TREC 2013 Temporal Summarization

Javed Aslam Fernando Diaz Matthew Ekstrand-Abueg
Virgi Pavlu Tetsuya Sakai

November 12, 2013

1 Introduction

Unexpected news events such as earthquakes or natural disasters represent a unique information access problem where traditional approaches fail. For example, immediately after an event, the corpus may be sparsely populated with relevant content. Even when, after a few hours, relevant content is available, it is often inaccurate or highly redundant. At the same time, crisis events demonstrate a scenario where users urgently need information, especially if they are directly affected by the event.

The goal of this track is to develop systems for efficiently monitoring the information associated with an event over time. Specifically, we are interested in developing systems which (1) can broadcast short, relevant, and reliable sentence-length updates about a developing event and (2) can track the value of important event-related attributes (e.g. number of fatalities).

The track has the following goals,

- to develop algorithms which detect sub-events with low latency,
- to model information reliability in the presence of a dynamic corpus,
- to understand and address the sensitivity of text summarization algorithms in an online, sequential setting, and
- to understand and address the sensitivity of information extraction algorithms in dynamic settings.

2 Task Descriptions

2.1 Sequential Update Summarization

During the simulation, a system should emit relevant and novel sentences to an event (exact metrics will be released in a separate document). Conceptually, a simulator should be structured as in Figure 1. The arguments to the simulator are the participant summarization system, the time-ordered corpus, the keyword

```
TemporalSummarization($\mathbf{S}, \mathcal{C}, q, t_s, t_e$)
    $\mathbf{S}$      ▷ Participant system.
    $\mathcal{C}$      ▷ Time-ordered corpus.
    $q$      ▷ Event keyword query.
    $t_s$     ▷ Event start time.
    $t_e$     ▷ Event end time.
 1  $\mathcal{U} \leftarrow \{\}$
 2  $\mathbf{S}$.Initialize($q$)
 3  for $d \in \mathcal{C}$
 4      do
 5          $\mathbf{S}$.Process($d$)
 6          $t \leftarrow d$.Time()
 7          if $t \in [t_s, t_e]$
 8            then
 9                $\mathcal{U}_t \leftarrow \mathbf{S}$.Decide()
10                for $u \in \mathcal{U}_t$
11                    do
12                        $\mathcal{U}$.Append($u, t$)
13  return $\mathcal{U}$
```

Figure 1: Sequential update summarization simulator.

query, and the relevant time range. In line 1, we initialize the output summary to empty. In line 2, we initialize the sequential update summarization system with the event query. The system should store some representation of this query for later processing and filtering. We iterate over the corpus in temporal order (line 3), processing each document in sequence (line 5). If the document we are processing is in the event timeframe (line 7), then we check to see if adding the document resulted in the system deciding to output a set of summary sentence ids (line 9). We then add these sentence ids to the summary timestamped with the time of the decision (lines 10-12).

We have tried to present an abstract representation of sequential update summarization. There are several comments worth making. First, if a participant is interested in efficiency and does not anticipate needing documents outside of the event timeframe, then the call to $\mathbf{S}$.Process(d) can be moved inside of the condition in line 7. Participants should be clear about any filtering of $\mathcal{C}$. For example, a participant should note if they are just iterating over documents with a high BM25 score. However, if this is done, care must be taken to make sure that filtering out a document d does not exploit information from sources after d.Time() (e.g. retrospective IDF values).

```
ValueTracking($\mathbf{S}, \mathcal{C}, q, a, t_s, t_e$)
     $\mathbf{S}$      $\triangleright$ Participant system.
     $\mathcal{C}$      $\triangleright$ Time-ordered corpus.
     $q$      $\triangleright$ Event keyword query.
     $a$      $\triangleright$ Event attribute.
     $t_s$     $\triangleright$ Event start time.
     $t_e$     $\triangleright$ Event end time.
 1   $\mathcal{V} \leftarrow \{\}$
 2   $\mathbf{S}$.Initialize($q, a$)
 3   $v \leftarrow \mathbf{S}$.InitialEstimate()
 4   $\mathcal{V}$.Append($\langle v, \emptyset \rangle, t_s$)
 5   for $d \in \mathcal{C}$
 6       do
 7          $\mathbf{S}$.Process($d$)
 8          $t \leftarrow d$.Time()
 9          if $t \in [t_s, t_e]$
10            then
11                 $\langle v, s \rangle \leftarrow \mathbf{S}$.EstimateValue()
12                 if $\langle v, s \rangle \neq \emptyset$
13                   then
14                        $\mathcal{V}$.Append($\langle v, s \rangle, t$)
15   return $\mathcal{V}$
```

Figure 2: Value tracking simulator.

2.2 Value Tracking

During the simulation, a system should emit accurate attribute value estimates for an event. Conceptually, a simulator should be structured as in Figure 2. The arguments to the simulator are the participant tracking system, the time-ordered corpus, the keyword query, the attribute of interest, and the relevant time range. In line 1, we initialize the value summary to empty. In line 2, we initialize the tracking system with the event query and attribute name. The system should store some representation of this information for later processing and filtering. We then ask for an initial estimate based solely on the query and attribute. Systems are welcome to use any data that existed at or before t_s. We iterate over the corpus in temporal order (line 5), processing each document in sequence (line 7). If the document we are processing is in the event timeframe (line 9), then we check to see if adding the document resulted in a change of the system's value estimate (line 11). If there is no change in the value estimate, the system should return $\emptyset$ and continue processing documents. If the estimate changes, the system should return the new value as well as the id of the supporting sentence.

3 Evaluation

3.1 Temporal Summarization

Each event will be retrospectively analyzed for important sub-events or 'nuggets', each with a precise timestamp and text describing the sub-event. Our evaluation metrics will measure the degree to which a system can generate these nuggets in a timely manner.

A system/run *update* is a timestamped short text string comparable in length to a sentence. Colloquially, an update can be thought of as an SMS message or tweet. We generally denote an update as the pair (string, time): $u = (u.string, u.t)$. For example $u =$ ("`The hurricane was upgraded to category 4`", 1330169580) represents an update describing the hurricane category, now 4, pushed out by system $\mathcal{S}$ at UNIX time 1330169580 (i.e. 1330169580 seconds after 0:00 UTC on January 1, 1970). In this year's evaluation, the update string is chosen from the set of segmented sentences in the corpus as defined in the guidelines.

Two updates are semantically comparable using a text similarity measure or a manual annotation process applied to their *string* components; if two updates u and u' refer to the same information (semantically matching), then we write this as $u \approx u'$, irrespective of their timestamps. Because two systems might deliver the same update *string* at different times, it is generally not the case that $u.string = u'.string$ implies $u.t = u'.t$.

Given an event, our manual annotation process generates a set of gold standard updates called *nuggets*, extracted from wikipedia event pages and timestamped according to the revision history of the page. Editorial guidelines recommend that nuggets be a very short sentence, including only a single sub-event, fact, location, date, etc, associated with topic relevance. We refer to the canonical set of updates as $\mathcal{N}$. This manual annotation process is retrospective and subject to error in the precision of the timestamp. As a result we might encounter situations where the timestamp of the nugget is later than the earliest matching update.

In response to an event's news coverage, a system/run broadcasts a set of timestamped updates generated in the manner described in the Guidelines. We refer to a system's set of updates as $\mathcal{S}$. The set of updates received before time τ is,

$$\mathcal{S}_\tau = \{u \in \mathcal{S} : u.t < \tau\} \tag{1}$$

Our goal in this evaluation is to measure the precision, recall, timeliness, and novelty of updates provided by a system.

3.1.1 Preliminaries

Our evaluation metrics are based on the following auxiliary functions.

- **Nugget Relevance.** Each nugget $n \in \mathcal{N}$ has an associated relevance/importance grade,

$$\mathbf{R} : \mathcal{N} \to [0, 1] \tag{2}$$

$\mathbf{R}(n)$ measures the importance of the content (information) in the nugget. Nugget importance was provided on a 0-3 scale by assessors (no importance to high importance). For graded relevance, we normalize on an exponential scale, since high importance nuggets are described as "of key importance to the query", whereas low importance nuggets are "of any importance to the query". When binary relevance is needed, everything of any relevance is relevant (0 is the only non-relevant grade). The actual relevance functions used are presented below; $n.i$ denotes the nugget importance as assigned by the assessor.

$$\mathbf{R}_{\text{graded}}(n) = \frac{e^{n.i}}{e^{\max_{n' \in \mathcal{N}} n'.i}} \qquad \text{Graded relevance} \tag{3}$$

$$\mathbf{R}_{\text{binary}}(n) = \begin{cases} 1 & \text{iff } n.i > 0 \\ 0 & \text{otherwise} \end{cases} \qquad \text{Binary relevance} \tag{4}$$

$$\tag{5}$$

Note that for graded relevance, returning exactly the nugget set as the system output updates and nothing more ("perfect system"), would usually *not* result in an expected gain of 1. However, using binary relevance, the perfect system would score an expected gain of 1.

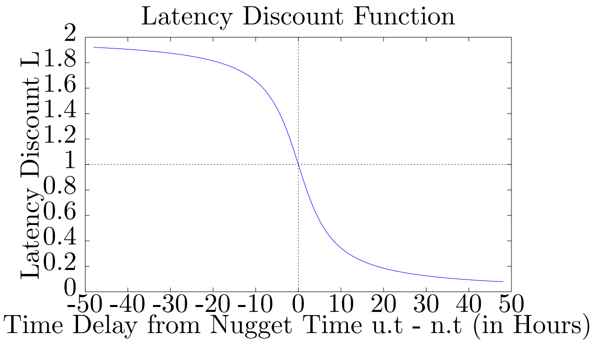
The relevance can be discounted in time or in size, hence the following discounting functions.

- **Latency Discount.** Given a reference timestamp of a matching nugget, t^*, a latency penalty function $\mathbf{L}(t^*, t)$ is a monotonically *decreasing* function of $t - t^*$. A system may return an update matching Wikipedia information before the Wikipedia information exists; thus we use a function that is smooth and decays on both the positive and negative sides.

The actual function used is presented below with arguments the nugget Wikipedia time (wiki-edit timestamp) $n.t$, and the update time $u.t$ as indicated by the system.

$$\mathbf{L}(n.t, u.t) = 1 - \frac{2}{\pi} \arctan(\frac{u.t - n.t}{\alpha}) \qquad \text{latency-discount} \tag{6}$$

$$\alpha = 3600 * 6 \qquad \text{latency-step (6 hours)} \tag{7}$$



Current parameters allow the latency discount factor to vary from 0 to 2 (1 means nugget time equal to update time), and flattens at around one day(± 24 hours). Note that as a result, gain and expected gain can be greater than 1.

- **Verbosity Normalization.** The task definition assumes that a user receives a stream of updates from the system. Consequently, we want to penalize systems for including unreasonably long updates, since these easily lead to significantly higher reading effort. The verbosity can be defined as a string length penalty function, monotonically increasing in the number of words of the update *string*. We will refer to this normalization function as $\mathbf{V}(u)$.

  For the actual verbosity implementation, we approximate the number of extra nuggets worth of information in a given update. This is done by finding all text which did not match a nugget (as defined by the assessors), and dividing the number of words in the text by the average number of words in a nugget for that query.

$$\begin{aligned}\mathbf{V}(u) &= 1 + \frac{|all\_words_u| - |nuggetmatching\_words_u|}{AVG_n|words_n|} && (8)\\ &= 1 + \frac{|u| - |\cup_{n \in \mathbf{M}^{-1}(u,\mathcal{S})} \mathbf{M}(n,\mathcal{S})|}{\text{avg}_{n \in \mathcal{N}}|n|} && \text{verbosity-normalization}\\ & && (9)\end{aligned}$$

  where $|u|$, $|n|$ are the length (in number of words) of the update u, and nugget n.

  Note that if an update has all its words part of some match to a nugget, the verbosity is $V(u)$=1; otherwise $V(u) - 1$ is an approximation of the "extra non-matching words" in terms of equivalent number of nuggets.

- **Update-Nugget Matching.** We also define a very important *earliest matching function* between a nugget and an update set,

$$\mathbf{M}(n,\mathcal{S}) = \text{argmin}_{\{u \in \mathcal{S}: n \approx u\}} u.t \qquad (10)$$

or $\emptyset$ if there is no matching update for n. $\mathbf{M}(n, \mathcal{S})$ should be interpreted as "given n, the earliest matching update in $\mathcal{S}$."

We also define the set of nuggets for which u is the earliest matching update as,

$$\mathbf{M}^{-1}(u, \mathcal{S}) = \{n \in \mathcal{N} : \mathbf{M}(n, \mathcal{S}) = u\} \tag{11}$$

Note that an update can be the earliest matching update for more than one nugget.

3.1.2 Metrics

Using the previously defined notion of *relevance, latency, verbosity, and matching* we can define several measures of interest for Temporal Summarization.

Given an update u and a matching nugget n (i.e. $u \approx n$), we can define the **discounted gain** as,

$$\mathbf{g}(u, n) = \mathbf{R}(n) \times \text{discounting factor} \tag{12}$$

Given the previously defined discounts, we have the following family of discounted gains,

$$\begin{align} \mathbf{g_F}(u, n) &= \mathbf{R}(n) && \text{discount-free gain} \tag{13} \\ \mathbf{g_L}(u, n) &= \mathbf{R}(n) \times \mathbf{L}(n.t, u.t) && \text{latency-discounted gain} \tag{14} \\ & \tag{15} \end{align}$$

Since an update can be the earliest to match several nuggets ($u \approx n$), we define the gain of an update with respect to a system (or participant run) $\mathcal{S}$ as the sum of [latency-discounted] relevance of the nuggets for which it is the earliest matching update:

$$\mathbf{G}(u, \mathcal{S}) = \sum_{n \in \mathbf{M}^{-1}(u, \mathcal{S})} \mathbf{g}(u, n) \tag{16}$$

where the gain can be any of the discounted gains described earlier. Note that for an appropriate discounting function, $\mathbf{G}(u, \mathcal{S}) \in [0, 1]$.

One way to evaluate a system is to measure the **expected gain** for a system update. This is similar to traditional notions of precision in information retrieval evaluation. Over a large population of system updates, we can estimate this measure reliably. The computation of the expected update gain for system $\mathcal{S}$ by time τ, is the average of the gain per update:

$$\begin{align} \mathbf{EG}(\mathcal{S}) &= \frac{1}{|\mathcal{S}|} \sum_{u \in \mathcal{S}} \mathbf{G}(u, \mathcal{S}) \tag{17} \\ &= \frac{1}{|\mathcal{S}|} \sum_{u \in \mathcal{S}} \sum_{n \in \mathbf{M}^{-1}(u, \mathcal{S})} \mathbf{g}(u, n) \\ &= \frac{1}{|\mathcal{S}|} \sum_{\{n \in \mathcal{N} : \mathbf{M}(n, \mathcal{S}) \neq \emptyset\}} \mathbf{g}(\mathbf{M}(n, \mathcal{S}), n) \tag{18} \end{align}$$

Additionally, we may penalize "verbosity" by normalizing not by the number of system updates, but by the overall **verbosity of the system**

$$\mathbf{EG}_{\mathbf{V}}(\mathcal{S}) = \frac{1}{\sum_{u \in \mathcal{S}} \mathbf{V}(u)} \sum_{\{n \in \mathcal{N} : \mathbf{M}(n,\mathcal{S}) \neq \emptyset\}} \mathbf{g}(\mathbf{M}(n,\mathcal{S}), n) \tag{19}$$

Our definition of **g** is such that:

- does not penalize for large a update matching several nuggets, as opposed to a few small updates matching a nugget each due to verbosity weighting,
- penalizes for late updates (against matched nugget reference timestamp), and
- penalizes "verbosity" of updates text not matching any nuggets.

Furthermore, we have that $\mathbf{G}(u, \mathcal{S}_\tau) \in [0, 1]$. Over a set of events, the mean expected gain is defined as,

$$\mathbf{MEG} = \frac{1}{|\mathcal{E}|} \sum_{\epsilon \in \mathcal{E}} \mathbf{EG}(\mathcal{S}^\epsilon) \tag{20}$$

where $\mathcal{E}$ is the set of evaluation events and $\mathcal{S}^\epsilon$ is the system submission for event ϵ.

Because a user interest may be concentrated immediately after an event and because a system's performance (in terms of gain) may be dependent on the time after an event, we will also consider a **time-sensitive expected gain** for the first τ seconds,

$$\mathbf{EG}_\tau(\mathcal{S}) = \mathbf{EG}(\mathcal{S}_\tau) \tag{21}$$

with $\mathbf{MEG}_\tau$ defined similarly.

In addition to good expected gain, we are interested in a system providing comprehensive updating. That is, we would like the system to cover as many nuggets as possible. This is similar to traditional notions of recall in information retrieval evaluation. Given a set of system updates, $\mathcal{S}$, we define the **comprehensiveness** (and **latency-comprehensiveness**) of the system as:

$$\mathbf{C}(\mathcal{S}) = \frac{1}{\sum_{n \in \mathcal{N}} \mathbf{R}(n)} \sum_{\{n \in \mathcal{N} : \mathbf{M}(n,\mathcal{S}) \neq \emptyset\}} \mathbf{g}(\mathbf{M}(n,\mathcal{S}), n) \tag{22}$$

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

$$= \frac{1}{\sum_{n \in \mathcal{N}} \mathbf{R}(n)} \sum_{u \in \mathcal{S}} \mathbf{G}(u, \mathcal{S}) \tag{23}$$

We also define a time-sensitive notion of comprehensiveness,

$$\mathbf{C}_\tau(\mathcal{S}) = \mathbf{C}(\mathcal{S}_\tau) \tag{24}$$

with an aggregated measure defined as,

$$\int_{t_s}^{t_e} \mathbf{C}_\tau(\mathcal{S})d\tau \tag{25}$$

which measures how quickly a system captures nuggets.

3.2 Value Tracking

3.2.1 Notation

Let $\mathcal{V}$ be the set of possible values deliverable to the user. A value is either a real number or a pair of numbers, depending on the type. An value update u refers to a timestamped value indexed such that $u_v \in \mathcal{V}$ and u_t is the timestamp of the update.

Given a set of value updates $\mathcal{U}$ for an event, we assume the predicted value at time τ is the value of the most recent update before τ. We represent this as $f_\mathcal{U}(\tau) \in \mathcal{V}$. In order for this function to be well-defined, we request that participants provide a baseline or prior prediction before having seen any evidence.

Given an event and an attribute type, our manual annotation process generates a set of timestamped target attribute values $\mathcal{U}^*$. In most cases, the annotation process will generate a single target value (i.e. $f_{\mathcal{U}^*}(\tau)$ is constant for all τ).

3.2.2 Metrics

We evaluate a system according to the expected error with respect to the true attribute value $f_{\mathcal{U}^*}$,

$$\mathbf{EE}(\mathcal{U},\mathcal{U}^*,\tau) = \frac{1}{t_e - t_s}\int_{t_s}^{t_e} \mathbf{Err}(\mathcal{U},\mathcal{U}^*,\tau)d\tau \tag{26}$$

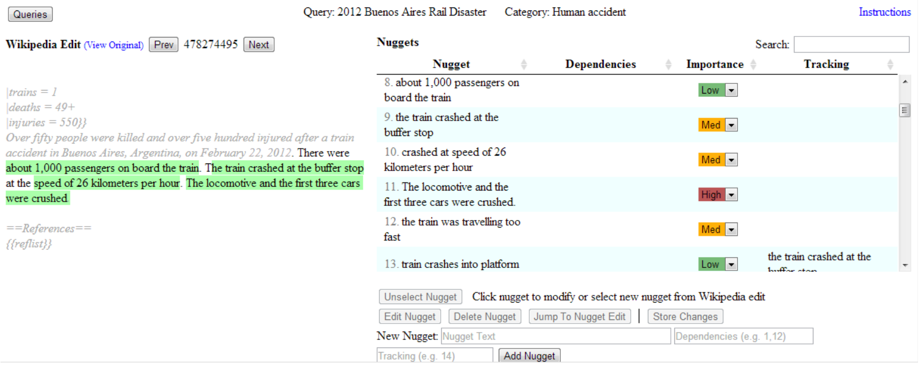
where, for scalar attributes, the error is defined as

$$\mathbf{Err}(\mathcal{U},\mathcal{U}^*,\tau) = |f_{\mathcal{U}^*}(\tau) - f_\mathcal{U}(\tau)| \tag{27}$$

and for geographic attributes, we use the Vincenty distance. We gathered target attribute values from a variety of sources, including Wikipedia and official weather reports.

4 Judging

The evaluation process occurred in two phases: 1) Gold Nugget Extraction and 2) Update-Nugget Matching. The first phase defined the space of relevant information for the queries and the second phase matched this information to updates provided by participants in order to evaluate their accuracy and coverage of the information space.



Query: 2012 Buenos Aires Rail Disaster Category: Human accident

Wikipedia Edit (View Original) 478274495

|trains = 1
|deaths = 49+
|injuries = 550}}
Over fifty people were killed and over five hundred injured after a train accident in Buenos Aires, Argentina, on February 22, 2012. There were about 1,000 passengers on board the train. The train crashed at the buffer stop at the speed of 26 kilometers per hour. The locomotive and the first three cars were crushed.

==References==
{{reflist}}

Nuggets Search:

Nugget	Dependencies	Importance	Tracking
8. about 1,000 passengers on board the train		Low	
9. the train crashed at the buffer stop		Med	
10. crashed at speed of 26 kilometers per hour		Med	
11. The locomotive and the first three cars were crushed.		High	
12. the train was travelling too fast		Med	
13. train crashes into platform		Low	the train crashed at the

Click nugget to modify or select new nugget from Wikipedia edit

New Nugget: Nugget Text Dependencies (e.g. 1,12)
Tracking (e.g. 14)

Figure 3: Extraction interface used by assessors to extract nuggets from Wikipedia edits.

4.1 Gold Nugget Extraction

In this first phase, assessors were asked to read all edits of the Wikipedia article for each query, manually extracting text perceived as relevant and novel for that edit. Additionally, assessors assigned an importance grade to every text fragment, or nugget, as well as noted any dependencies in the information. An example portion of the extraction interface can be seen in Figure 4.1.

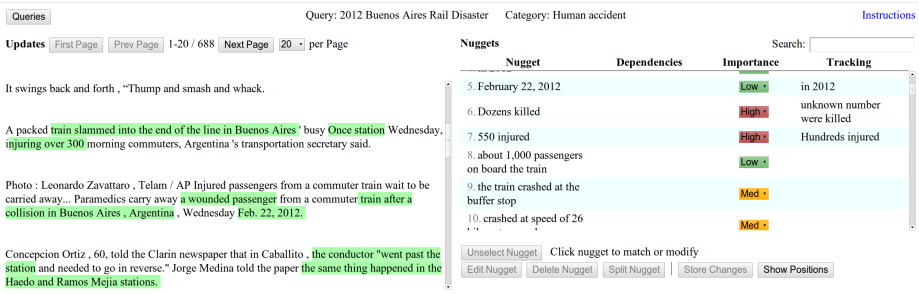
In order to simplify later matching, assessors were told to extract nuggets such that they were atomic pieces of information relevant to the query. However, knowing that information can be highly contextual, we allow for the notion of dependencies between nuggets: a nugget may be relevant to a query if and only if another nugget is also present. For evaluation purposes, a nugget was considered unmatched if it had unmatched nuggets on which it depended.

Additionally, we provided a method for assessors to track updates to pieces of information. This was primarily used to allow them to collate their work and reduce redundancy, but it was also used in the matching phase to help the assessor find the closest piece of information to a match.

4.2 Update-Nugget Matching

Once submissions were received, we performed a variant of depth-pooling in order to sample updates for evaluation. We sampled the top 60 updates per query and run as sorted by the provided confidence scores (highest first). Additionally, we performed near-duplicate detection among update text to increase the covered set. This resulted in sampled update counts as per Table 1. One note here is that not all runs contained 60 updates per query; for the run-query pairs with less than 60 updates, all updates were sampled.

The sampled updates were presented in an interface similar to the one for extraction. Assessors examined and matched updates to nuggets by selecting



Query: 2012 Buenos Aires Rail Disaster Category: Human accident

Updates 1-20 / 688

It swings back and forth , “Thump and smash and whack.

A packed train slammed into the end of the line in Buenos Aires ' busy Once station Wednesday, injuring over 300 morning commuters, Argentina 's transportation secretary said.

Photo : Leonardo Zavattaro , Telam / AP Injured passengers from a commuter train wait to be carried away... Paramedics carry away a wounded passenger from a commuter train after a collision in Buenos Aires , Argentina , Wednesday Feb. 22, 2012.

Concepcion Ortiz , 60, told the Clarin newspaper that in Caballito , the conductor "went past the station and needed to go in reverse." Jorge Medina told the paper the same thing happened in the Haedo and Ramos Mejia stations.

Nuggets

Nugget	Dependencies	Importance	Tracking
5. February 22, 2012		Low	in 2012
6. Dozens killed		High	unknown number were killed
7. 550 injured		High	Hundreds injured
8. about 1,000 passengers on board the train		Low	
9. the train crashed at the buffer stop		Med	
10. crashed at speed of 26		Med	

Click nugget to match or modify

Figure 4: Matching interface used by assessors to match updates and nuggets.

portions of updates which matched a given nugget, as nuggets are atomic but updates are not. An assessor was allowed to break a nugget into two or more new nuggets to improve atomicity if desired. Note that a nugget may match multiple updates, and an update may match multiple nuggets. An example view of the matching interface can be seen in Figure 4.2.

5 Results

5.1 Sequential Update Summarization

The results for the sequential update summarization task are summarized in Table 2. For evaluation results for individual queries, see the appendices of the TREC proceedings. It is important to note that the performance on the Gain-based measures tend to be inversely related with the performance on the Comprehensiveness-based measures. This is expected, as these measures, like precision and recall, are difficult to simultaneously optimize.

This is clear from the number of updates given by various groups, as can be seen in Table 1. The groups with higher gain have fewer updates, and those with high comprehensiveness have more updates. This shows that the task is achievable from both standpoints, but that improvements can come from attempting to rationalize these two needs or at least by prioritizing them based on user need.

These average scores are somewhat less indicative of the discriminative power of the measures as there was a fairly large amount of variance between queries. However, these averages are consistent with the performance across the queries and are therefore useful as overall measures of run performance. Additionally, it should be noted that the mean number of runs per update was 3.5.

Unfortunately, the submission for one group, BJUT, was in an inconsistent format, and as such was not pooled. Therefore even with a fixed submission, the submission was less likely to match the sampled updates, and in fact matched none of them. Another group, hltcoe, submitted fixed versions of their runs

RunID	# Updates
UWaterlooMDS-rg1	414.1 (113.2)
UWaterlooMDS-rg2	402.8 (113.7)
UWaterlooMDS-UWMDSqlec2t25	370.4 (104.6)
UWaterlooMDS-UWMDSqlec4t50	357.2 (107.0)
UWaterlooMDS-rg4	281.6 (116.6)
UWaterlooMDS-rg3	275 (107.9)
hltcoe-EXTERNAL	184.4 (77.8)
hltcoe-Baseline	183.9 (90.0)
hltcoe-BasePred	167.7 (75.5)
uogTr-uogTrEMMQ2	147.6 (34.7)
uogTr-uogTrNMM	140 (35.2)
PRIS-cluster1	100.9 (17.9)
PRIS-cluster4	100.7 (17.7)
uogTr-uogTrNMTm3FMM4	96.4 (29.2)
uogTr-uogTrNMTm1MM3	89.3 (29.9)
PRIS-cluster2	89 (17.3)
uogTr-uogTrNSQ1	80.7 (16.0)
hltcoe-TuneBasePred2*	74.8 (57.3)
wim_GY_2013-SUS1	73.1 (15.5)
UWaterlooMDS-NormEgrep	65.3 (21.3)
ICTNET-run1	56.9 (24.9)
ICTNET-run2	55.3 (24.5)
hltcoe-TuneExternal2*	49.7 (46.8)
PRIS-cluster3	38.2 (11.7)
PRIS-cluster5	21.8 (2.6)
UWaterlooMDS-CosineEgrep	11.9 (7.5)
BJUT-Q0*	0 (0.0)
ALL	145.5 (136.5)

Table 1: μ and σ(in parens) of number of updates sampled for each run over all topics. *run not pooled.

RunID	E[Gain]	E[Latency Gain]	Comprehensiveness	Latency Comp.
PRIS-cluster5	**0.149 (0.101)**	**0.136 (0.090)**	0.099 (0.099)	0.126 (0.164)
ICTNET-run2	0.102 (0.045)	0.127 (0.075)	0.192 (0.105)	0.251 (0.169)
ICTNET-run1	0.101 (0.045)	0.125 (0.075)	0.194 (0.103)	0.253 (0.169)
hltcoe-TuneExternal2*	0.109 (0.091)	0.117 (0.073)	0.162 (0.127)	0.203 (0.156)
hltcoe-TuneBasePred2*	0.093 (0.079)	0.114 (0.117)	0.180 (0.117)	0.244 (0.188)
PRIS-cluster3	0.103 (0.084)	0.103 (0.050)	0.131 (0.138)	0.176 (0.203)
PRIS-cluster2	0.071 (0.039)	0.074 (0.031)	0.204 (0.146)	0.260 (0.217)
uogTr-uogTrNMTm1MM3	0.077 (0.036)	0.069 (0.053)	0.201 (0.129)	0.216 (0.203)
PRIS-cluster4	0.065 (0.034)	0.067 (0.026)	0.224 (0.178)	0.288 (0.262)
PRIS-cluster1	0.065 (0.034)	0.067 (0.026)	0.224 (0.178)	0.292 (0.270)
hltcoe-BasePred	0.053 (0.041)	0.067 (0.057)	0.281 (0.186)	0.368 (0.272)
hltcoe-Baseline	0.049 (0.032)	0.063 (0.046)	0.291 (0.178)	0.381 (0.261)
uogTr-uogTrNSQ1	0.069 (0.031)	0.060 (0.031)	0.174 (0.123)	0.184 (0.171)
ALL–	0.055 (0.057)	0.058 (0.061)	0.231 (0.194)	0.288 (0.288)
hltcoe-EXTERNAL	0.047 (0.023)	0.054 (0.027)	0.318 (0.186)	0.413 (0.291)
uogTr-uogTrNMTm3FMM4	0.062 (0.030)	0.049 (0.028)	0.183 (0.120)	0.170 (0.143)
uogTr-uogTrNMM	0.057 (0.031)	0.045 (0.023)	0.244 (0.155)	0.254 (0.231)
uogTr-uogTrEMMQ2	0.050 (0.023)	0.040 (0.024)	0.241 (0.179)	0.259 (0.254)
wim_GY_2013-SUS1	0.037 (0.019)	0.036 (0.029)	0.128 (0.107)	0.148 (0.149)
UWaterlooMDS-rg4	0.025 (0.020)	0.028 (0.019)	0.386 (0.202)	0.516 (0.339)
UWaterlooMDS-rg3	0.024 (0.017)	0.026 (0.015)	0.384 (0.185)	0.506 (0.323)
UWaterlooMDS-rg2	0.021 (0.019)	0.022 (0.018)	0.441 (0.198)	0.562 (0.349)
UWaterlooMDS-rg1	0.020 (0.016)	0.021 (0.016)	**0.445 (0.191)**	**0.571 (0.358)**
UWaterlooMDS-UWMDSqlec4t50	0.016 (0.013)	0.018 (0.016)	0.423 (0.175)	0.530 (0.325)
UWaterlooMDS-UWMDSqlec2t25	0.016 (0.014)	0.017 (0.016)	0.433 (0.170)	0.537 (0.322)
UWaterlooMDS-CosineEgrep	0.007 (0.009)	0.010 (0.015)	0.013 (0.017)	0.018 (0.027)
UWaterlooMDS-NormEgrep	0.001 (0.002)	0.001 (0.002)	0.050 (0.076)	0.061 (0.117)
BJUT-Q0*	0 (0.0)	0 (0.0)	0 (0.0)	0 (0.0)

Table 2: μ and σ of primary task metrics over all queries, sorted by Expected Latency Gain. *run not pooled.

	location	deaths	injuries	financial impact (10^9)
baseline	20038.0	195.111	473.222	13.3539
PRIS-PRISTS1	18101.8	37880.4	64886.5	**9.5251**
PRIS-PRISTS2	11864.1	88666.5	106099	25.1175
PRIS-PRISTS3	11796.3	88666.5	106375	42.5367
BJUT-Q1	20038.0	**138.1**	473.222	13.3539
ICTNET-ValueTask	20038.0	188.495	**390.985**	13.3539
wim_GY_2013-VT1	14483.6	2726.06	410.092	13.3539
wim_GY_2013-VT2	**4660.76**	2396.12	410.531	13.3539

Table 3: Value Tracking Expected Error by Attribute. The baseline run predicts '0' for all non-location attributes and achieves a maximum location distance of 20038 km.

shortly after the deadline, but their initial submission was included in the pool.

5.2 Value Tracking

We present results for value tracking in Table 3. No system consistently performed strongly across attributes and several runs focused on individual attributes, omitting predictions for others. In order to provide a reference, Table 3 also includes the performance for a run which outputs '0' for non-location attributes. This allows us to detect runs which ignored an attribute and, for those that did not, whether the system was effective. In the cases where runs underperformed the baseline, this was often due to egregious extraction error (e.g. predicting tens of thousands of injuries instead of hundreds). Even when relatively accurate predictions were made, the expected error metric is sensitive to outliers such as these.