchers can use these methodologies towards building an effective Temporal Summarization system for TREC in future years. Also, in this chapter, we propose an automatic query expansion method using Lin's distributional similarity where seed words and expanded words are identified automatically, unlike the hand-crafted manual selection done in Chapter 3.

4.1 Experimental Setup and Baseline runs

We use the same experimental setup as in the previous chapter which was built while participating in TREC 2013. In Section 3.1 and 3.2 of Chapter 3, we explained the process of experimental setup which involves downloading KBA corpus, indexing the collection and extracting documents for the TREC queries using Language Modeling approach.

Once the documents have been extracted for further processing of sentences, we carried out the following experiments for studying the effectiveness of various methods. The baseline run for each of the experiments varies but predominantly the baseline run is built by ranking and selecting sentences with respect to the original query without query expansion.

4.2 Effectiveness of adaptive cutoff based sentence selection

Sentence selection is one of the important steps while building a Temporal Summarization system. A sentence should be returned in the list of updates based on the novelty, importance, and especially without any latency from the time when it appeared in the news/document stream. In Section 3.4 of Chapter 3, we introduced a sentence selection algorithm with an adaptive cutoff for the sentence scores.

To study the effectiveness of the adaptive cutoff based sentence selection criteria, we consider a baseline algorithm which returns the top ranked sentences every hour (during query time period) similar to the ranked document lists (top ranked documents) presented in a web search engine.

In the adaptive cutoff based algorithm presented in Section 3.4 earlier, if $S > S_h$ (parameter for the algorithm) sentences are returned for hour h, then the sentence score of the S_h^{th} sentence would be used as a cutoff (lower bound) for the sentences in hour $h+1$, i.e., only sentences which are above the cutoff score would be returned in hour $h+1$ and so on.

In the baseline run, top K (parameter for the algorithm) sentences are returned at the end of every hour during the query time period. Since the decision is taken at the end of the hour, the decision timestamp for every sentence is the end of the hour.

Adaptive sentence selection has the following advantages over baseline:

1. The decision of the sentence selection is made at the time the sentence is seen (or scored), without waiting till the end of the hour for selecting the top sentences in the hour. Thus, this method avoids the latency penalty for the sentence returned.

2. Only sentences which have a good score (greater than the determined cutoff) are returned every hour, unlike the output of the top K sentences every hour in the baseline run.

3. The adaptive cutoff score is a monotonically increasing function over the query time period, so sentences returned in the later time periods of the query have higher cutoffs compared to the sentences returned in the previous hours. This is particularly useful in TST, because the relevant information in the documents is more abundant as the time progresses and gain associated might be lower due to the latency with which

information is presented to the user. So, our sentence selection algorithm inherently prefers sentences returned in the early hours compared to return of similarly scored sentences in the later hours.

4.2.1 Results and Discussion

The experiments involved preparing runs for both the adaptive cutoff based selection algorithm and the baseline (top K sentences per hour) algorithm. While preparing the runs, it was ensured that the rest of the steps are the same for both algorithms, i.e., sentence scoring using BM25 function, no query expansion, and no deduplication was performed for both the approaches.

Multiple runs were prepared for the adaptive cutoff based algorithm by varying the parameter S_h (sentences per hour) in the algorithm mentioned in Section 3.4. The values for parameter S_h were picked in the range from 1 to 1000. D_h (documents per hour) was selected to be 1000. For every combination of $< S_h, D_h >$, there is one run created which is a point on the graph with a particular EG, ELG, C, LC, and Avg. Number of Updates per query.

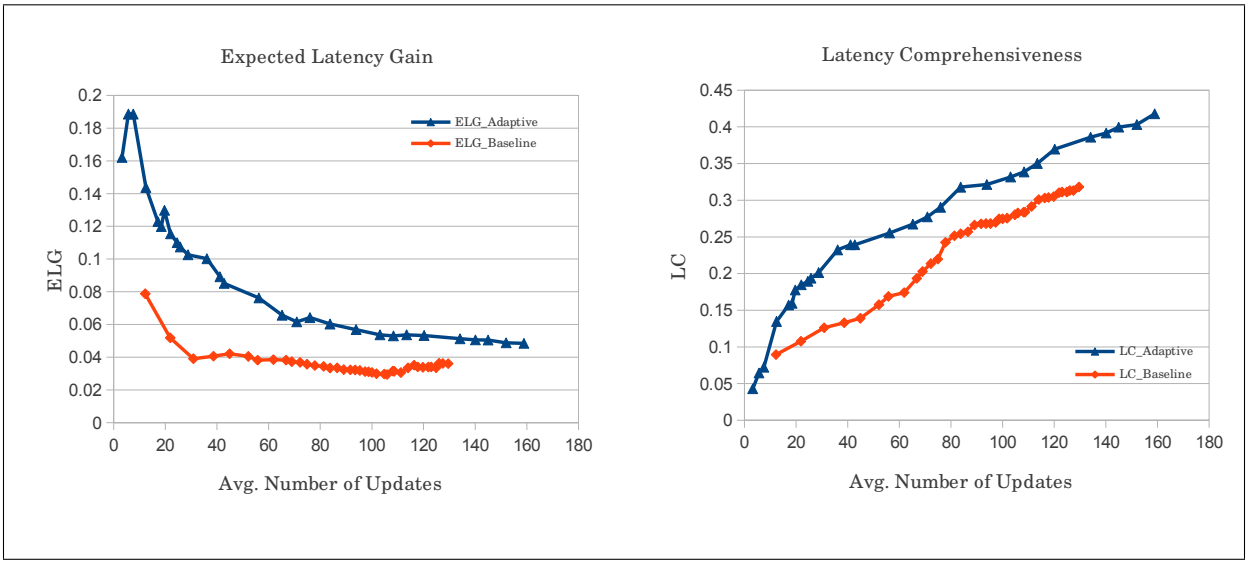

Figure 4.1: ELG and LC metrics for Adaptive cutoff and Baseline algorithms against the average number of updates evaluated per query.

Whereas for the baseline run, which returns top K sentences every hour within the

query time period, the value of K was increased from 1 to 40 in increments of 1. For each value of K, there is one run created, and a point is plotted on the graph with the metric values.

Figure 4.1, shows the ELG and LC metric values for both algorithms, i.e., adaptive cutoff based sentence selection and the baseline. The X-axis for both graphs shows the average number of updates present in the run which were evaluated. We can observe from the figure that adaptive cutoff selection algorithm outperforms the baseline algorithm in both track metrics. With the increase in updates in the run, the ELG of the run decreases but LC of the run increases. For the same number of updates evaluated in the runs, the adapative cutoff algorithm performs better than the baseline.

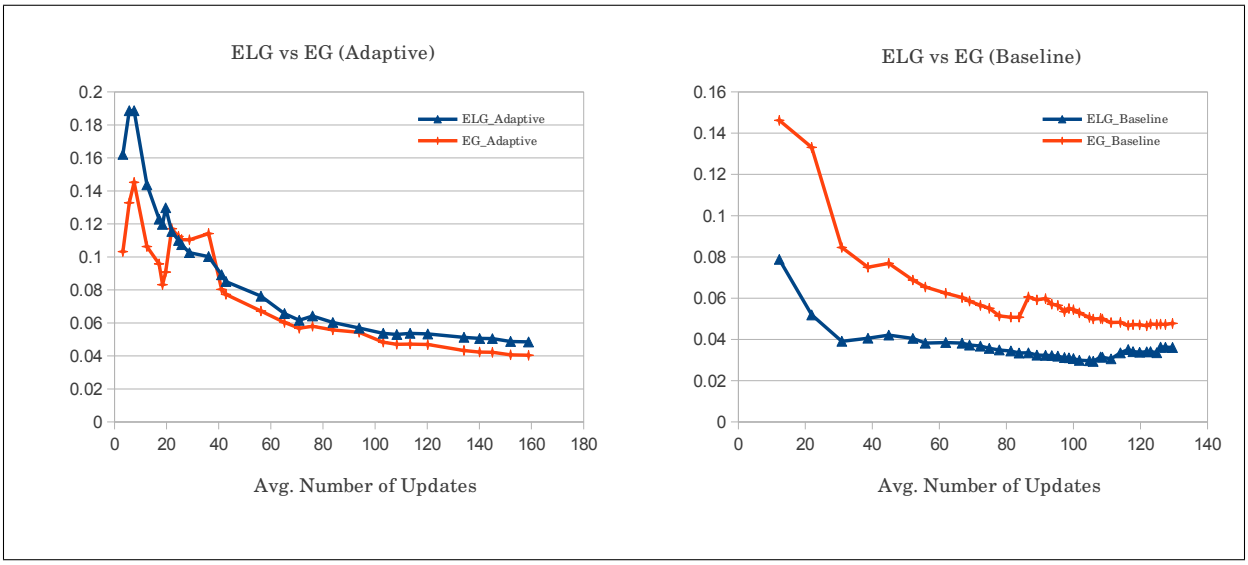

Figure 4.2: ELG vs EG for Adaptive cutoff and Baseline algorithms against the average number of updates evaluated per query.

Figure 4.2 compares the ELG and EG metrics for the adaptive cutoff and baseline algorithms. We can see that for the baseline algorithm EG of the run is always greater than the ELG metric, which means that the sentences were penalized due to the latency factor. However for the adaptive cutoff algorithm, the latency penalization is not seen and the ELG of the system is almost near or slightly higher than the EG of the run.

Figure 4.3, shows the ELG and LC metric values for both algorithms against the total number of updates returned by the TS system. Clearly from the figure, with increase in the number of sentences outputted the ELG decreases and the LC increases. It should be

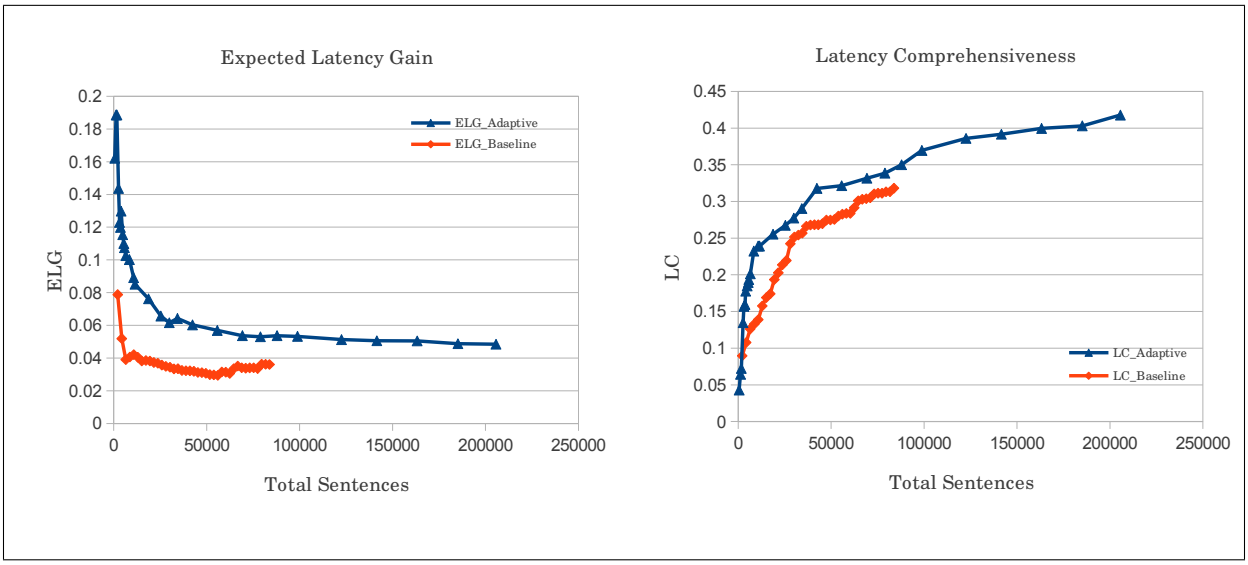

Figure 4.3: ELG and LC metrics for Adaptive cutoff and Baseline algorithms against the total number of sentences returned by the system.

noted that, for the same number of sentences returned by the TS systems, adaptive cutoff selection algorithm outperforms the baseline.

Appendix A shows the list of runs for both algorithms, for which the graphs have been plotted.

Thus, we propose that the adaptive cutoff selection algorithm presented in Section 3.4 of Chapter 3 can be used as a reliable sentence selection algorithm while making decisions on the sentences to be returned in the list of updates.

4.3 Effectiveness of stemming

In this section, we detail the experiments conducted for studying the effectiveness of using stemming in tokenization of query and sentences. We prepared multiple runs for two different methods, one without using stemming and another method by using Porter stemmer [Porter, 1980]. For both methods, the rest of the criteria such as adaptive cutoff based sentence selection, no deduplication, and no query expansion were same.

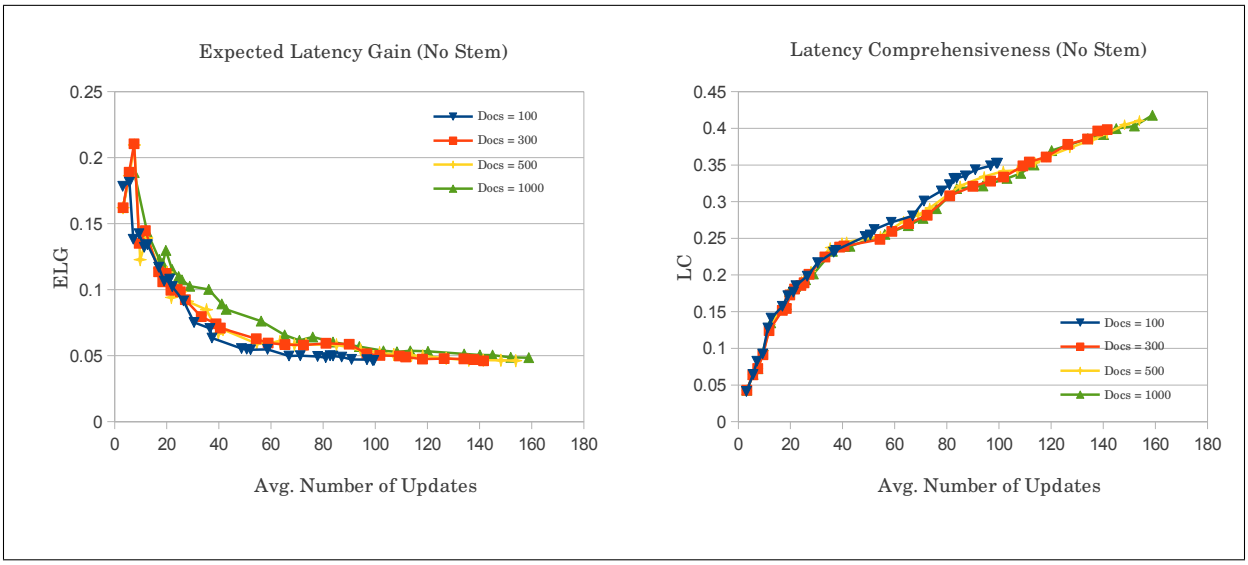

Figure 4.4: ELG and LC for without stemming runs (varying D_h and S_h in adaptive cutoff algorithm).

4.3.1 Results and Discussion

Figure 4.4 shows the ELG and LC values for the runs which do not use stemming. The parameter values for D_h (in adaptive sentence selection) was chosen from {100, 300, 500, 1000}, and the values for S_h were picked in the range from 1 to 1000. The X-axis in both graphs of Figure 4.4 show the average number of updates evaluated per query. From the graphs, we can observe that choosing the value of $D_h = 1000$, yields better ELG when compared to $D_h = 100$ for the same number of updates, but at the same time, it has lower LC compared to $D_h = 100$.

In order to compare the effectiveness of stemming, we fixed the value of $D_h = 1000$ and then prepared different runs changing S_h for both "with stemming" and "without stemming" approaches. Figure 4.5 shows the performance of systems in ELG and LC metrics. From the graph plots in Figure 4.5, we do not see much difference in the ELG and LC values for "stemming" vs "no stemming" approaches when approximately the same number of updates are returned.

Only when the average number of updates returned by the system is fewer than 20 per query, we see that there is a considerable difference in the ELG of both systems. We also see sharp increase and decrease in ELG of the system when the average number of updates is less than 20 per query. This is due to the high offset in the gain of the system created

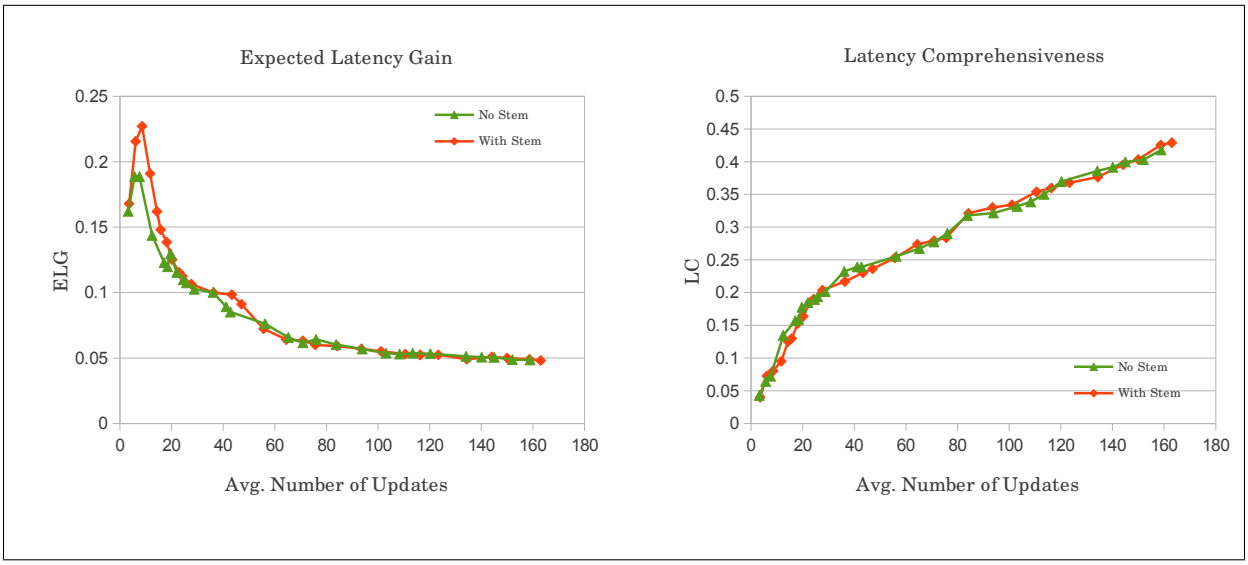

Figure 4.5: ELG and LC for "Stemming" vs "No Stemming" (D_h=1000 and varying S_h in adaptive cutoff algorithm)

by one query for which the system returned only one update.

Appendix B shows the list of runs for both algorithms, for which the graphs have been plotted in Figure 4.5.

Since overall there is no significant loss or gain in performance with the use of stemming in TST, we suggest researchers employ different techniques like Query Expansion in order to increase the recall (LC), i.e. coverage of nuggets in the system's list of updates.

4.4 Deduplication

In this section, we study different deduplication algorithms which can be used in a TS system. In the deduplication step of a TS system, a sentence is added to an existing list of updates returned by the system, if and only if there is no other sentence in the updates which is a near duplicate of the current sentence. Through this step, we maintain the novelty of the system's updates and there will be no penalty received by our system in the ELG and LC metrics.

The baseline run for the deduplication step is the "exact match" approach, where a sentence is considered to be duplicate only if there is another sentence already returned in the system's list of updates which is "exactly the same" as the current sentence.

Let s denote the current sentence and U denote the list of updates returned by the system so far.

Let the function $isDup(s, U)$ define whether the sentence s is duplicate of any other sentence $s' \in U$. For the baseline run, the function is defined as follows:

```
bool isDup(s, U)
{
    for each s' in U
       if (s == s')
          return true;

    return false;
}
```

4.4.1 Percent Match

In the "percent match" approach, we check for the percentage of words in the sentence s which are also present in the sentence $s' \in U$. If the percentage match crosses a fixed threshold, then the sentence s is considered as a duplicate. The functions are defined as:

```
bool isDup(s, U)
{
    for each s' in U
       if (percent_match(s, s') > cutoff)
          return true;

    return false;
}
double percent_match(s, s')
{
    count = 0;
    for each word w in s
       if s' contains w
          count++;
    return count/|s|;
}
```

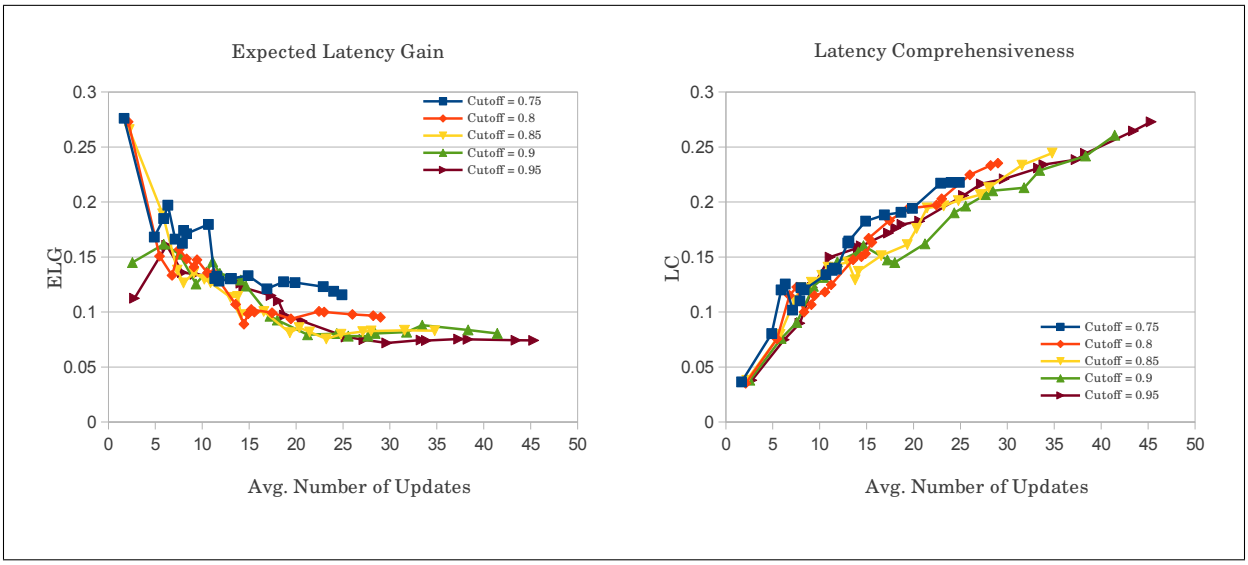

Figure 4.6: ELG and LC of runs for various cutoffs of "percent match" threshold

Figure 4.6 shows that the runs with cutoff for "percent match" = 0.75 perform better than the runs with other cutoffs in both the metrics ELG and LC. This is because with lower cutoffs the deduplication algorithm is highly selective and selects only updates which are unique and not contained in any of the previous updates. This improves the novelty of updates, and helps to improve the ELG and LC scores.

4.4.2 Cosine

In this deduplication approach, we calculate the cosine similarity between the sentences s and s' and when it crosses a fixed threshold, then the sentence s is discarded as a duplicate. The function is defined as follows:

```
double cosine_sim(s, s')
{
    vec1 = tokenize(s);
    vec2 = tokenize(s');

    sum = 0;
    for each token t in vec1:
       if vec2 contains token t:
```

```
            sum += vec1[t] * vec2[t];  // multiply frequencies

    sum1 = 0;
    for each token t in vec1:
       sum1 += vec1[t] * vec1[t];

    sum2 = 0;
    for each token t in vec2:
       sum2 += vec2[t] * vec2[t];

    return sum/sqrt(sum1 * sum2);
}
```

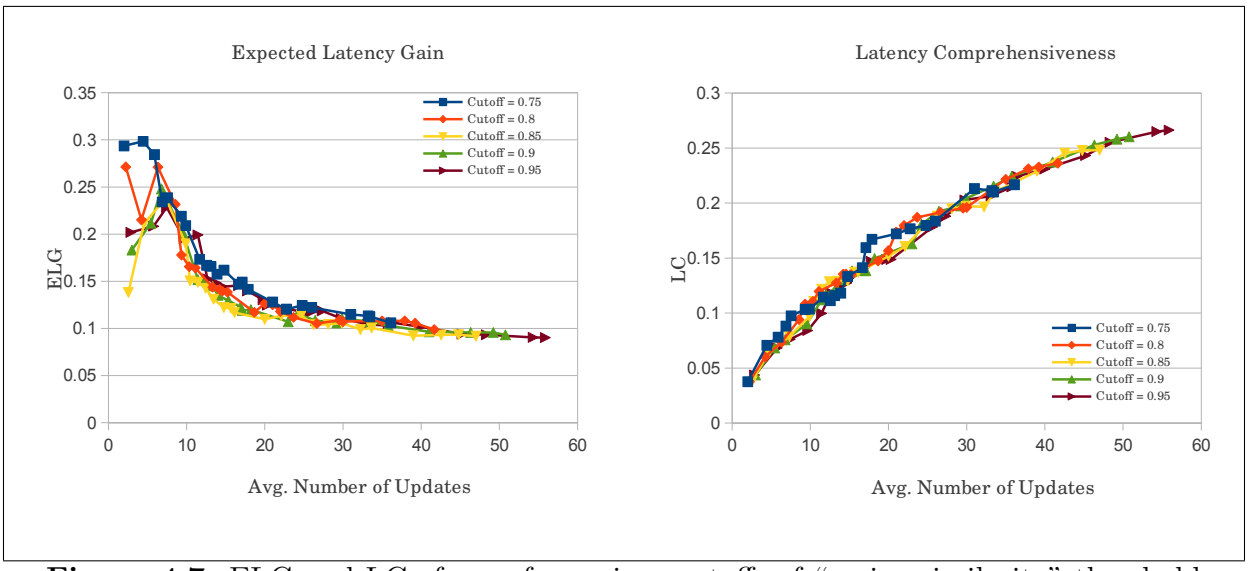

Figure 4.7: ELG and LC of runs for various cutoffs of "cosine similarity" threshold

Figure 4.7 shows that the runs with cutoff for "cosine similarity" = 0.75 perform slightly better than the runs with other cutoffs in both the metrics ELG and LC. This is similar to the pattern observed in the previous section. Having low cutoff for the deduplication measure helps in the algorithm to be more selective and return only updates which are unique. However, having very low thresholds can be a problem in erroneously discarding a sentence as duplicate.

4.4.3 Simhash

Simhash, a locality sensitive hashing scheme was first proposed by Charikar in 2002, which can detect similar objects based on the hash values [Charikar, 2002], and is also patented by Google[1]. Simhash was effectively used by Manku et al. [2007] to detect near duplicate documents from the repository containing multi-billion webpages. The authors used a 64-bit simhash fingerprint to effectively detect duplicate documents among 8 billion documents. Henzinger [2006] showed that Simhash performs better (higher precision) than shingling algorithm [Broder et al., 1997] while detecting near-duplicate webpages from a collection of 1.6 billion webpages.

Like the cutoff threshold scores used in the previous sections to detect duplicate sentences, we use a threshold for the similarity value calculated between two sentence hash values. We used 128-bit simhash for hashing and the sentences are tokenized with the whitespace space tokenizer while hashing.

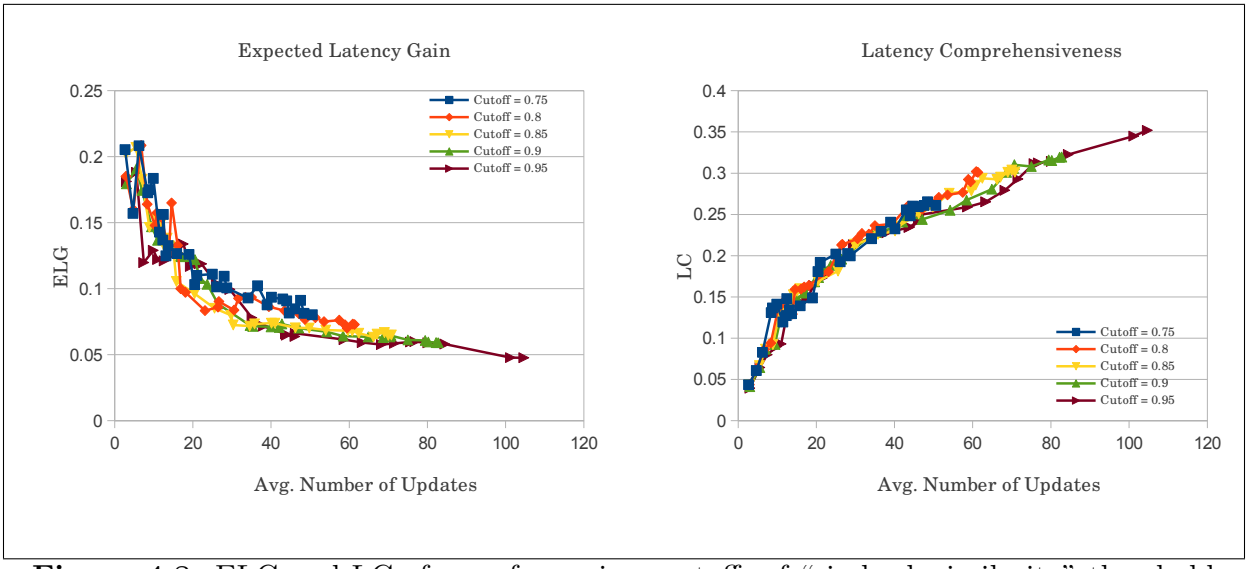

Figure 4.8: ELG and LC of runs for various cutoffs of "simhash similarity" threshold

Figure 4.8 shows the ELG and LC metrics of runs for various cutoffs used as deduplication threshold. We observe that runs with cutoff for "simhash similarity" = 0.75 perform slightly better than the runs with other cutoffs in both the metrics ELG and LC.

[1] http://www.google.com/patents/US7158961

4.4.4 Results and Discussion

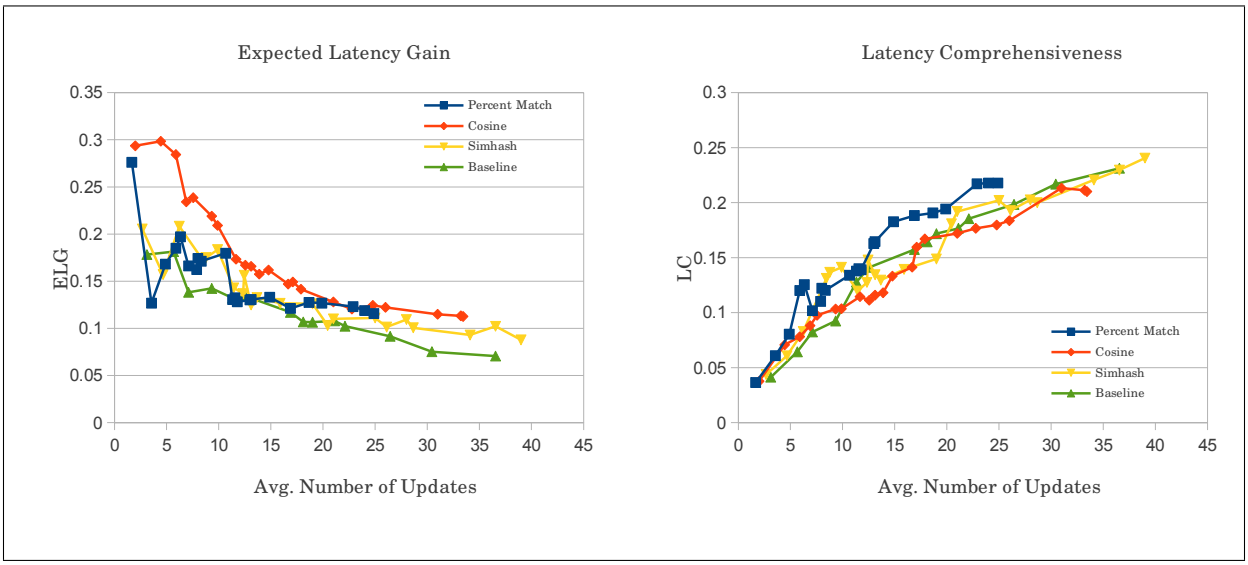

Figure 4.9: ELG and LC of runs for various deduplication methods

From Fig. 4.9 we can observe that Cosine similarity outperforms other deduplication algorithms in the ELG metric, whereas the percent match algorithm is better in the LC metric. We can also see that as the number of updates returned in the run increases, the cosine curve stabilizes to the same values of ELG (≈ 0.12) and LC (≈ 0.21) as the percent match curve. So, either of the two deduplication methods (cosine or percent match) could be used for deduplication with the cutoff as 0.75. For further experiments, we prefer cosine similarity as the deduplication method due to better performance in ELG with a small number of updates.

Appendix C shows the list of tables with metrics such as average number of updates, total number of sentences, ELG, and LC for various runs comparing different deduplication algorithms.

4.5 Automatic Query Expansion using Lin's distributional similarity

In Chapter 3, we detailed the query expansion method using Lin's distributional similarity which was used for preparing the runs for the TS track. But the method involved manual

selection of seed and expansion words which is a bottleneck for new event types. In this section, we propose an automatic query expansion method in which seed and expansion words are found automatically.

4.5.1 Seed words using KLD

Kullback-Leibler divergence (KLD) [Kullback and Leibler, 1951], is a measure for comparing two probability distributions. Given two probability distributions $p(x)$ & $q(x)$, the KL divergence is given by:

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x) log \frac{p(x)}{q(x)} dx$$

Large values for KLD indicate that the distributions are different, and when the distributions are identical, KLD is 0.

Let us assume we have two document collections, namely "R" and "NR" for relevant and non-relevant documents respectively for a given topic. For example, in TST if the event type is "earthquake", we collect all the earthquake related Wikipedia articles into document collection "R" and all other Wikipedia articles are added to "NR".

We now use the following method to rank the terms for the given topic:

```
for each term 't' in R:
   p(t) = N(t,R)/|R|;   //N(t,R) = number of times t occurs in R
   q(t) = N(t,NR)/|NR|;   //N(t,NR) = number of times t occurs in NR
   KL(t) = p(t)log(p(t)/q(t));

Rank all the terms t in decreasing order of KL(t)
```

With the parameter K_s, we automatically choose top K_s terms from the above list of terms as the seed words. Intuition here is that, if a term 't' is able to differentiate between the collections R and NR well, then it is a good candidate as a seed word. The idea here is to construct language model from relevant documents, also known as relevance model, and model from collection [Smucker et al., 2009]. We also removed the stop words from the list of terms ranked by KLD. The stop words list used is developed by Salton and Buckley [1971] for SMART IR system[2].

[2] http://www.lextek.com/manuals/onix/stopwords2.html

For each event type, we crawled the Wikipedia articles automatically from a single Wikipedia page which contains a list of those events. For example, the Wikipedia page on "List_of_20th-century_earthquakes"[3] contains the list of earthquakes that occurred between 1901 - 2000 with links to corresponding articles. These articles were automatically downloaded into the collection "R" only if the event occurred before the year 2011 adhering to track guidelines. For the "NR" collection, we downloaded all the articles crawling Wikipedia pages through links only if that article is not already downloaded to the document collection "R".

Table 4.1 shows the list of relevant wikipedia articles downloaded to set "R" for each event type. Table 4.2 shows the top terms identified by KLD for earthquake event type.

Earthquakes	http://en.wikipedia.org/wiki/List_of_earthquakes http://en.wikipedia.org/wiki/List_of_20th-century_earthquakes http://en.wikipedia.org/wiki/List_of_21st-century_earthquakes
Accidents	http://en.wikipedia.org/wiki/List_of_accidents_and_disasters_by_death_toll
Bombings	http://en.wikipedia.org/w/index.php?title=Special:Search&limit=500&offset=0&redirs=0&profile=default&search=bombings
Shootings	http://en.wikipedia.org/w/index.php?title=Special:Search&limit=500&offset=0&redirs=0&profile=default&search=shootings
Storms	http://en.wikipedia.org/wiki/2010_Atlantic_hurricane_season (year in the link ranges from 1901 to 2010)

Table 4.1: List of relevant Wikipedia articles crawled for each event type.

After building the ranked lists of KLD identified terms for each event type, we choose top K_s terms as the seed terms.

4.5.2 Merging lists of expansion words

The top 200 seed words (upper bound for the parameter K_s) for each event type, and the set of relevant documents "R" for that event type, are given as input to the Lin's

[3] http://en.wikipedia.org/wiki/List_of_20th-century_earthquakes

1. earthquake	11. reports	21. epicentre
2. tsunami	12. rupture	22. destroying
3. damage	13. buildings	23. event
4. fault	14. estimates	24. disaster
5. magnitude	15. collapsed	25. zone
6. quake	16. subduction	26. waves
7. aftershocks	17. landslides	27. scale
8. plate	18. intensity	28. shaking
9. epicenter	19. people	29. islands
10. seismic	20. tectonic	30. shock

Table 4.2: Examples of KLD identified seed terms for earthquake event type

distributional similarity algorithm explained in Section 2.4 of Chapters 2. The output would be the list of expansion terms (with Lin's score) found for each seed word.

Lin's score	**Expansion terms**
0.47551	earthquake
0.25742	event
0.21607	disaster
0.19040	aftershock
0.18767	shock
0.18450	tsunami
0.12674	magnitude
0.12449	damage
0.12100	tremor
0.11523	mainshock

Table 4.3: Examples of Lin identified expansion terms for seed word "quake"

Table 4.3 shows the expansion words identified for the seed word "quake" and the corresponding Lin's score for each word.

After the identification of expansion terms for each seed word as described above, we choose top K_e (parameter) expansion words for each seed word. The top K_s seed words and the list of top K_e expansion words for each seed word are merged into one list. Since, a seed word can potentially appear as an expansion word in one list and some expansion words might be common among the lists, while merging we add the Lin's score (which is already normalized) for same word. Then, finally in the merged list we choose the top K

(parameter) terms as the final expansion terms.

For tuning the parameters K_s (for seed words), K_e (for Lin's expansion terms) and K (final list of expanded terms), we randomly chose the test event (query 9) related to earthquakes as the training topic. The parameter K_s was varied in {20, 35, 50, 65, 80, 100}, the parameter K_e in {3, 5, 7, 10, 15, -1 (include all terms)}, and the parameter K was chosen from {50, 75, 100, 125, 150}.

After evaluating the ELG and LC metrics for the training topic, K_s was chosen to be 35, $K_e = 5$ and $K = 100$, for the automatic query expansion using Lin's method.

4.5.3 Results & Discussion

In order to evaluate the effectiveness of query expansion using Lin's distributional similarity compared to the baseline approach (no query expansion), we first fix the parameters and the rest of the methods as follows (based on the previous sections):

1. We use adaptive cutoff based sentence selection method for both query expansion and the baseline approach. We use the sentence selection parameters as $D_h = 100$ and $S_h = 250$ for both approaches.
2. We use stemming for both query expansion and baseline approaches.
3. We use Cosine similarity with cutoff = 0.75 as the deduplication method for both approaches.

Table 4.4 shows the ELG and LC values for the runs built using query expansion (Lin's distributional similarity) and the baseline run which doesn't use query expansion.

Smucker et al. [2007] show that randomization test, student's paired t-test and bootstrap test produce comparable p-values for the TREC ad-hoc retrieval system and suggest that IR researchers choose these methods compared to Wilcoxon and Sign tests. The authors [Smucker et al., 2007] suggest that randomization test be used as a distribution-free test for IR evaluation. Especially, with only 8 topics (topic 7 excluded by track organizers and topic 9 used for training) for TST, the randomization test is easy to perform with the number of permutations generated = 2^8, and the null hypothesis being that "the systems compared are identical" for the metric. So, unless otherwise specified in this section, we use randomization test for checking the statistical significance.

Topic Id	ELG (expanded)	LC (expanded)	ELG (baseline)	LC (baseline)
1	0.0798	0.733	0.079	0.6401
2	0.0924	0.3116	0.1258	0.2102
3	0.2572	0.0626	0.1652	0.0457
4	0.1434	0.199	0.1813	0.1601
5	0	0	0	0
6	0.0965	0.0153	0.0936	0.0088
8	0.0671	0.1232	0.1176	0.0799
#9	#0.1009	#0.4595	#0.1033	#0.3788
10	0.1676	0.4995	0.1687	0.3948
AVG	0.1130	**0.2430***	**0.1164**	0.1925

Note: *p<.05 for randomization test for expanded vs baseline with null hypothesis: "the systems are identical". # indicates the training topic was not used for the average and statistical significance tests.

Table 4.4: Comparisons of the ELG and LC metrics for Lin's method of expansion vs baseline (no query expansion) approach with the rest of the parameters and methods the same.

Table 4.4 shows a statistical significant improvement (p-value < 0.05 for randomization test) in the LC metric of the expanded run without any significant drop or change in the ELG metric compared to the baseline.

Figure 4.10 also shows the ELG and LC metrics of the expanded (query expansion with Lin's distributional similarity) and the baseline (without query expansion) runs. We can observe from the figure that query expansion with Lin's similarity helps in increasing the LC metric of the runs without losing on the ELG metric for the same number of updates evaluated.

Table 4.5 compares our system, which uses Lin's distributional similarity for query expansion along with other techniques mentioned earlier, to other systems submitted to the TS track.

Our system performs significantly better than 18 other systems in the track in the ELG metric, and the null hypothesis H_0 ("systems are identical") for ELG metric couldn't be rejected for the remaining 8 systems.

Our system performs significantly better than 4 other systems in the track in LC metric, and the null hypothesis H_0 ("systems are identical") for LC metric couldn't be rejected for 15 systems. Our system performs worse compared to 7 other systems in the LC metric, however there is significant gain in the ELG metric when compared to those systems. This

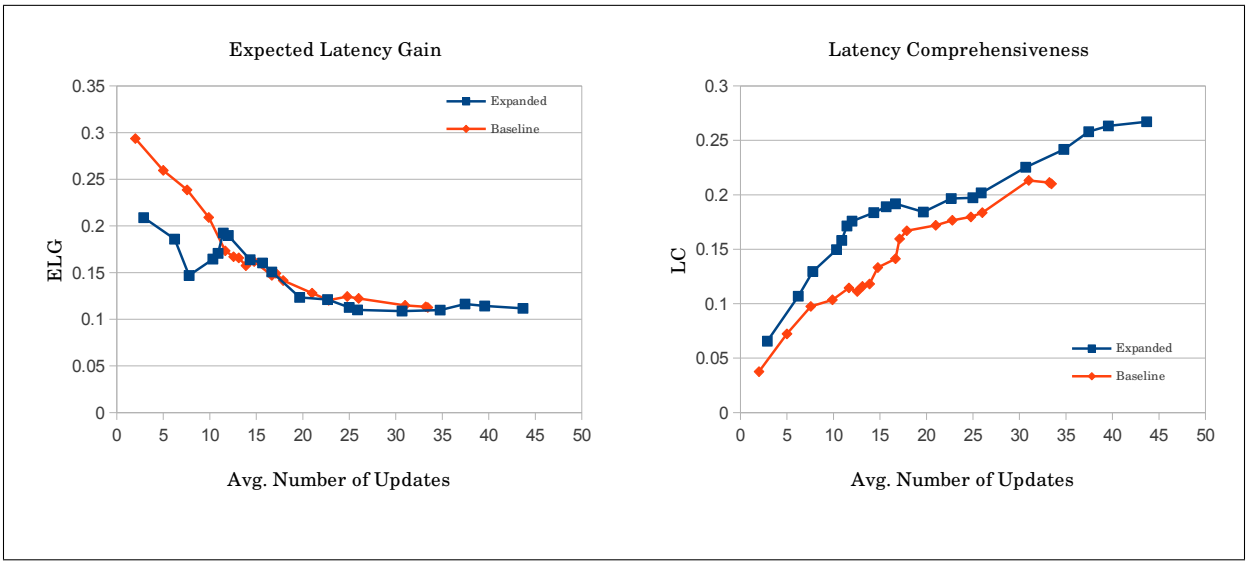

Figure 4.10: ELG and LC of the expanded vs baseline runs with avg. number of updates evaluated.

is because the other systems returned large number of updates and score highly on the LC metric and very poorly on the ELG metric.

It is important to note that, our system performs significantly better than 12 other systems in the track in at least one of the metrics and without performing worse on the other metrics. There is no system in the track which performs better than our system in at least one of the metrics without performing worse on the other.

In conclusion, we suggest that query expansion techniques like Lin's distributional similarity along with deduplication and adaptive cutoff based sentence selection, can be used effectively for building temporal summarization systems for tracking updates of known event types.

Run	ELG	LC	ELG Significance	LC Significance
cluster5	0.1465	0.1321	H_0 holds	Better*
run2	0.1317	0.2154	H_0 holds	H_0 holds
run1	0.1301	0.2169	H_0 holds	H_0 holds
TuneExternal2	0.1242	0.2010	H_0 holds	H_0 holds
TuneBasePred2	0.1197	0.2315	H_0 holds	H_0 holds
our system	**0.1130**	**0.2430**	-	-
cluster3	0.1054	0.1695	H_0 holds	H_0 holds
cluster2	0.0768	0.2364	H_0 holds	H_0 holds
uogTrNMTm1MM3	0.073	0.2146	H_0 holds	H_0 holds
BasePred	0.0715	0.3517	Better**	H_0 holds
cluster1	0.0693	0.2726	Better*	H_0 holds
cluster4	0.0692	0.2681	Better*	H_0 holds
Baseline	0.0670	0.3656	Better**	H_0 holds
uogTrNSQ1	0.0630	0.1839	Better**	H_0 holds
EXTERNAL	0.0575	0.4023	Better**	Worse*
uogTrNMTm3FMM4	0.0517	0.1780	Better*	H_0 holds
uogTrNMM	0.0477	0.2598	Better**	H_0 holds
uogTrEMMQ2	0.0429	0.2757	Better*	H_0 holds
SUS1	0.0375	0.1488	Better**	Better**
rg4	0.0278	0.4751	Better**	Worse**
rg3	0.0263	0.4635	Better**	Worse**
rg2	0.0227	0.5170	Better**	Worse**
rg1	0.0211	0.5170	Better**	Worse**
UWMDSqlec4t50	0.0189	0.5002	Better**	Worse**
UWMDSqlec2t25	0.0186	0.5050	Better**	Worse**
CosineEgrep	0.0073	0.0155	Better**	Better**
NormEgrep	0.0012	0.062	Better**	Better*

Note: * p<.05, ** p<.01

Table 4.5: Comparison of our new system with other systems in the track (ordered by ELG). "Better" means our system is significantly better. "H_0 holds" means the null hypothesis H_0 (systems are identical) couldn't be rejected. "Worse" means our system performs worse compared to that system.

Chapter 5

Evaluation of Temporal Summarization Systems

In this chapter we revisit the current evaluation methodology used by TS track. After identifying the limitations of the existing evaluation, we propose a modified evaluation method which reduces manual effort of assessors and also correlates well with the official track's evaluation of systems. We then extend the idea to a supervised learning approach which further reduces the manual effort of assessors and correlates well with the official track's evaluation of systems.

5.1 TST Evaluation for TREC 2013

TS track's evaluation method was already introduced in Section 1.4 of Chapter 1. In summary, the evaluation method is as follows [Aslam et al., 2014]:

1. For each test query (i.e. event), an assessor (TST used only one assessor) pools a list of nuggets along with their timestamps and the importance level of each nugget (as 1, or 2, or 3, with 3 being the most important). Aslam et al. [2014] define nugget as "a very short sentence, including only a single sub-event, fact, location, date, etc., associated with topic relevance".

   Nuggets are pooled from the Wikipedia event pages of the event along with the timestamp obtained from the revision history of the page. Table 5.1 shows a sample list of nuggets pooled for the training event.

queryId	nuggetID	timestamp	importance	nugget string
Train	TRAIN-1.001	1344701957	3	2012 Tabriz earthquake
Train	TRAIN-1.002	1344701957	3	a pair of destructive earthquakes
Train	TRAIN-1.003	1344701957	1	ccurred in northwestern Iran
Train	TRAIN-1.004	1344701957	2	Saturday August 11, 2012

Table 5.1: Sample list of nuggets pooled for the training event "iran earthquake".

queryId	updateId	nuggetId	start	end
1	1329993579-4ff6708235e2148ac570e7079d2e90d9-0	VMTS13.01.078	133	150
1	1329993579-4ff6708235e2148ac570e7079d2e90d9-0	VMTS13.01.077	154	175
1	1329993579-4ff6708235e2148ac570e7079d2e90d9-0	VMTS13.01.064	73	128
1	1329993579-4ff6708235e2148ac570e7079d2e90d9-0	VMTS13.01.050	0	30

Table 5.2: List of matches for the updateId: "1329993579-4ff6708235e2148ac570e7079d2e90d9-0"

2. For every test query (i.e. event), a list of updates from all the runs are pooled by track organizers. The track organizers select the top 60 updates (ranked by the confidence level of updates) per query from every run submitted to the track and add them to the pool of updates.

3. The assessor then prepares a list of matches (i.e. nugget-update pairs) for all the updates pooled. The assessor is provided with a user interface[1], as shown in Figure 5.1, to help the assessor mark the nuggets present in the update. These matches are also stored using the interface after selection of the nuggets by the assessor. Note that an update can contain multiple nuggets and a nugget can be present in multiple updates.

   Table 5.2 shows the list of matches stored for a particular updateId. "start" and "end" in the table specify the character positions in the update which contains the nugget.

4. The list of matches generated above is then used to evaluate the runs using a script, which calculates various metrics like Expected Gain, Expected Latency Gain, Comprehensiveness and Latency Comprehensiveness (discussed in Chapter 1).

In the above evaluation method, one of the main bottlenecks is step #3, in which the assessor manually identifies the nuggets in every pooled update using the interface shown

[1] http://fiji.ccs.neu.edu/~mattea/ts13/matchview/

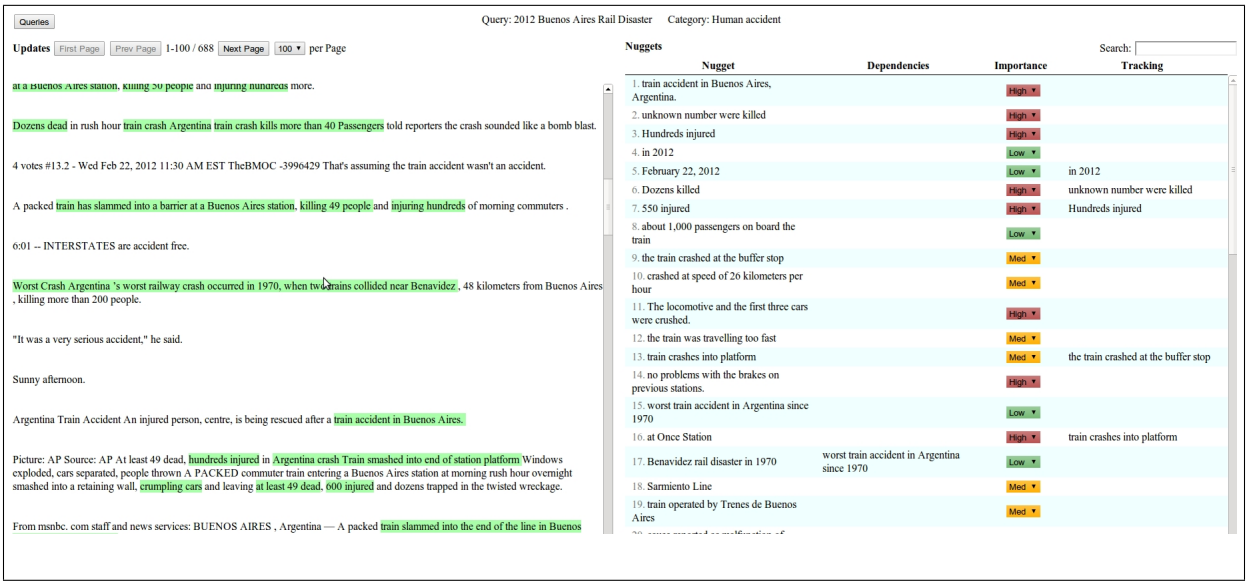

Figure 5.1: Matching interface used by assessor to match the nuggets to updates (Figure shown for Query 1).

in Figure 5.1. According to the TREC TST organizers, the assessor spent around 15 to 20 hours per query manually matching all the pooled updates with nuggets. Especially for query 6, the assessor spent nearly 30 hours. With more systems participating in the track, the number of pooled updates would be even larger, thereby making the matching process more time consuming. Similarly with an increase in the number of topics, this process of annotation would consume more time.

Table 5.3 shows the per-query statistics of the number of nuggets pooled, number of pooled updates, number of possible matching pairs evaluated by the assessor (i.e. number of nuggets $\times$ number of updates), and the number of actual matches found by the assessor.

In the column "#Updates" of Table 5.3, the value within the parentheses indicates the number of pooled updates for a particular query, where as the value outside the parentheses indicates the number of unique pooled updates for the query which was actually evaluated by the assessor.

Similarly for the column "#matchpairs" of Table 5.3, the value within the parentheses indicates the number of match pairs for a particular query, where as the value outside the parentheses indicates the number of unique match pairs for the query which was actually evaluated by the assessor.

queryId	#Nuggets	#Updates	#matchpairs	found matches
1	56	688 (779)	38528 (43624)	1167 (1172)
2	89	737 (912)	65593 (81168)	829 (836)
3	139	651 (762)	90489 (105918)	270 (271)
4	97	1192 (1463)	115624 (141911)	727 (730)
5	108	1069 (1069)	115452 (115452)	108 (109)
6	418	1331 (1517)	556358 (634106)	750 (760)
8	88	924 (1128)	81312 (99264)	245 (245)
9	45	703 (873)	31635 (39285)	360 (361)
10	37	521 (610)	19277 (22570)	349 (366)
ALL	1077	7816 (9113)	1114268 (1283298)	4805 (4850)

Table 5.3: Per-Query statistics with number of nuggets, pooled updates, match pairs and matches found by assessors. Numbers in the parentheses indicate the actual values , while numbers outside the parentheses in columns three and four indicate the unique counts (matched by the assessor).

In the last column of the Table 5.3, the value within the parentheses indicates the actual number of nugget-update pairs found by the assessor, but some of the pairs contained invalid nuggetId's which were not present in the pooled nuggets list. So, the correct value of the matches found is indicated outside the parentheses.

In the sections below, we propose methods for reducing the number of match pairs and the number of updates evaluated by the assessor, and use the matches found from these pairs for evaluating the systems (using Step 4 mentioned earlier). We check the correlation between the systems' rankings using the matches found by our method against the official track's rankings.

5.2 Preliminaries

In this section we define the formulae for computing a match score between a nugget and a sentence. These formulae will be used in the subsequent sections which describe the modified method for finding matches by assessors.

Notation: N denotes the nugget, S denotes the sentence, $N \cap S$ denotes the set of words common between N and S, $N - S$ indicates the set of words in N but not in S, w denotes a word, w' denotes the best expanded word of w (with max Lin score) present in S, $Lin(w, w')$ indicates the Lin's similarity score between w and w' (which is always < 1),

$RR(w, w')$ indicates the reciprocal rank of w' in the list of expanded words of w, $IDF(w)$ indicates the IDF weight of the word w in the collection of documents appearing before the particular query, $|N|$ and $|S|$ denote the length of nugget and sentence respectively.

$$\text{PercentMatch}(N, S) = \frac{|N \cap S|}{|N|}$$

$$\text{ScoreIDF}(N, S) = \frac{\sum_{w \in N \cap S} IDF(w)}{\sum_{w \in N} IDF(w)}$$

$$\text{ScoreLIN}(N, S) = \frac{\sum_{w \in N \cap S} IDF(w) + \sum_{w \in N-S; w' \in S} (IDF(w) * Lin(w, w'))}{\sum_{w \in N} IDF(w)}$$

$$\text{PercentLIN}(N, S) = \frac{|N \cap S| + \sum_{w \in N-S; w' \in S} Lin(w, w')}{|N|}$$

$$\text{ScoreRR}(N, S) = \frac{\sum_{w \in N \cap S} IDF(w) + \sum_{w \in N-S; w' \in S} (IDF(w) * RR(w, w'))}{\sum_{w \in N} IDF(w)}$$

$$\text{PercentRR}(N, S) = \frac{|N \cap S| + \sum_{w \in N-S; w' \in S} RR(w, w')}{|N|}$$

5.3 Reducing the number of matches evaluated

Table 5.3 shows the number of matching pairs and the number of updates evaluated by the assessor to identify the actual matches. From the table we can observe that out of the total possible 1,283,298 (or 1,114,268 unique) nugget-update pairs, only 4,805 pairs were found by the assessor as relevant. The remaining pairs were evaluated to be not relevant, i.e. the update doesn't contain the nugget. We hypothesize that if we can reduce the number of pairs evaluated by the assessor, i.e. automatically remove the pairs which do not match and without erroneously removing the actual matching pairs, it would save the assessor's time.

Upon close observation of the matches file generated by assessors, it was found that more than 98% of the updates which have at least one nugget present in them, also can contain up to a maximum of 5 nuggets. Only less than 2% of the total updates in the matches generated have more than 5 nuggets present in them. Intuitively this is understandable as well, because updates (i.e. sentences) in the given document collection have an average of 34 words (found earlier in Chapter 3), and with unique nuggets pooled by assessors (which

K	#matches evaluated	#Percentage Matches evaluated	ELG correlation	LC correlation
5	45565	3.5506	0.93883	0.90837
10	91130	7.1012	0.96149	0.94024
15	136695	10.6519	0.97211	0.95225
20	182260	14.2025	0.97742	0.96286
25	227825	17.7531	0.98805	0.98408
30	273390	21.3037	0.98274	0.97878
35	318955	24.8543	0.99202	0.98939

Table 5.4: Kendall's τ correlation of ELG and LC ranking of systems' when compared with official track's ranking

contain at least two to three words), the update can be estimated to contain fewer than 10 nuggets in an average case.

We employ this intuitive idea to retain only top K nuggets as potential matches (ranked by the ScoreLIN) for every update in the list of updates. Varying the parameter K, we prepare the matches file assuming the assessor judged only the matches of top K nuggets for each update.

Table 5.4 shows the correlation of the systems' ranking for ELG and LC metrics with the official track's ranking. We observe that by showing only 5 nuggets per update to the assessor for identifying potential matches, a high correlation (>0.9) with the official TREC ranking of systems is achieved, i.e. with annotating only $\approx 3.55\%$ of the original matches the ranking of the systems is stable. Generally, a "good" value of Kendall's τ is 0.8 but researchers use a threshold of 0.9 [Lin and Demner-Fushman, 2005].

The exact estimate of the time saved for assessor is not calculated theoretically because the assessor still annotates the entire list of updates as before (shown in Figure 5.1) and the exact time gained by showing only top K nuggets for every update as potential matches is unknown. However, we estimate that there would be substantial amount of time saved for the assessor for queries like topic 6 (see Table 5.3), where for all the 1331 unique pooled updates the assessor has to identify matches from a huge list of 418 nuggets, which is a time consuming effort given the size of the list of nuggets. With the new method only top K nuggets will be shown for every update which reduces the assessor's time browsing through the list of all nuggets.

Another method to improve the efficiency of the system for the assessor is by ranking the nuggets using ScoreLIN for every update. This way the assessor can go through the ranked

list of nuggets for every update with decreasing probability of relevance, and according to Probability Ranking Principle this system would be more efficient for the assessor than the existing system.

In the next section, we propose a supervised learning approach where the assessor does not need to manually match all the pooled updates. With sufficient training sample size of updates per query, our system predicts the matches for the remaining pooled updates and the total matches generated can be used to rank the systems with high correlation to the official track's rankings in both metrics.

5.4 Supervised learning approach

Let us assume we are given a labeled dataset of matches with features, i.e. we have nugget-update pairs with certain feature values and class labels: "true" or "false" assigned based on whether or not the update contains the nugget. In this section, we try to build a binary classifier which can predict the label of a new nugget-update pair based on the feature values.

We selected the following features for the nugget-update pair (N, S): $|N|$, $|S|$, PercentMatch(N, S), ScoreIDF(N, S), ScoreLIN(N, S), PercentLIN(N, S), ScoreRR(N, S), PercentRR(N, S), which are detailed in Section 5.2 earlier.

Our total dataset for the TS track (for all the queries) as shown in Table 5.3 has 1,283,298 negative labels and 4,805 positive labels. This is a highly imbalanced (or skewed) dataset like many of the practical applications of bioinformatics, text classification, detection of oil spills, etc. which faces the challenge of misclassification of minority class being costly [Chawla, 2005; Mollineda and Sotoca, 2007]. Different resampling strategies like random oversampling of the minority class samples with replacement, random undersampling of majority class, focused undersampling or oversampling, synthetic generation of new samples (SMOTE) [Chawla et al., 2011], selective pre-processing of imbalanced data (SPIDER) [Stefanowski and Wilk, 2008] have been proposed by researchers to increase the classification accuracy of the minority class. Survey papers such as [Japkowicz et al., 2000; Chawla, 2005; He and Garcia, 2009; Galar et al., 2012; Longadge and Dongre, 2013] explain different strategies proposed by researchers to solve this problem.

We initially picked a random subsample (of 30% size) from the whole data set for training and testing classifiers. We trained and tested different classifiers: J48, Naive Bayes, Logistic Regression and Nearest Neighbour (NN1), with resampling techniques: random oversampling, random undersampling and SMOTE. However, the classifiers achieved low

classification rate on the minority class i.e. the F-measure of the "true" class for the test data was lower than state-of-the-art systems like POURPRE.

Working on the same assumption as explained in the previous section, we now retain only the top 30 nuggets (ranked by ScoreLIN) for every update (assuming the smallest nugget can be of one word and the update size is 34 on average obtained in Chapter 3). This helps in reducing the total dataset size from 1,283,298 pairs to 273,390 pairs only. We work on the reduced dataset as follows:

1. We first fix the training sample size (T% of total updates). For a fixed T, we cross-validated the following classifiers: J48, Naive Bayes, J48 with AdaBoost. J48 with AdaBoost [Freund et al., 1996] performs well on the random training sample of sizes (T = 10, 20, 30, 40) with cross-validation and in general is proven to work well for imbalanced datasets by Seiffert et al. [2008].

2. For a fixed T, we split the total data set randomly into training (with T% updates) and test for every query, repeating 100 times.

3. We then train the J48 with AdaBoost classifier on one full training sample per query, and predicted the values on the corresponding test sample, repeating 100 times for the samples created above per query.

4. Then the corresponding "training" and "predicted" outputs for each query are combined to form a single match file. 100 match files are randomly chosen.

5. Each match file is then used to rank the systems and the Kendall's τ correlation coefficient is calculated with the official rankings. The Kendall's τ correlation coefficient is averaged over the 100 match files and is reported with 95% confidence interval of the mean in the Table 5.5.

5.5 Results & Discussion

Table 5.5 shows the results of the supervised learning approach with varying training size. For training size of more than 20%, rankings of systems by our method correlates well ($\tau > 0.9$) with the official track's rankings. This means that only 20% of the updates per query need to be marked/annotated by the assessor, and these annotations along with predictions by our classifier on the rest of the sentences can be used to rank systems.

Train %Updates	%Matches evaluated	ELG correlation	LC correlation
10	2.1297	0.869 (±0.007)	0.907 (±0.005)
20	4.2617	0.905 (±0.007)	0.926 (±0.004)
30	6.3913	0.925 (±0.005)	0.933 (±0.004)
40	8.5210	0.939 (±0.004)	0.942 (±0.004)
50	10.6530	0.945 (±0.003)	0.948 (±0.003)

Note: Values in parentheses indicate the 95% confidence interval.

Table 5.5: Kendall's τ correlation of ELG and LC ranking of systems when compared with the official track's ranking.

POURPRE when used to automatically match nuggets to sentences, without any user annotation, achieves $\tau = 0.834$ for the ELG metric, but doesn't correlate well with the systems ranking by LC metric ($\tau = 0.77$). We adapted POURPRE for TST by using a threshold for the match score which maximizes the classification accuracy.

In summary, with our proposed method of using binary classifier for predicting nuggets in sentences where assessor annotates only 20% of the updates, the ranking of the systems is reliable. This also means that the assessor's time could be saved by 80% when compared to the original track's evaluation where all the updates need to be annotated. And as the percentage of the training sample increases, the system tends to become closer to the actual track's evaluation.

Chapter 6

Conclusion & Future Work

6.1 Conclusion

In this thesis we have laid out a streamlined approach to build a Temporal Summarization system to track real time updates for disaster related event types such as accident, bombing, earthquake, shooting, and storm. We studied the effectiveness of various techniques which could be used in building a Temporal Summarization system. We propose a modified evaluation method for the evaluation of Temporal Summarization systems which can reduce the manual effort required from assessor, but at the same time correlate well with the official TS track's evaluation for TREC 2013.

In Chapter 3, we outlined the process of building the TS system while participating in the Temporal Summarization track of TREC 2013. We implemented novel techniques like adaptive cutoff based sentence selection for the selection of sentences in the updates, and used Lin's distributional similarity as a query expansion technique to find related words to a set of seed words of particular event type.

In Chapter 4, we used the same experimental setup to test the effectiveness of various techniques and algorithms while building a temporal summarization system. In particular, we tested the adaptive cutoff based sentence selection algorithm and found it to be effective compared to the baseline algorithm of returning top K sentences per hour. We also found that through adaptive sentence selection, there was no latency penalty awarded to the Expected Gain achieved by the system unlike the baseline approach. We found that there is not much difference in the runs built using stemming and no stemming. It was observed that deduplication algorithms are useful in the Temporal Summarization system, and found

that Cosine Similarity performs well compared to other approaches like Simhash. Later we showed that Lin's distributional similarity technique of finding related terms for query expansion is a promising approach to improve the coverage of nuggets (recall or LC) by the TS system without affecting the ELG.

In Chapter 5, we propose an evaluation method where the assessor is shown a ranked list of nuggets (or only top K nuggets) for each update, and the matches thus found by the assessor could be used for the evaluation of the systems. We later propose a supervised learning method to predict matches for the pooled updates with sufficient training size for each query. We show that with only 20% of training data (i.e. 20% of total updates annotated by the assessor), our semi-automatic evaluation system correlates well with the official track's evaluation of systems, i.e., the assessor's time could be saved by nearly 80% without affecting the evaluation of systems.

6.2 Future Work

Temporal Summarization track is introduced for the first time at TREC 2013. The organizers used 9 topics to evaluate the systems. We would like to test our new TS system (described in Chapter 4) on more topics by participating in the next year's Temporal Summarization track.

We would also like to do the following as part of future work:

1. Investigate the time taken by an assessor to build the matches (nugget-update) pairs, for the following approaches:
   (a) The current approach of matching all the pooled updates to all the nuggets per query.
   (b) The approach of showing only top K (=5) nuggets as ranked by the ScoreLIN (described in Chapter 5) for every update.
   (c) The approach of showing ranked list of nuggets for every update.
   (d) The machine learning approach, where only 20% of the updates per query are matched by the assessor.
2. Investigate other query expansion techniques, such as Pseudo-Relevance Feedback (PRF), combination of PRF and Lin's Distributional Similarity, etc.

3. Investigate different retrieval models for the retrieval of sentences for Temporal Summarization systems.

4. In the machine learning approach, we would like to use textual features like tagged entities and typed dependency relations to further improve the classification accuracy.

5. Current evaluation of the track uses two metrics, namely Expected Latency Gain (ELG) and Latency Comprehensiveness (LC). Direct comparison of two systems is difficult due to the existence of two metrics both of which are important. We would like to investigate the design of a single metric along the lines of n-DCG, TBG [Smucker and Clarke, 2012], which could be useful in ranking the systems.

6. In the current evaluation of TS systems used for TREC 2013, top 60 updates per query are pooled for every system. If a system returns more updates, the possibility of the system covering more pooled updates increases, hence the LC metric could increase. However, the user may not want to see so many updates. We wish to explore the design of an evaluation metric which correlates well with user satisfaction.

APPENDICES

Appendix A

Adaptive Cutoff vs Fixed Cutoff

K	Total Sentences	#Updates	EG	ELG	C	LC
1	2162	12.2222	0.1462	0.0788	0.0861	0.0895
2	4322	21.8889	0.1331	0.0519	0.117	0.1077
3	6482	30.8889	0.0846	0.0391	0.1495	0.1261
4	8642	38.6667	0.075	0.0406	0.1617	0.1329
5	10802	44.8889	0.0769	0.0421	0.1728	0.139
6	12962	52.1111	0.0688	0.0405	0.1867	0.1576
7	15122	55.7778	0.0654	0.0382	0.1965	0.1689
8	17282	61.8889	0.0624	0.0385	0.2013	0.1741
9	19441	66.7778	0.0603	0.0382	0.2141	0.1935
10	21600	69	0.0586	0.0373	0.2209	0.2028
11	23758	72.2222	0.0565	0.0368	0.2234	0.2136
12	25916	74.8889	0.0551	0.0357	0.2249	0.2196
13	28074	77.8889	0.0514	0.0349	0.2336	0.2425
14	30231	81.3333	0.0508	0.0344	0.2411	0.2515
15	32386	83.7778	0.0508	0.0334	0.2419	0.2542
16	34539	86.5556	0.0606	0.0335	0.2512	0.2569
17	36692	89.1111	0.0592	0.0324	0.2572	0.2662
18	38843	91.6667	0.0598	0.0323	0.2584	0.2679
19	40990	93.5556	0.0571	0.0322	0.2584	0.2682
20	43137	95.3333	0.0565	0.0319	0.2584	0.2682
21	45285	97.3333	0.0535	0.0312	0.2561	0.2697
22	47431	98.6667	0.055	0.0311	0.2582	0.2746
23	49577	100	0.0544	0.0307	0.2582	0.2747
24	51722	101.778	0.0529	0.0299	0.2582	0.2755
25	53867	104.778	0.0507	0.0297	0.2585	0.2802
26	56011	105.889	0.0499	0.0294	0.2585	0.2826
27	58153	108.111	0.0502	0.0315	0.2593	0.2837
28	60295	108.667	0.0499	0.0314	0.2593	0.2837
29	62437	111.222	0.0482	0.0306	0.2593	0.2914
30	64579	114	0.0483	0.0335	0.2663	0.3009
31	66720	116.333	0.0468	0.0351	0.2663	0.303
32	68859	117.778	0.0472	0.0341	0.2701	0.3038
33	70998	119.778	0.0471	0.0338	0.2704	0.305
34	73138	121.889	0.0466	0.034	0.2729	0.3104
35	75278	123	0.0475	0.0341	0.2754	0.3112
36	77416	124.889	0.0473	0.0336	0.2757	0.3112
37	79553	126	0.0474	0.0363	0.2765	0.3132
38	81689	127.444	0.0473	0.0362	0.2765	0.3132
39	83824	129.556	0.0478	0.0361	0.2813	0.3181

Table A.1: Fixed Cutoff Algorithm returning top 'K' sentences

S_h	Total Sentences	#Updates	EG	ELG	C	LC
1	525	3.2222	0.1032	0.1621	0.0279	0.0429
5	1268	5.6667	0.1328	0.1886	0.042	0.0644
10	1683	7.5556	0.1452	0.1886	0.0467	0.0723
20	2603	12.4444	0.1063	0.1437	0.0888	0.1347
25	3104	17.1111	0.0958	0.1229	0.1079	0.157
30	3628	18.4444	0.0832	0.1198	0.1092	0.1594
35	4014	19.6667	0.0908	0.1298	0.1194	0.1776
40	4834	22	0.1171	0.1154	0.1239	0.1847
45	5411	24.5556	0.1125	0.11	0.1265	0.1895
50	5743	25.7778	0.1105	0.1074	0.1289	0.1936
60	6526	28.7778	0.1104	0.1026	0.1405	0.2013
75	8438	36.1111	0.1142	0.1002	0.1598	0.2323
90	10735	41.1111	0.0804	0.0892	0.1636	0.2392
100	11445	42.7778	0.0773	0.0851	0.1636	0.2392
125	18757	56.2222	0.0672	0.0762	0.1754	0.2554
150	25373	65.2222	0.0603	0.0657	0.1874	0.2674
175	29912	70.8889	0.0568	0.0616	0.1965	0.2773
200	34264	76	0.058	0.0642	0.2074	0.2904
250	42372	83.7778	0.0558	0.0603	0.2223	0.3178
300	55708	93.8889	0.0543	0.0569	0.2269	0.3215
350	69219	103.111	0.0483	0.0537	0.2366	0.3317
400	78921	108.333	0.0471	0.053	0.2419	0.3386
450	87880	113.444	0.0472	0.0537	0.2478	0.3501
500	98849	120.222	0.0469	0.0533	0.26	0.3696
600	122536	134.111	0.0433	0.0513	0.2761	0.3859
700	141578	140.111	0.0424	0.0506	0.2828	0.3916
800	163227	145	0.0422	0.0505	0.287	0.3996
900	185143	152	0.0407	0.0488	0.2913	0.4031
1000	205649	158.889	0.0404	0.0485	0.3003	0.4178

Table A.2: Adaptive cutoff based sentence selection algorithm, $D_h = 1000$

Appendix B

Stemming vs No Stemming

S_h	Total Sentences	#Updates	EG	ELG	C	LC
1	525	3.2222	0.1032	0.1621	0.0279	0.0429
5	1268	5.6667	0.1328	0.1886	0.042	0.0644
10	1683	7.5556	0.1452	0.1886	0.0467	0.0723
20	2603	12.4444	0.1063	0.1437	0.0888	0.1347
25	3104	17.1111	0.0958	0.1229	0.1079	0.157
30	3628	18.4444	0.0832	0.1198	0.1092	0.1594
35	4014	19.6667	0.0908	0.1298	0.1194	0.1776
40	4834	22	0.1171	0.1154	0.1239	0.1847
45	5411	24.5556	0.1125	0.11	0.1265	0.1895
50	5743	25.7778	0.1105	0.1074	0.1289	0.1936
60	6526	28.7778	0.1104	0.1026	0.1405	0.2013
75	8438	36.1111	0.1142	0.1002	0.1598	0.2323
90	10735	41.1111	0.0804	0.0892	0.1636	0.2392
100	11445	42.7778	0.0773	0.0851	0.1636	0.2392
125	18757	56.2222	0.0672	0.0762	0.1754	0.2554
150	25373	65.2222	0.0603	0.0657	0.1874	0.2674
175	29912	70.8889	0.0568	0.0616	0.1965	0.2773
200	34264	76	0.058	0.0642	0.2074	0.2904
250	42372	83.7778	0.0558	0.0603	0.2223	0.3178
300	55708	93.8889	0.0543	0.0569	0.2269	0.3215
350	69219	103.111	0.0483	0.0537	0.2366	0.3317
400	78921	108.333	0.0471	0.053	0.2419	0.3386
450	87880	113.444	0.0472	0.0537	0.2478	0.3501
500	98849	120.222	0.0469	0.0533	0.26	0.3696
600	122536	134.111	0.0433	0.0513	0.2761	0.3859
700	141578	140.111	0.0424	0.0506	0.2828	0.3916
800	163227	145	0.0422	0.0505	0.287	0.3996
900	185143	152	0.0407	0.0488	0.2913	0.4031
1000	205649	158.889	0.0404	0.0485	0.3003	0.4178

Table B.1: No stemming, $D_h = 1000$

S_h	Total Sentences	#Updates	EG	ELG	C	LC
1	459	3.5556	0.1072	0.1679	0.0268	0.04
5	1177	6.1111	0.146	0.2155	0.048	0.0726
10	1689	8.5556	0.2401	0.2271	0.0537	0.0803
20	2484	11.6667	0.2182	0.1909	0.0658	0.0949
25	2973	14.3333	0.1821	0.1624	0.0887	0.1251
30	3332	15.7778	0.1763	0.1482	0.0916	0.1299
35	3902	18.1111	0.1693	0.1386	0.1026	0.1529
40	4440	20.2222	0.1215	0.1252	0.1087	0.1636
45	5044	22.8889	0.1149	0.1155	0.1252	0.1852
50	5594	24.2222	0.1141	0.1124	0.1285	0.1897
60	6547	27.6667	0.1102	0.1065	0.1402	0.2036
75	8841	36.3333	0.114	0.0999	0.1484	0.2169
90	10873	43.3333	0.1122	0.0984	0.1576	0.2303
100	13611	47.1111	0.0923	0.0911	0.1604	0.2363
125	17684	55.6667	0.0644	0.0724	0.1717	0.2531
150	25814	64.4444	0.0583	0.0641	0.192	0.2739
175	32885	70.8889	0.0579	0.0632	0.1999	0.2794
200	37293	75.6667	0.0537	0.0601	0.2014	0.2838
250	47945	84.2222	0.0551	0.0591	0.226	0.3213
300	58004	93.5556	0.0524	0.057	0.2352	0.3301
350	67286	101.111	0.0503	0.0551	0.2377	0.3342
400	78575	110.556	0.048	0.053	0.2499	0.354
450	89141	116.333	0.0471	0.0524	0.2531	0.3598
500	97943	123.333	0.0473	0.0525	0.2607	0.3679
600	121218	134.333	0.0415	0.0491	0.2689	0.3765
700	142951	144.111	0.0427	0.0509	0.2848	0.3954
800	166385	150	0.0417	0.05	0.289	0.4033
900	189295	158.667	0.0406	0.049	0.3023	0.4256
1000	208436	163	0.0398	0.0481	0.3042	0.4291

Table B.2: Use of stemming, $D_h = 1000$

Appendix C

Percent Match vs Cosine similarity vs Simhash

S_h	percentmatch	cosine	simhash	ELG_percent	ELG_cosine	ELG_simhash
1	1.6667	2	2.6667	0.2761	0.2936	0.2054
5	3.5556	4.4444	4.6667	0.1267	0.2984	0.157
10	4.8889	5.8889	6.2222	0.1681	0.2843	0.2083
15	5.8889	6.8889	8.4444	0.185	0.2341	0.1727
20	6.3333	7.5556	8.7778	0.1971	0.2386	0.175
25	7.1111	9.3333	9.8889	0.1662	0.219	0.1835
30	7.8889	9.8889	12.4444	0.1624	0.2091	0.1563
35	8	11.6667	11.4444	0.1742	0.1733	0.1429
40	8.3333	12.5556	12.3333	0.1712	0.167	0.1369
45	10.6667	13.1111	13.6667	0.1797	0.1658	0.1327
50	11.3333	13.8889	13.1111	0.1305	0.1575	0.1249
60	11.5556	14.7778	15.8889	0.1325	0.1618	0.1266
75	11.7778	16.6667	19	0.1281	0.147	0.126
90	13	17.1111	20.4444	0.1301	0.1492	0.1031
100	13.1111	17.8889	21	0.1306	0.1415	0.1101
125	14.8889	21	25	0.1331	0.128	0.111
150	16.8889	22.7778	26.1111	0.1211	0.1204	0.1014
175	18.6667	24.7778	28	0.1275	0.1244	0.1094
200	19.8889	26	28.6667	0.1268	0.1223	0.1006
250	22.8889	31	34.1111	0.1231	0.1149	0.093
300	24	33.2222	36.5556	0.1189	0.1133	0.1022
350	24.8889	33.4444	39	0.1157	0.1126	0.0878

Table C.1: Deduplication Comparison for ELG metric along with the average number of updates, $D_h = 100$, Deduplication Cutoff= 0.75

S_h	percentmatch	cosine	simhash	LC_percent	LC_cosine	LC_simhash
1	1.6667	2	2.6667	0.0365	0.0376	0.0437
5	3.5556	4.4444	4.6667	0.0609	0.0707	0.0608
10	4.8889	5.8889	6.2222	0.0804	0.0781	0.0828
15	5.8889	6.8889	8.4444	0.1201	0.0882	0.1311
20	6.3333	7.5556	8.7778	0.1255	0.0975	0.1367
25	7.1111	9.3333	9.8889	0.1018	0.1033	0.1412
30	7.8889	9.8889	12.4444	0.1102	0.1036	0.1479
35	8	11.6667	11.4444	0.1222	0.1145	0.1196
40	8.3333	12.5556	12.3333	0.1202	0.1112	0.1274
45	10.6667	13.1111	13.6667	0.134	0.1159	0.1296
50	11.3333	13.8889	13.1111	0.1378	0.1181	0.1344
60	11.5556	14.7778	15.8889	0.1399	0.1332	0.1393
75	11.7778	16.6667	19	0.139	0.1413	0.1488
90	13	17.1111	20.4444	0.1628	0.1595	0.1808
100	13.1111	17.8889	21	0.1646	0.167	0.1919
125	14.8889	21	25	0.1826	0.172	0.202
150	16.8889	22.7778	26.1111	0.1883	0.1766	0.1927
175	18.6667	24.7778	28	0.1906	0.1797	0.2025
200	19.8889	26	28.6667	0.1942	0.1836	0.1999
250	22.8889	31	34.1111	0.2171	0.2132	0.2206
300	24	33.2222	36.5556	0.2178	0.2113	0.2296
350	24.8889	33.4444	39	0.2178	0.2101	0.2405

Table C.2: Deduplication Comparison for LC metric along with the average number of updates, $D_h = 100$, Deduplication Cutoff= 0.75

S_h	percentmatch	cosine	simhash	ELG_percent	ELG_cosine	ELG_simhash
1	328	243	304	0.2761	0.2936	0.2054
5	770	675	791	0.1267	0.2984	0.157
10	1072	954	1042	0.1681	0.2843	0.2083
15	1443	1235	1438	0.185	0.2341	0.1727
20	1624	1396	1635	0.1971	0.2386	0.175
25	1881	1681	1903	0.1662	0.219	0.1835
30	2171	1866	2140	0.1624	0.2091	0.1563
35	2373	2199	2279	0.1742	0.1733	0.1429
40	2621	2376	2524	0.1712	0.167	0.1369
45	2935	2614	2808	0.1797	0.1658	0.1327
50	3181	2828	2975	0.1305	0.1575	0.1249
60	3436	3179	3353	0.1325	0.1618	0.1266
75	3869	3832	3933	0.1281	0.147	0.126
90	4725	4479	4742	0.1301	0.1492	0.1031
100	4925	4820	5115	0.1306	0.1415	0.1101
125	5902	5680	6478	0.1331	0.128	0.111
150	6642	6485	7056	0.1211	0.1204	0.1014
175	7619	7467	8191	0.1275	0.1244	0.1094
200	8037	8094	8710	0.1268	0.1223	0.1006
250	9552	9795	10607	0.1231	0.1149	0.093
300	10810	11357	11575	0.1189	0.1133	0.1022
350	11375	11932	12809	0.1157	0.1126	0.0878

Table C.3: Deduplication Comparison for ELG metric along with the total number of sentences, $D_h = 100$, Deduplication Cutoff= 0.75

S_h	percentmatch	cosine	simhash	LC_percent	LC_Cosine	LC_Simhash
1	328	243	304	0.0365	0.0376	0.0437
5	770	675	791	0.0609	0.0707	0.0608
10	1072	954	1042	0.0804	0.0781	0.0828
15	1443	1235	1438	0.1201	0.0882	0.1311
20	1624	1396	1635	0.1255	0.0975	0.1367
25	1881	1681	1903	0.1018	0.1033	0.1412
30	2171	1866	2140	0.1102	0.1036	0.1479
35	2373	2199	2279	0.1222	0.1145	0.1196
40	2621	2376	2524	0.1202	0.1112	0.1274
45	2935	2614	2808	0.134	0.1159	0.1296
50	3181	2828	2975	0.1378	0.1181	0.1344
60	3436	3179	3353	0.1399	0.1332	0.1393
75	3869	3832	3933	0.139	0.1413	0.1488
90	4725	4479	4742	0.1628	0.1595	0.1808
100	4925	4820	5115	0.1646	0.167	0.1919
125	5902	5680	6478	0.1826	0.172	0.202
150	6642	6485	7056	0.1883	0.1766	0.1927
175	7619	7467	8191	0.1906	0.1797	0.2025
200	8037	8094	8710	0.1942	0.1836	0.1999
250	9552	9795	10607	0.2171	0.2132	0.2206
300	10810	11357	11575	0.2178	0.2113	0.2296
350	11375	11932	12809	0.2178	0.2101	0.2405

Table C.4: Deduplication Comparison for LC metric along with the total number of sentences, $D_h = 100$, Deduplication Cutoff= 0.75

References

James Allan, Courtney Wade, and Alvaro Bolivar. Retrieval and novelty detection at the sentence level. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 314–321. ACM, 2003.

Javed Aslam, Fernando Diaz, Matthew Ekstrand-Abueg, Virgi Pavlu, and Tetsuya Sakai. Trec 2013 temporal summarization. In *Proceedings of TREC 2013*, 2014.

Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

Niranjan Balasubramanian, James Allan, and W Bruce Croft. A comparison of sentence retrieval techniques. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 813–814. ACM, 2007.

Gaurav Baruah, Rakesh Guttikonda, Adam Roegiest, and Olga Vechtomova. University of waterloo at the trec 2013 temporal summarization track. In *Proceedings of TREC 2013*, 2014.

Jagdev Bhogal, Andy Macfarlane, and Peter Smith. A review of ontology based query expansion. *Information processing & management*, 43(4):866–886, 2007.

Bodo Billerbeck and Justin Zobel. Questioning query expansion: An examination of behaviour and parameters. In *Proceedings of the 15th Australasian database conference-Volume 27*, pages 69–76. Australian Computer Society, Inc., 2004.

Catherine Blake. A comparison of document, sentence, and term event spaces. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 601–608. Association for Computational Linguistics, 2006.

Andrei Z Broder, Steven C Glassman, Mark S Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8):1157–1166, 1997.

Stefan Büttcher, Charles LA Clarke, and Gordon V Cormack. *Information retrieval: Implementing and evaluating search engines*. MIT Press, 2010.

Claudio Carpineto and Giovanni Romano. A survey of automatic query expansion in information retrieval. *ACM Computing Surveys (CSUR)*, 44(1):1, 2012.

Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002.

Nitesh V Chawla. Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*, pages 853–867. Springer, 2005.

Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *arXiv preprint arXiv:1106.1813*, 2011.

Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393, 1999.

Kevyn Collins-Thompson, Paul Ogilvie, Yi Zhang, and Jamie Callan. Information filtering, novelty detection, and named-page finding. In *TREC*, 2002.

Kevyn Collins-Thompson, Paul Bennett, Fernando Diaz, Charles LA Clarke, and Ellen Voorhees. Trec 2013 web track overview. In *22nd Text REtrieval Conference, Gaithersburg, Maryland*, 2014.

W Bruce Croft and John Lafferty. *Language modeling for information retrieval*, volume 13. Springer, 2003.

Hoa Trang Dang, Jimmy Lin, and Diane Kelly. Overview of the trec 2006 question answering track. In *Proceedings of TREC 2006*, 2007.

Adriel Dean-Hall, Charles LA Clarke, Jaap Kamps, Paul Thomas, Nicole Simone, and Ellen Voorhees. Overview of the trec 2013 contextual suggestion track. In *22nd Text REtrieval Conference, Gaithersburg, Maryland*, 2014.

Tamas E Doszkocs. Aid, an associative interactive dictionary for online searching. *Online Information Review*, 2(2):163–173, 1978.

Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156, 1996.

George W. Furnas, Thomas K. Landauer, Louis M. Gomez, and Susan T. Dumais. The vocabulary problem in human-system communication. *Communications of the ACM*, 30 (11):964–971, 1987.

Mikel Galar, Alberto Fernandez, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42(4):463–484, 2012.

D Graff. The aquaint corpus of english news text. philadelphia, pa, linguistic data consortium, 2002.

Donna Harman and Paul Over. The duc summarization evaluations. In *Proceedings of the second international conference on Human Language Technology Research*, pages 44–51. Morgan Kaufmann Publishers Inc., 2002.

David J Harper and Cornelis Joost Van Rijsbergen. An evaluation of feedback in document retrieval using co-occurrence data. *Journal of documentation*, 34(3):189–216, 1978.

Haibo He and Edwardo A Garcia. Learning from imbalanced data. *Knowledge and Data Engineering, IEEE Transactions on*, 21(9):1263–1284, 2009.

Monika Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 284–291. ACM, 2006.

Nathalie Japkowicz et al. Learning from imbalanced data sets: a comparison of various strategies. In *AAAI workshop on learning from imbalanced data sets*, volume 68. Menlo Park, CA, 2000.

Jiwoon Jeon, W Bruce Croft, and Joon Ho Lee. Finding similar questions in large question and answer archives. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 84–90. ACM, 2005.

Makoto P Kato, Matthew Ekstrand-Abueg, Virgil Pavlu, Tetsuya Sakai, Takehiro Yamamoto, and Mayu Iwata. Overview of the ntcir-10 1click-2 task. In *Proceedings of the 10th NTCIR Conference*, 2013.

Myoung-Cheol Kim and Key-Sun Choi. A comparison of collocation-based similarity measures in query expansion. *Information processing & management*, 35(1):19–30, 1999.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, pages 79–86, 1951.

Victor Lavrenko and W Bruce Croft. Relevance based language models. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 120–127. ACM, 2001.

Chin-Yew Lin and Eduard Hovy. Automatic evaluation of summaries using n-gram co-occurrence statistics. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 71–78. Association for Computational Linguistics, 2003.

Dekang Lin. Automatic retrieval and clustering of similar words. In *Proceedings of the 17th international conference on Computational linguistics-Volume 2*, pages 768–774. Association for Computational Linguistics, 1998.

Jimmy Lin and Dina Demner-Fushman. Automatically evaluating answers to definition questions. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 931–938. Association for Computational Linguistics, 2005.

Jimmy Lin and Dina Demner-Fushman. Will pyramids built of nuggets topple over? In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 383–390. Association for Computational Linguistics, 2006.

Rushi Longadge and Snehalata Dongre. Class imbalance problem in data mining review. *arXiv preprint arXiv:1305.1707*, 2013.

Rila Mandala, Tokunaga Takenobu, and Tanaka Hozumi. The use of wordnet in information retrieval. In *Use of WordNet in Natural Language Processing Systems: Proceedings of the Conference*, pages 31–37, 1998.

Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th international conference on World Wide Web*, pages 141–150. ACM, 2007.

Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

Gregory Marton and Alexey Radul. Nuggeteer: Automatic nugget-based evaluation using descriptions and judgements. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 375–382. Association for Computational Linguistics, 2006.

Andreas Merkel and Dietrich Klakow. Comparing improved language models for sentence retrieval in question answering. In *Computational Linguistics in the Netherlands Conference Proceedings*, 2007.

Donald Metzler, Yaniv Bernstein, W Bruce Croft, Alistair Moffat, and Justin Zobel. Similarity measures for tracking information flow. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 517–524. ACM, 2005.

George A. Miller. Wordnet: A lexical database for english. *COMMUNICATIONS OF THE ACM*, 38:39–41, 1995.

V García JS Sánchez RA Mollineda and R Alejo JM Sotoca. The class imbalance problem in pattern classification and learning. 2007.

Saeedeh Momtazi, Matthew Lease, and Dietrich Klakow. Effective term weighting for sentence retrieval. In *Research and Advanced Technology for Digital Libraries*, pages 482–485. Springer, 2010.

Vanessa Graham Murdock. *Aspects of sentence retrieval*. PhD thesis, University of Massachusetts Amherst, 2006.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

Laurence AF Park and Kotagiri Ramamohanarao. Query expansion using a collection dependent probabilistic latent semantic thesaurus. In *Advances in Knowledge Discovery and Data Mining*, pages 224–235. Springer, 2007.

Marius Pasca and Sanda Harabagiu. The informative role of wordnet in open-domain question answering. In *Proceedings of NAACL-01 Workshop on WordNet and Other Lexical Resources*, pages 138–143, 2001.

Jay M Ponte and W Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281. ACM, 1998.

Martin F Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137, 1980.

Yonggang Qiu and Hans-Peter Frei. Concept based query expansion. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 160–169. ACM, 1993.

S.E. Robertson, S. Walker, S. Jones, M.M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. pages 109–126, 1996.

Stephen E Robertson. The probability ranking principle in ir. *Journal of documentation*, 33(4):294–304, 1977.

Stephen E Robertson and Steve Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 232–241. Springer-Verlag New York, Inc., 1994.

Joseph John Rocchio. Relevance feedback in information retrieval. 1971.

Tetsuya Sakai and Hideo Joho. Overview of ntcir-9. In *Proceedings of NTCIR-9 Workshop*, pages 1–7, 2011.

Tetsuya Sakai, Makoto P Kato, and Young-In Song. Click the search button and be happy: Evaluating direct and immediate information access. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 621–630. ACM, 2011.

Gerard Salton and C Buckley. The smart information retrieval system, 1971.

Barry Schiffman. Experiments in novelty detection at columbia university. In *TREC*, 2002.

Hinrich Schütze and Jan O Pedersen. A cooccurrence-based thesaurus and two applications to information retrieval. *Information Processing & Management*, 33(3):307–318, 1997.

Chris Seiffert, Taghi M Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. Resampling or reweighting: a comparison of boosting implementations. In *Tools with Artificial Intelligence, 2008. ICTAI'08. 20th IEEE International Conference on*, volume 1, pages 445–451. IEEE, 2008.

Mark D Smucker and Charles LA Clarke. Time-based calibration of effectiveness measures. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 95–104. ACM, 2012.

Mark D Smucker, James Allan, and Ben Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 623–632. ACM, 2007.

Mark D Smucker, Charles L Clarke, and Gordon V Cormack. Experiments with clueweb09: Relevance feedback and web tracks. Technical report, DTIC Document, 2009.

Jerzy Stefanowski and Szymon Wilk. Selective pre-processing of imbalanced data for improving classification performance. In *Data Warehousing and Knowledge Discovery*, pages 283–292. Springer, 2008.

Olga Vechtomova. Query expansion for information retrieval. In *Encyclopedia of Database Systems*, pages 2254–2257. Springer, 2009.

Olga Vechtomova. A semi-supervised approach to extracting multiword entity names from user reviews. In *Proceedings of the 1st Joint International Workshop on Entity-Oriented and Semantic Search*, page 2. ACM, 2012.

Olga Vechtomova and Stephen E Robertson. A domain-independent approach to finding related entities. *Information Processing & Management*, 48(4):654–670, 2012.

Ellen M Voorhees. Query expansion using lexical-semantic relations. In *SIGIR'94*, pages 61–69. Springer, 1994.

Ellen M Voorhees. Question answering in trec. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 535–537. ACM, 2001.

Ellen M. Voorhees. Overview of the trec 2003 question answering track. In *Proceedings of TREC 2003*, 2004.

Ellen M. Voorhees. Overview of the trec 2004 question answering track. In *Proceedings of TREC 2004*, 2005.

Ellen M. Voorhees. Overview of the trec 2005 question answering track. In *Proceedings of TREC 2005*, 2006.

Ellen M Voorhees and Dawn M Tice. The trec-8 question answering track evaluation. In *TREC*, 1999.

Julie Weeds and David Weir. A general framework for distributional similarity. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 81–88. Association for Computational Linguistics, 2003.

Jinxi Xu and W Bruce Croft. Query expansion using local and global document analysis. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 4–11. ACM, 1996.

ChengXiang Zhai and John Lafferty. Two-stage language models for information retrieval. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 49–56. ACM, 2002.

Min Zhang, Ruihua Song, Chuan Lin, Shaoping Ma, Zhe Jiang, Yijiang Jin, Yiqun Liu, Le Zhao, and S Ma. Expansion-based technologies in finding relevant and new information: Thu trec 2002: Novelty track experiments. *NIST SPECIAL PUBLICATION SP*, (251):586–590, 2003.