SeqCluSum: Combining Sequential Clustering and Contextual Importance Measuring to Summarize Developing Events over Time

Markus Zopf
Research Training Group AIPHES / Knowledge Engineering Group
Department of Computer Science, Technische Universität Darmstadt
Hochschulstraße 10, 64293 Darmstadt, Germany
zopf@aiphes.tu-darmstadt.de

Abstract

Unexpected events such as accidents, natural disasters and terrorist attacks represent an information situation where it is essential to give users access to important and non-redundant information as fast as possible. In this paper, we introduce SeqCluSum, a temporal summarization system which combines sequential clustering to cluster sentences and a contextual importance measurement to weight the created clusters and thereby to identify important sentences. We participated with this system in the TREC Temporal Summarization track where systems have to generate extractive summaries for developing events by publishing sentence-length updates extracted from web documents. Results show that our approach is very well suited for this task by achieving best results. We furthermore point out several improvement possibilities to show how the system can further be enhanced.

1 Introduction

Events like accidents, natural disasters and terrorist attacks provide a important information situation. Shortly after the event occurred, the information situation is usually unclear. There might be some first vague information available, for example that an earthquake had occurred, but details like the magnitude, the epicenter, and if a tsunami has to be expected are not known at this early point in time. More and more information will become available later when details about the event are published by the media.

Traditional summarization approaches fail due to this unique characteristic of these information access problems. But especially during such crisis events the affected people urgently need information. The fastest information channel for information is the internet, where information are distributed via news sites, blogs, and microblogging services. Unfortunately the internet is not only the fastest, but also a highly redundant, bulky, and unreliable information source. This makes it impossible for an end-user to monitor this stream of news to extract important information timely.

A system which aims to aid people in retrieving information out of the web site stream has to deal with several difficulties. Since there are *vast amounts of documents* in the web, a system has to be highly efficient in order to process the documents. A system has to filter out *unrelated documents*, since the majority of the documents appearing on the web in the investigated timespan will not be related to the event and can be considered as noise. But noisiness on the document level is not the only problem. Even relevant documents contain a lot of unimportant information like unrelated news articles, advertisements, and navigation elements. This *within-document noise* has to be removed as well. Since information about big events are usually repeated on different web sites several times, the system has to deal with a *highly redundant* information situation. Another critical aspect is the *trustworthiness* of the sources. Several news pages publish uncertain information and this tendency is even higher in weblogs and on microblogging sites. The last, but also one of the most crucial challenges during such events, is the *timely detection* of new information. Since the users urgently need information, new information has to be detected as fast as possible.

In this paper we present SeqCluSum, a system which is able to aid users to satisfy their information needs in the previously described situations. We mainly address the problems of within-document noise, redundancy, and timely detection. Efficiency is only partially considered, since we assume a situation where our system is applied

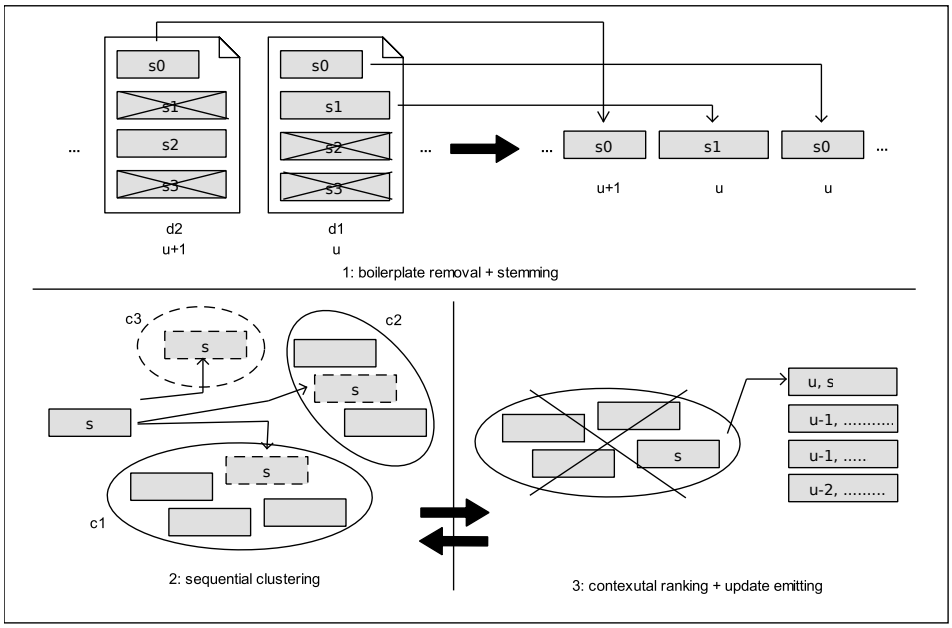

Figure 1: System overview. The timely-ordered documents $d_1, d_2, \ldots$ run through a boilerplate removal and stemming system in step 1. Only relevant sentences should pass this step. The documents are marked with timestamps $u, u+1, \ldots$. In step 2, a sentence s is either added to the existing clusters c_1 or c_2 or creates a new cluster depending on the similarity of s to c_1 and c_2. Step 3 illustrates the publishing of a sentence. The sentence is added to the summary with time marker u and the cluster is set to *published*.

after an information retrieval step where irrelevant documents has already been filtered out. Our system is, like in the most summarization scenarios, not concerned with trustworthiness. Therefore, we assume that all documents which are passed to our system can be considered as trustworthy.

An overview about the system is given in Figure 1. First, we use two preprocessing steps (Figure 1, step 1) which remove boilerplate content and stem all tokens. This step is further described in Section 3. Second, we use sequential clustering described in Section 4.1 to cluster similar sentences together (Figure 1, step 2). Last, we weight the clusters after processing a document according to their contextual importance and publish updates (Figure 1, step 3). This step is described in Section 4.2 in detail.

With this system, we participated in the Temporal Summarization track of the Twenty-Fourth Text REtrieval Conference (TREC). We describe the shared task in detail in Section 5 and our results in Section 5.4.

Due du time limitations, we did not push all components of our system to the limit. Therefore, we describe some improvement possibilities in Section 7 from which we expect a further improvement of performance.

2 Related Work

Early work on document summarization started with Luhn (1958), Baxendale (1958), and Edmundson (1969), who used word frequencies, position features, and key phrases to detect important sentences.

McKeown and Radev (1995) introduced the problem of multi-document summarization. Temporal summarization in the TREC-TS challenge is strongly related to extractive multi-document summarization (Nenkova and McKeown, 2011), since the systems are only allowed to extract sentences from the source documents verbatim. Radev et al. (2004) presented an open-source multi-document summarization system, MEAD, which is able to summarize large news topics by clustering sentences and finding centroids. Mihalcea and Tarau (2004) used a graph-based method to extract text for summarization. Carbonell and Goldstein (1998) introduced the greedy approach MMR to extract topic related and non-redundant sentences jointly.

For the particular task of temporal summarization, we review some of the participating systems from last years TREC-TS challenges. The best performing run in the challenge in 2014 was the "2APSal" run by team "cunlp" (Kedzie et al., 2014), who used affinity propagation clustering. Zhao et al. (2014) used a query expansion and information retrieval step with the Lemur toolkit[1] and a k-means clustering (Zhang et al., 1996) and sentence selection step. McCreadie et al. (2014) used a pipeline to filter out unrelevant documents, to classify sentence according to their relevance, and to filter out redundant sentence to build a real-time summarization system.

Our approach described in this paper combines the timely detection of a pipelining approach like McCreadie et al. (2014) with the redundancy avoidance strategy of a clustering approach like Zhao et al. (2014) by using a sequential clustering.

3 Preprocessing

In this section, we describe two important preprocessing steps which are used to prepare the input documents for the temporal summarization system. Both preprocessing steps, boilerplate removal and word stemming, are well-known and crucial for the performance of the subsequent summarization. Since they are self-contained systems and independent from the subsequent summarization system we did not investigate them in detail and expect further potential for improvement of the whole system if these steps are improved.

Documents retrieved from the web are usually very noisy since they do not only contain one article, but also advertisements, navigation elements, text snippets from other websites, author information, templates, pictures, links to related articles, etc. This noise was described in the introduction as *within-document noise* and is commonly called *boilerplate text*. Content which is unrelated to the target topic should never be included in the output of a summarization system. Therefore, the first preprocessing removes the within-document noise. To tackle this problem, we utilize the software Boilerpipe (Kohlschütter et al., 2010) which uses shallow text features to remove unrelated text fragments. This preprocessing step is crucial for a good performance of the summarization system described below since it publishes frequent information without any knowledge of the actual query. Since navigation elements and advertisements are very frequent on web sites the system would include a lot of this unimportant content into the summary.

In a second step, we stem all words in the source documents with the well-known Porter-stemmer (Porter, 1980). This step is assumed to help the system to detect semantically similar but syntactically different words. Stemming will become important in Section 4.1 and Section 4.2 when we cluster similar sentences to measure their importance and to detect redundancy. More abstract semantic methods like word embeddings (Mikolov et al., 2013) are expected to model the semantic similarity of words better than stemming since they can, for example, not only detect a similarity between the words *crash* and *crashed*, but also between *crash* and *accident*. Therefore, we assume that a better representation could further improve the performance of similarity and redundancy detection in the following summarization system.

4 Temporal Summarization with `SeqCluSum`

After explaining the preprocessing of the documents, we now describe the combination of two building blocks, a sequential clustering algorithm with a contextual importance measurement.

The first building block of the system, a sequential clustering algorithm, is responsible to detect redundant information by clustering similar sentences together. Since the second block will select at most one sentence per cluster, this will prevent the system to publish similar sentences (and therefore similar information) multiple times. The sequential clustering is described in Section 4.1.

To decide which sentences should be included in the summary, we apply in a second block a contextual importance measuring, which is further described in Section 4.2. This building block publishes sentences depending on the weight of the sentences, which itself depends on the content of the sentences as well as on already published sentences (therefore *contextual*), and the weight of the previously generated clusters.

The both blocks are applied for each document one after another and therefore work closely together to jointly solve the problem of timely detection of important information and reduction of redundancy. For convenience, we give pointers to the pseudo-code of the system in Algorithm 1.

[1] `http://www.lemurproject.org`

4.1 Sequential Clustering

The first building block of the temporal summarizer is a sequential clustering algorithm. We apply the Basic Sequential Algorithm Scheme (BSAS) (Theodoridis and Koutroumbas, 2009), which is used to iterate over all (unpruned) sentences in the currently processed document (Algorithm 1, line 5). The algorithm searches for the nearest existing cluster by using a similarity measure (line 7) for all sentences which were not pruned in the boilerplate removal step. We describe the distance measure below in more detail. If the similarity to all existing clusters is lower than a fixed threshold Θ and the maximum number of clusters I is not already reached, a new cluster is created and the sentence is added to the new cluster (line 9). Otherwise, the sentences is added to the nearest existing cluster (line 11).

The similarity measure calculates the similarity between a sentence s_i and a cluster C_j by calculating the similarity between the sentence s_i and the first sentence of the cluster C_{j_1}. By doing so, the sentences to cluster similarity measurement is reduced to a sentences to sentence similarity measurement. This has several advantages in comparison to considering all sentences in the cluster.

First, it emphasizes the notion of a cluster as a set of sentences which contains one distinct information. In a scenario where the similarity function works perfectly (i.e. equals 1 if the sentence matches the represented information by the cluster and equals 0 otherwise), there is no need to compare it to the other sentences. The result would be the very same for each sentence. We choose the first sentences of each cluster to be the representative of the cluster since each first sentence has a special role. This special role derives from the fact that the first sentence of each cluster was the reason why the cluster had been created. The sentences did not fit to another cluster and therefore created its own, new cluster.

Second, the center of the cluster is fixed when we compare only with the first sentence of the cluster. This prevents the cluster from a topic drift, which was a serious issue when we compared to more than one sentences. By adding more and more sentences and also using the newly added sentences for further distance calculations could instead change the initial notion of the cluster significantly since the center of the cluster would move.

Third, the approach is computational efficient in comparison to an approach, where we would compute the similarity by incorporating all sentence in a particular cluster.

The actual similarity measure metric (line 7) is a linear combination of various well-known similarity measures. We use five different n-gram Jaccard measures, for $n \in \{1, 2, 3, 4, 5\}$. The n-gram Jaccard measure is defined in Equation 1, where $\Pi_n(s)$ is the set of all n-grams in sentence s.

$$jaccard_n(s_1, s_2) = \frac{|\Pi_n(s_1) \cap \Pi_n(s_2)|}{|\Pi_n(s_1) \cup \Pi_n(s_2)|} \quad (1)$$

Furthermore, we use a longest common subsequence comparator. Additionally to the string comparison methods, we apply different variations of cosine similarities based on inverse term frequency (ITF) vectors. To calculate the ITF values, we use a small sample of 2,182 documents as background corpus B. The ITF value of a token t equals the logarithm of the quotient of the number of all tokens in the background corpus B, $|B|$ and the number of appearances of the token t in the background corpus B, $|t \in B|$ (Equation 2). This value measures how common a token is in a background corpus and measures therefore how much information a token provides in general.

$$ITF(t) = \log\left(\frac{|B|}{|t \in B|}\right) \quad (2)$$

All documents in the background corpus were retrieved from documents which were created before the event started to ensure that we do not use information which were not available at the time when the event happened.

We combine these different similarity measures into one similarity measure to cover different aspects of similarity. The overall similarity measure for two sentences s_1 and s_2 is a linear combination as shown in Equation 3, where $\mathcal{S}$ is a set containing all previously described similarity measures sim_i.

$$sim\,(s_1, s_2) = \frac{1}{|\mathcal{S}|} \cdot \sum_{sim_i \in \mathcal{S}} sim_i \quad (3)$$

Some values for different similarity measures are listed in Table 1. We show values of the 2-gram and 3-gram Jaccard measure, two different cosine similarity values, and the value of the longest common subsequence comparator. All measures are

Measure	$s(s_1, s_2)$	$s(s_1, s_3)$	$s(s_1, s_4)$
2-gram J	0.097	0.400	0.118
3-gram J	0.032	0.269	0.059
cosine 1	1.000	0.078	0.024
cosine 2	1.000	0.814	0.290
LCSC	0.613	0.821	0.490
mean	0.696	0.397	0.148

Table 1: Different similarity measures

implemented in the software DKPro Similarity[2], described in Bär et al. (2013). In the example, sentence s_2 is a permutation of sentence s_1, sentence s_3 was clustered to the same cluster as sentence s_1, and sentence s_4 was assigned to a different cluster than sentence s_1. We see that the cosine similarity is not affected by the ordering of the words in comparison to the n-gram Jaccard measure. Therefore, the cosine similarity properly detects when two sentences are using the same words, which signals that their content is similar. Nevertheless, incorporating the Jaccard measures gives additional insights to the similarity of two sentences, since the ordering of the words can be considered as an even stronger signal for similarity.

4.2 Measuring Contextual Importance

After all unpruned sentences in a document are clustered with the previously described sequential clustering algorithm, the system evaluates the current cluster landscape to detect important information in the document stream. To achieve this, all clusters are weighted using a weight measures based on a TF*ITF values. The score of a cluster C_i equals the sum of the weights of all sentences C_{i_j} in the cluster (Equation 4).

$$weight(C_i) = \sum_{C_{i_j} \in C_i} weight(C_{i_j}) \quad (4)$$

Summarizing over all sentences in a cluster addresses the assumed property of the document stream that more important information is repeated more frequently in the source documents since bigger clusters get higher weight scores.

The weight of a sentence s_i is defined as the sum of the weights of the tokens s_{i_j} contained in sentence s_i (Equation 5).

$$weight(s_i) = \sum_{s_{i_j} \in s_i} weight(s_{i_j}) \quad (5)$$

In the following we define a context-free and two contextual metrics to measure the weight of a single token. The context-free metric measures how important a token is. But since we actually want to know if we should publish a sentences or not, we have to consider already published sentences as well as the importance of the sentences because we don't want to publish an important sentence when a similar sentence has already been published. We assume that certain tokens are responsible for covering a particular information nugget. Therefore, the contextual token weights incorporate the already published tokens in the computation.

The context-free weight of a token $weight_{cf}$ is computed based on the temporal TF*ITF-based measures of the token (Equation 6).

$$weight_{cf}(t, D_\tau) = TF_{cf}(t, D_\tau) \cdot ITF(t) \quad (6)$$

The computation of the ITF value is described in Section 4.1. We use the ITF value not to measure if a document is relevant to a topic (like the commonly used IDF value does), but to calculate if a token is important in a stream of documents. Since we have no access to all documents in the stream, but only the documents which were published until a given timestamp τ, we can only compute the temporal TF values for the set D_τ. We measure therefore, how salient a token i is in a document stream D_τ until the timestamp τ. To do so, we count the number of appearances of token t in the documents D_τ and divide this quantity by the total number of tokens in the documents $|D_\tau|$. This gives us an impression on how salient a token is in the document collection D_τ. Equation 7 formalizes the computation of the context-free token weights.

$$TF_{cf}(t, D_\tau) = \frac{|t \in D_\tau|}{|D_\tau|} \quad (7)$$

Since the document collection D_τ changes after each time step τ, the temporal TF values are constantly updated when new documents are processed.

The contextual weight of a token models the weight of a token with respect to already published tokens. We reduce the context-free TF values for

[2] https://dkpro.github.io/dkpro-similarity

Algorithm 1 `SeqCluSum`

$documents$ = ordered list of documents, I = maximum number of clusters

1: $clusters \leftarrow \{\}$, $updates \leftarrow \{\}$
2: **for each** $document \in documents$ **do**
3: $document \leftarrow$ pruneBoilerplate $(document)$
4: $document \leftarrow$ stemWords $(document)$
5: **for each** $sentence \in document$ **do** ▷ add new sentences to clusters
6: **if** $sentence$ is not pruned **then**
7: $nearestCluster =$ argmax$_{c \in clusters}$similarity$(sentence, c)$
8: **if** $i = 0$ **or** (similarity$(sentence, nearestCluster) > \Theta$ **and** $i < I$) **then**
9: $clusters \leftarrow clusters \cup \{sentence\}$; $i \leftarrow i + 1$ ▷ create new cluster
10: **else**
11: $nearestCluster \leftarrow nearestCluster \cup sentence$ ▷ add to nearest cluster
12: **end if**
13: **end if**
14: **end for**
15: **for each** $cluster \in clusters$ **do** ▷ evaluate clusters and generate updates
16: **if** $cluster$ is not published **and** weight$(cluster) > \mu$ **then**
17: $bestSentence =$ argmax$_{c \in cluster}$weight(s) ▷ find best sentence in $cluster$
18: $updates \leftarrow updates \cup (document.timestamp, bestSentence)$
19: set $cluster$ to *published*
20: **end if**
21: **end for**
22: **end for**
23: **return** $Updates$

tokens which have already been published. By implementing such a reduction of already published tokens, we achieve a contextual importance grading of the tokens which leads to an avoidance of redundancy.

We use two different strategies for the reduction of the TF_{cf} values. In the first contextual strategy, cs_1, the values of all already published tokens are set to 0, which leads to a very strict and extensive avoidance of redundancy. In the second contextual strategy, cs_2, we reduced the values by dividing the context-free value $TF_{cf}(t, D_\tau)$ by the number of appearances of token t in the already published updates. This leads to a more laxly filtering which is expected to increase recall at the cost of precision. The computation of the second contextual importance measure is described in Equation 8, where $|t \in U|$ denotes the number of appearances of the token t in the current set of updates U.

$$TF_{cs_2}(t, D_\tau) = \frac{TF_{cf}(t, D_\tau)}{|t \in U|} \tag{8}$$

A cluster is considered as sufficiently important if a fixed threshold μ is exceeded (line 16). This threshold can be varied in a productive environment easily, to produce more or less verbose summaries. For the experiments, we choose two values by hand without any sophisticated evaluation or optimization method. If a cluster exceeds the threshold, the best sentence according to the sentence score described above is published. The cluster is marked as *published* in this case. This means, that the cluster will not be selected in the future to publish another sentence. Nevertheless, it is still possible to add new sentences to an published cluster. This is important because we will likely see more sentences, which are similar to already published sentences.

5 TREC Temporal Summarization Track

We participated with the previously described system, `SeqCluSum`, in the TREC 2015 Temporal Summarization track. In the following section we give detailed information about the challenge, describe the evaluation methodology and discuss our performance in the task.

5.1 Introduction

We described in Section 1 the task of temporal summarization and outlined its importance and special properties in comparison to classical multi-document summarization. In this section, we describe the Temporal Summarization track (Guo et al., 2013) of the Text REtrival Conference 2015 (TREC), or TREC-TS 2015 for short. The goal of this track is to develop systems which can detect useful, new, and timely sentence-length updates about a developing event.

The first major difference to classical summarization challenges like DUC[3] and TAC[4] is the notion of time during summarization. In TREC-TS it is not only crucial to detect and extract important information, but to detect the important information as early as possible during the developing of an event.

A second important difference compared to DUC is the evaluation methodology. In DUC, gold standard summaries were written by human experts and the automatically generated summaries are compared to these summaries with the ROUGE (Lin, 2004) score. In contrast, summaries produced by the TREC-TS participant systems have to cover as many as possible fixed information nuggets. If a sentence contains an information is evaluated by hand for each sentence. Furthermore, it is, compared to DUC and TAC, solely focused on information retrieval and not on writing quality.

5.2 Setup

The setup of the TREC-TS track is as follows. There are 3 subtasks in the track. Each subtask consists of 21 events. The first two subtasks, *Filtering and Summarization*, deal with high-volume streams of news article documents and blog posts which do not have to be related to the given event. In this scenario, the participants have to filter out irrelevant documents to be able to summarize the rest of the stream. In the third subtask, *Summarization only*, the participants face low-volume streams of pre-filtered documents. The corresponding corpus to this subtask is called TREC-TS-2015F-RelOnly[5]. In this scenario, the information retrieval problem is considered to be solved on the document level. This means that all documents in the stream contain relevant information. We participated with our system in the latter subtask.

All documents in a stream have an associated timestamp which equals the crawling time of the document. The documents have to be processed in temporal order. The result of the system is a list of updates. These updates are considered to be sentence-length update messages, which could be published via a microblogging service like Twitter[6] to keep an end-user up-to-date about the development of an event. When a system decides that a sentence from the stream should be published, this particular sentence is marked with the timestamp of the currently processed document. The systems are not allowed to use *information from the future* to make this decision. This means, incorporating information that was only available after the current timestamp is not allowed. One example for a not allowed usage would be the computation of TF*ITF values over the whole corpus since this would provide information about which words will become frequent in the future.

5.3 Evaluation

The evaluation of the submissions was based on Wikipedia article revision histories of the corresponding events. In a first step, information nuggets were extracted from the revision histories by the track organizers. The nuggets were marked with a timestamp and classified with an importance score of 1, 2, or 3, where 1 is the least and 3 the most important category. This timestamp equals the time where the information was included in the Wikipedia article the first time. Therefore, it represents the time when the information could be considered to be publicly available. An example of such a nugget is *VM26.002-1358344449-3-'On 16 January 2013,at around 0800 GMT'*, where *VM26.002* is the nugget id, *1358344449* is the timestamp[7] when the information has been published in Wikipedia, *3* is the importance of the nugget, and *'On 16 January 2013,at around 0800 GMT'* is the actually information encoded by this nugget.

In a second step, sentences were pooled from each submission. For each submission, at most 60 sentences were pooled due to the huge annotation effort. These sentences were manually

[3] http://duc.nist.gov

[4] http://www.nist.gov/tac

[5] http://dcs.gla.ac.uk/~richardm/TREC-TS-2015RelOnly.aws.list

[6] https://twitter.com

[7] The timestampes equal to the number of seconds since January 1, 1970

matched against the extracted information nuggets since there is no system available which is able to find these matchings precisely. A sentence could thereby match multiple nuggets as well as no nugget. A sentence, which matches relevant information nuggets, which were not matched by sentences published earlier by a submission, was considered to be a valuable sentence. If a sentence matches only already matched information nuggets or no nuggets at all, it was considered to be worthless. Since such sentences do not improve the information content of the summary, the system got penalty points for being to verbose.

For the evaluation, the precision of the generated summaries was evaluated with the (normalized) Expected Gain metric $n\text{EG}(\mathcal{S})$. This metric measured to which degree the sentences in the summary were on-topic and novel. The recall was measured by the Comprehensiveness metric $\text{C}(\mathcal{S})$. This metric measured how many of the information nuggets that could have been retrieved were covered by the summary. A third measure, the Expected Latency E[Latency], measured to which degree the information in the summary was outdated. A combined F-measure, $\mathcal{H}$, based on the precision and recall which also incorporates the latency was used as official target measure.

5.4 Results

We submitted four runs for evaluation at the TREC-TS shared task. Runs 1 and 2 used the strict contextual strategy cs_1. The runs 3 and 4 used the more laxly contextual strategy cs_2. Runs 1, 2 and 4 had a maximum number of clusters of 1,000 whereas run 3 had a maximum of 100. The minimum cluster score, which had to be obtained by the cluster to be published was set to 1 for the runs 1, 3, and 4. For run 2, this value was set to 3. Furthermore, we modified the boilerplate removal for runs 3 and 4 slightly to prune less sentences during the preprocessing. The similarity function used was the very same for all four runs. These changes led to differently verbose runs, where run 1 was the least verbose run followed by run 2. Run 3 and run 4 were the runs which published most updates due to the more laxly contextual strategy.

Unfortunately, we introduced errors in the result files which resulted in invalid sentence identifiers. Therefore, our sentences could not be included in the pool of sentences which were matched against the information nuggets extracted from Wikipedia as described in Section 5.3. Since other systems selected at least some of the sentences, which we had selected, we were at least able to get some points for the overlapping sentences. Since we did not get points for sentences which were only selected by our system, this score represents only a lower bound for the true performance of the system. We report the percentage of unevaluated sentences in column *unevaluated*. The sentences which were not evaluated were considered as irrelevant or redundant without looking at them. As example, for run 1 only 41.59% of the top 60 updates were published by other systems as well and therefore only this amount was evaluated. The remaining 58.41% were considered to be irrelevant or redundant and we were penalized for them.

Nevertheless, run 1 and run 2 achieved considerable better results than the best other participating system. Run 3 and run 4 performed better than the second best system.

Table 2 shows the described lower bounds for our runs, a comparison with the best 3 systems and the average grading over all runs sorted by the main target metric $\mathcal{H}$. Additionally, we provide the percentage of unevaluated sentences to make the results better assessable. The evaluation results of the other systems are taken from the overview paper of the TREC-TS challenge.

6 Conclusions

We presented `SeqCluSum`, a system for temporal summarization based on sequential clustering and contextual importance measurement. The sequential clustering arranges similar sentences together to detect redundancy. The contextual importance measure weights the clusters according to their importance. During this step, we incorporate the knowledge about previously published sentences to avoid redundancy.

With this system, we participated in the *Summarization only* subtask of the Temporal Summarization track at the Twenty-Fourth Text REtrieval Conference. Unfortunately, we introduced errors in our submission and can only report lower bounds of the performance of our runs. Nevertheless, these lower bounds show already a better performance than the best fellow competitor run. Therefore, we conclude that our approach is well suited to handle the special difficulties in this challenge.

TeamID	RunID	$\mathcal{H}$	unevaluated	$n\mathbf{EG}(\mathcal{S})$	$\mathbf{C}(\mathcal{S})$	E[Latency]
AIPHES	Run1	0.1083	58.41%	0.1576	0.2854	0.4776
AIPHES	Run2	0.1016	59.97%	0.1260	0.2982	0.4607
WaterlooClark	UWCTSRun4	0.0853	-	0.1840	0.1710	0.3983
AIPHES	Run4	0.0781	63.34%	0.0897	0.4005	0.5679
AIPHES	Run3	0.0722	62.64%	0.0905	0.4164	0.5061
BJUT	DMSL2N2	0.0649	-	0.0645	0.6557	0.5606
uogTr	uogTrhEQR2	0.0639	-	0.0667	0.5459	0.5335
Mean	-	0.0472	-	0.0595	0.5627	0.5524

Table 2: Lower bound evaluation values for AIPHES runs 1-4, the 3 best non-AIPHES runs and average scores of all participating runs sorted descending by $\mathcal{H}$, the main metric of the challenge. The colums $n\mathbf{EG}(\mathcal{S})$, $\mathbf{C}(\mathcal{S})$, and E[Latency] show the results according to official evaluation metrics described in Section 5.3. The column *unevaluated* shows how many percent of the sentences were unevaluated.

7 Future Work

We propose in the following several possibilities to further enhance the system in order to improve the performance of our system and subsystems which were not considered due to time limitations.

The sequential clustering depends strongly on the accuracy of the used similarity measure. Since we used predefined standard similarity measures without any fine-tuning or adaption to the task, we assume that a more sophisticated selection of similarity measures could lead to an improvement of the performance.

As described in Section 3, a semantic representation of the text could also help to detect similarities between sentences more accurately.

Another improvement of the similarity measure would be to utilize the weight of the tokens, which are used during the contextual measuring of importance also in the clustering. If we weight important tokens higher in the similarity metrics, we could achieve that the clusters would be more focused on a particular information nugget. For example, the token *15* should be more influential to the clustering when there were 15 people injured in an event and the token *the* should not have a high influence on the similarity of sentences. Generally speaking, a sentence $A = (a, x, a, a, a)$ should be more similar to the sentence $B = (b, b, b, x, b)$ than to sentence $C = (a, a, a, a, a)$ when x is an important token in the current topic and a and b are two unimportant tokens, although the tokens a and b are more frequent in the sentences than token x.

The system described in this paper relies on the assumption that important information is mentioned frequently. This is in particular problematic for temporal summarization, because it takes some time (or documents) to detect that a word is irregularly frequent in the corpus. For example, during the first mentioning of the word *bomb* the word is not considered to be important for the corpus, because it occurred only once. Only when more documents are processed that contain the token the system is able to detect that this token is important. If we would leverage the fact that the mentioning of *bomb* is always important when summarizing in the domain of terrorist attacks, we would be able to detect the importance faster. For *bomb* this improvement can be rather easily achieved, because when a bomb explodes this is nearly always important. But the target of the bomb will also be very important. However, it is nearly impossible to know beforehand whether the word *cab*, *Quetta*, or *main station* will be important.

Furthermore, it should be possible to add a sentence to multiple clusters because a sentence can contain multiple information nuggets. The general idea of the clustering is that each particular information nugget is represented by exactly one cluster. In the current system, a sentence can only be part of one cluster.

Another improvement concerning the clustering could be a re-cluster step, which could split a cluster into multiple clusters when the system detects that one cluster contains too diverse sentences. This is currently prevented by the parameter Θ. By enabling the system to execute a re-clustering step, this parameter could become needless. The same holds for the upper bound of clusters. Furthermore, the system could merge multiple clusters to one cluster, if it detects that the content of the clusters are more similar as the seed sentences

suggested. This situation can easily occur when synonyms are used.

Acknowledgments

This work has been supported by the German Research Foundation (DFG) as part of the Research Training Group Adaptive Preparation of Information from Heterogeneous Sources (AIPHES) under grant No. GRK 1994/1.

References

Daniel Bär, Torsten Zesch, and Iryna Gurevych. 2013. Dkpro similarity: An open source framework for text similarity. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 121–126, Sofia, Bulgaria, August. Association for Computational Linguistics.

Phyllis B Baxendale. 1958. Machine-made index for technical literature: an experiment. *IBM Journal of Research and Development*, 2(4):354–361.

Jaime Carbonell and Jade Goldstein. 1998. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336. ACM.

Harold P Edmundson. 1969. New methods in automatic extracting. *Journal of the ACM (JACM)*, 16(2):264–285.

Qi Guo, Fernando Diaz, and Elad Yom-Tov. 2013. Updating users about time critical events. In Pavel Serdyukov, Pavel Braslavski, SergeiO. Kuznetsov, Jaap Kamps, Stefan Rüger, Eugene Agichtein, Ilya Segalovich, and Emine Yilmaz, editors, *Advances in Information Retrieval*, volume 7814 of *Lecture Notes in Computer Science*, pages 483–494. Springer Berlin Heidelberg.

Chris Kedzie, Kathleen McKeown, and Fernando Diaz. 2014. Summarizing disasters over time. In *Proceedings of the Twenty-Third Text REtrieval Conference*.

Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. 2010. Boilerplate detection using shallow text features. In *Proceedings of the 3rd ACM International Conference on Web Search and Data Mining*, pages 441–450.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Proceedings of the ACL Text Summarization Workshop*, pages 74–81.

Hans Peter Luhn. 1958. The automatic creation of literature abstracts. *IBM Journal of Research and Development*, 2(2):159–165.

Richard McCreadie, Romain Deveaud, M-Dyaa Albakour, Stuart Mackie, Nut Limsopatham, Craig Macdonald, Iadh Ounis, Thibaut Thonet, and Bekir Taner Dinçer. 2014. University of glasgow at trec 2014: Experiments with terrier in contextual suggestion, temporal summarisation and web tracks. In *Proceedings of the Twenty-Third Text REtrieval Conference*.

Kathleen McKeown and Dragomir R Radev. 1995. Generating summaries of multiple news articles. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 74–82. ACM.

Rada Mihalcea and Paul Tarau. 2004. Textrank: Bringing order into texts. Association for Computational Linguistics.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Ani Nenkova and Kathleen McKeown. 2011. Automatic summarization. *Foundations and Trends in Information Retrieval*, 5(2-3):103–233.

Martin Porter. 1980. An algorithm for suffix stripping. *Program*, 14(3):130–137.

Dragomir R Radev, Hongyan Jing, Małgorzata Styś, and Daniel Tam. 2004. Centroid-based summarization of multiple documents. *Information Processing & Management*, 40(6):919–938.

Sergios Theodoridis and Konstantinos Koutroumbas, 2009. *Pattern Recognition*, chapter 12, pages 633–634. Elsevier Ltd, Oxford.

Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. Birch: an efficient data clustering method for very large databases. In *ACM SIGMOD Record*, volume 25, pages 103–114. ACM.

Yun Zhao, Fei Yao, Huayang Sun, and Zhen Yang. 2014. Bjut at trec 2014 temporal summarization track. In *Proceedings of the Twenty-Third Text REtrieval Conference*.