2-1-1998

Topic Detection and Tracking Pilot Study Final Report

James Allan
University of Massachusetts - Amherst

Jaime G. Carbonell
Carnegie Mellon University, jgc@cs.cmu.edu

George Doddington
DARPA

Jonathan Yamron
Dragon Systems

Yiming Yang
Carnegie Mellon University, yiming@cs.cmu.edu

Topic Detection and Tracking Pilot Study Final Report

James Allan[*], *Jaime Carbonell*[†], *George Doddington*[‡], *Jonathan Yamron*[§], *and Yiming Yang*[†]

[*]UMass Amherst, [†]CMU, [‡]DARPA, [§]Dragon Systems, and [†]CMU

ABSTRACT

Topic Detection and Tracking (TDT) is a DARPA-sponsored initiative to investigate the state of the art in finding and following new events in a stream of broadcast news stories. The TDT problem consists of three major tasks: (1) segmenting a stream of data, especially recognized speech, into distinct stories; (2) identifying those news stories that are the first to discuss a new event occurring in the news; and (3) given a small number of sample news stories about an event, finding all following stories in the stream.

The TDT Pilot Study ran from September 1996 through October 1997. The primary participants were DARPA, Carnegie Mellon University, Dragon Systems, and the University of Massachusetts at Amherst. This report summarizes the findings of the pilot study.

The TDT work continues in a new project involving larger training and test corpora, more active participants, and a more broadly defined notion of "topic" than was used in the pilot study.

The following individuals participated in the research reported.

James Allan, UMass
Brian Archibald, CMU
Doug Beeferman, CMU
Adam Berger, CMU
Ralf Brown, CMU
Jaime Carbonell, CMU
Ira Carp, Dragon
Bruce Croft, UMass,
George Doddington, DARPA
Larry Gillick, Dragon
Alex Hauptmann, CMU
John Lafferty, CMU
Victor Lavrenko, UMass
Xin Liu, CMU
Steve Lowe, Dragon
Paul van Mulbregt, Dragon
Ron Papka, UMass
Thomas Pierce, CMU
Jay Ponte, UMass
Mike Scudder, UMass
Charles Wayne, DARPA
Jon Yamron, Dragon
Yiming Yang, CMU

1. Overview

The purpose of the Topic Detection and Tracking (TDT) Pilot Study is to advance and accurately measure the state of the art in TDT and to assess the technical challenges to be overcome. At the beginning of this study, the general TDT task domain was explored and key technical challenges were clarified. This document defines these tasks, the performance measures to be used to assess technical capabilities and research progress, and presents the results of a cooperative investigation of the state of the art.

To appear in *Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop*, February, 1998.

1.1. Background

The TDT study is intended to explore techniques for detecting the appearance of new topics and for tracking the reappearance and evolution of them. During the first portion of this study, the notion of a "topic" was modified and sharpened to be an "event", meaning some unique thing that happens at some point in time. The notion of an event differs from a broader category of events both in spatial/temporal localization and in specificity. For example, the eruption of Mount Pinatubo on June 15th, 1991 is consider to be an event, whereas volcanic eruption in general is considered to be a class of events. Events might be unexpected, such as the eruption of a volcano, or expected, such as a political election.

The TDT study assumes multiple sources of information, for example various newswires and various news broadcast programs. The information flowing from each source is assumed to be divided into a sequence of stories, which may provide information on one or more events. The general task is to identify the events being discussed in these stories, in terms of the stories that discuss them. Stories that discuss unexpected events will of course follow the event, whereas stories on expected events can both precede and follow the event.

The remainder of this section outlines the three major tasks of the study, discusses the evaluation testbed, and describes the evaluation measures that were used. Section presents the approaches used by the study members to address the problem of text segmentation and discusses the results. The detection task is taken up and similarly described in Section . Section presents the approaches and results of the tracking task, including a brief section on tracking using a corpus created from speech recognition output.

1.2. The Corpus

A corpus of text and transcribed speech has been developed to support the TDT study effort. This study corpus spans the period from July 1, 1994 to June 30, 1995 and includes nearly 16,000 stories, with about half taken from Reuters newswire and half from CNN broadcast news transcripts. The transcripts were produced by the Journal of Graphics Institute (JGI). The stories in this corpus are arranged in chronological order, are structured in SGML format, and are available

from the Linguistic Data Consortium (LDC).[1]

A set of 25 target events has been defined to support the TDT study effort. These events span a spectrum of event types and include both expected and unexpected events. They are described in some detail in documents provided as part of the TDT Corpus. The TDT corpus was completely annotated with respect to these events, so that each story in the corpus is appropriately flagged for each of the target events discussed in it. There are three flag values possible: YES (the story discusses the event), NO (the story doesn't discuss the event), and BRIEF (the story mentions the event only briefly, or merely references the event without discussion; less than 10% of the story is about the event in question). Flag values for all events are available in the file `tdt-corpus.judgments`.[2]

1.3. The Tasks

The Topic Detection and Tracking Study is concerned with the detection and tracking of events. The input to this process is a stream of stories. This stream may or may not be pre-segmented into stories, and the events may or may not be known to the system (i.e., the system may or may not be trained to recognize specific events). This leads to the definition of three technical tasks to be addressed in the TDT study. These are namely the tracking of known events, the detection of unknown events, and the segmentation of a news source into stories.

The Segmentation Task The segmentation task is defined to be the task of segmenting a continuous stream of text (including transcribed speech) into its constituent stories. To support this task the story texts from the study corpus will be concatenated and used as input to a segmenter. This concatenated text stream will include only the actual story texts and will exclude external and internal tag information. The segmentation task is to correctly locate the boundaries between adjacent stories, for all stories in the corpus.

The Detection Task The detection task is characterized by the lack of knowledge of the event to be detected. In such a case, one may wish to retrospectively process a corpus of stories to identify the events discussed therein, or one may wish to identify new events as they occur, based on an on-line stream of stories. Both of these alternatives are supported under the detection task.

Retrospective Event Detection The retrospective detection task is defined to be the task of identifying all of the events in a corpus of stories. Events are defined by their association with stories, and therefore the task is to group the stories in the study corpus into clusters, where each cluster represents an event and where the stories in the cluster discuss the event. It will be assumed that each story discusses at most one event. Therefore each story may be included in at most one cluster.[3]

On-line New Event Detection The on-line new event detection task is defined to be the task of identifying new events in a stream of stories. Each story is processed in sequence, and a decision is made whether or not a new event is discussed in the story, after processing the story but before processing any subsequent stories). A decision is made after each story is processed. The first story to discuss an event should be flagged YES. If the story doesn't discuss any new events, then it should be flagged NO.

The Tracking Task The tracking task is defined to be the task of associating incoming stories with events known to the system. An event is defined ("known") by its association with stories that discuss the event. Thus each target event is defined by a list of stories that discuss it.

In the tracking task a target event is given, and each successive story must be classified as to whether or not it discusses the target event. To support this task the study corpus will be divided into two parts, with the first part being the training set and the second part being the test set. (This division is different for each event, in order to have appropriate training and test sets.) Each of the stories in the training set will be flagged as to whether it discusses the target event, and these flags (and the associated text of the stories) will be the only information used for training the system to correctly classify the target event. The tracking task is to correctly classify all of the stories in the test set as to whether or not they discuss the target event.

A primary task parameter is the number of stories used to define ("train") the target event, N_t. The division of the corpus between training and test will be a function of the event and the value of N_t . Specifically, the training set for a particular event and a particular value of N_t will be all of the stories up to and including the N_t^{th} story that discusses that event. The test set will be all subsequent stories.

1.4. The Evaluation

To assess TDT application potential, and to calibrate and guide TDT technology development, TDT task performance will be evaluated formally according to a set of rules for each

[1] Linguistic Data Consortium Telephone: 215 898-0464 3615 Market Street Fax: 215 573-2175 Suite 200 ldc@ldc.upenn.edu Philadelphia, PA, 19104-2608, USA. `http://www.ldc.upenn.edu`

[2] Only values of YES and BRIEF are listed, thus reducing the size of the judgment file by two orders of magnitude. (The vast majority of stories have flag values of NO for all events.)

[3] While it is reasonable that a story will typically discuss a single event, this is not always the case. In addition to multifaceted stories, there are also overlapping events. For example, in the case of the TDT study's corpus and target events, there are 10 stories that have a YES or BRIEF tag for more than one event. One of these (story 8481) has a YES tag for two events (namely Carter in Bosnia and Serbs violate Bihac). Nonetheless, the assumption that each story discusses only one event will be used, because it is reasonable for the large majority of stories and because it vastly simplifies the task and the evaluation.

of the three TDT tasks. In these evaluations, there will be numerous conditions and questions to be explored. Among these are:

- How does performance vary when processing different sources and types of sources?
- How does selection of training source and type affect performance?

In general evaluation will be in terms of classical detection theory, in which performance is characterized in terms of two different kinds of errors, namely misses (in which the target event is not detected) and false alarms (in which the target event is falsely detected). In this framework, different events will be treated independently of each other and a system will have separate outputs for each of the target events.

2. Segmentation

The segmentation task addresses the problem of automatically dividing a text stream into topically homogeneous blocks. The motivation for this capability in this study arises from the desire to apply event tracking and detection technology to automatically generated transcriptions of broadcast news, the quality of which have improved considerably in recent years. Unlike newswire, typical automatically transcribed audio data contains little information about how the stream should be broken, so segmentation must be done before further processing is possible. Segmentation is therefore an "enabling" technology for other applications, such as tracking and new event detection.

Given the nature of the medium, "topically homogeneous blocks" of broadcast speech should correspond to stories, hence a segmenter which is designed for this task will find story boundaries. The approaches described below, however, are quite general; there is no reason that the same technology, suitably tuned, cannot be applied to other segmentation problems, such as finding topic breaks in non-news broadcast formats or long text documents.

There is a relatively small but varied body of previous work that has addressed the problem of text segmentation. This work includes methods based on semantic word networks [10], vector space techniques from information retrieval [7], and decision tree induction algorithms [11]. The research on segmentation carried out under the TDT study has led to the development of several new and complementary approaches that do not directly use the methods of this previous work, although all of the approaches share a common rationale and motivation.

2.1. Evaluation

Segmentation will be evaluated in two different ways. First, segmentation will be evaluated directly in terms of its ability to correctly locate the boundaries between stories. Second, segmentation will be evaluated indirectly in terms of its ability to support event tracking and preserve event tracking performance.

For the segmentation task, all of the TDT study corpus will be reserved for evaluation purposes. This means that any material to be used for training the segmentation system must come from sources other than the TDT study corpus. Also, the nature of the segmentation task is that the segmentation is performed on a single homogeneous data source. Therefore, for the purpose of evaluating the segmentation task, segmentation will be performed not only on the TDT Corpus as a whole, but also on its two separate sub-streams–one comprising just the Reuters stories, and the other comprising just the CNN stories. In addition, the segmentation task must be performed without explicit knowledge of the source of the text, whether from newswire or transcribed speech.

Direct Evaluation of Segmentation Segmentation will be evaluated directly using a modification of a method suggested by John Lafferty.[4]

This is an ingenious method that avoids dealing with boundaries explicitly. Instead, it measures the probability that two sentences drawn at random from the corpus are correctly classified as to whether they belong to the same story. For the TDT study, the calculation will be performed on words rather than sentences.[5] Also, the error probability will be split into two parts, namely the probability of misclassification due to a missed boundary (a "miss"), and the probability of misclassification due to an extraneous boundary (a "false alarm"). These error probabilities are defined as

$$P_{miss} = \frac{\sum_{i=1}^{N-k} \delta_{hyp}(i, i+k) \cdot (1 - \delta_{ref}(i, i_+k))}{\sum_{i=1}^{N-k} (1 - \delta_{ref}(i, i+k))}$$

$$P_{FalseAlarm} = \frac{\sum_{i=1}^{N-k} (1 - \delta_{hyp}(i, i+k)) \cdot \delta_{ref}(i, i_+k)}{\sum_{i=1}^{N-k} \delta_{ref}(i, i+k)}$$

where the summations are over all the words in the corpus and where

$$\delta(i,j) = \begin{cases} 1 & \text{when words } i \text{ and } j \text{ are from the same story} \\ 0 & \text{otherwise} \end{cases}$$

Choice of k is a critical consideration in order to produce a meaningful and sensitive evaluation. For the TDT study corpus, k will be chosen to be half the average document length, in words, of the text stream on which we evaluate (about 250 for the TDT Corpus, for example).

[4] "Text Segmentation Using Exponential Models", by Doug Beeferman, Adam Berger, and John Lafferty.

[5] There are several reasons for using words rather than stories. First, there will likely be less debate and fewer problems in deciding how to delimit words than how to delimit sentences. Second, the word seems like a more suitable unit of measurement, because of the relatively high variability of the length of sentences.

Indirect Evaluation of Segmentation Segmentation will be evaluated indirectly by measuring event tracking performance on stories as they are defined by automatic segmentation means. A segment will contribute to detection errors proportionate to how it overlaps with stories that would contribute to the error rates. Details of this evaluation are presented in Section in the tracking chapter.

2.2. Dragon Approach

Theory Dragon's approach to segmentation is to treat a story as an instance of some underlying topic, and to model an unbroken text stream as an unlabeled sequence of these topics. In this model, finding story boundaries is equivalent to finding topic transitions.

At a certain level of abstraction, identifying topics in a text stream is similar to recognizing speech in an acoustic stream. Each topic block in a text stream is analogous to a phoneme in speech recognition, and each word or sentence (depending on the granularity of the segmentation) is analogous to an "acoustic frame". Identifying the sequence of topics in an unbroken transcript therefore corresponds to recognizing phonemes in a continuous speech stream. Just as in speech recognition, this situation is subject to analysis using classic Hidden Markov Model (HMM) techniques, in which the hidden states are topics and the observations are words or sentences.

More concretely, suppose that there are k topics $T^{(1)}$, $T^{(2)}$, ..., $T^{(k)}$. There is a language model associated with each topic $T^{(i)}$, $1 \leq i \leq k$, in which one can calculate the probability of any sequence of words. In addition, there are transition probabilities among the topics, including a probability for each topic to transition to itself (the "self-loop" probability), which implicitly specifies an expected duration for that topic. Given a text stream, a probability can be attached to any particular hypothesis about the sequence and segmentation of topics in the following way:

1. Transition from the start state to the first topic, accumulating a transition probability.
2. Stay in topic for a certain number of words or sentences, and, given the current topic, accumulate a self-loop probability and a language model probability for each.
3. Transition to a new topic, accumulating the transition probability. Go back to step 2.

A search for the best hypothesis and corresponding segmentation can be done using standard HMM techniques and standard speech recognition tricks (using thresholding if the search space gets too large, for example).

Implementation Details Since the entire TDT Corpus is set aside for evaluation, training data for a segmenter must come from other sources. One such source available to all sites is the portion of Journal Graphics data from the period January 1992 through June 1994. This data was restricted to the CNN shows included in the TDT Corpus, and stories of fewer than 100 and more than 2,000 words were removed. This left 15,873 stories of average length 530 words. A global unigram model consisting of 60,000 words was built from this data.

The topics used by the segmenter, which are referred to as *background* topics, were constructed by automatically clustering news stories from this training set. The clustering was done using a multi-pass k-means algorithm that operates as follows:

1. At any given point there are k clusters. For each story, determine its distance to the closest cluster (based on the measure described below), and if this distance is below a threshold, insert the story into the cluster and update the statistics. If this distance is above the threshold, create a new cluster.
2. Loop through the stories again, but now consider switching each story from its present topic to the others, based on the same measure as before. Some clusters may vanish; additional clusters may need to be created. Repeat this step as often as desired.

The distance measure used in the clustering was a variation of the symmetric Kullback-Leibler (KL) metric:

$$\begin{aligned} d &= \sum_n (s_n/S) \log \frac{s_n/S}{(c_n + s_n)/(C+S)} \\ &+ \sum_n (c_n/C) \log \frac{c_n/C}{(c_n + s_n)/(C+S)}, \end{aligned}$$

where s_n and c_n are the story and cluster counts for word w_n, with $S = \sum s_n$ and $C = \sum c_n$.

A background topic language model was built from each cluster. To simplify this task, the number of clusters was limited to 100 and each topic was modeled with unigram statistics only. These unigram models were just smoothed versions of the raw unigram models generated from the clusters. Smoothing each model consisted of performing absolute discounting followed by backoff to the global unigram model. The unigram models were filtered against a stop list to remove 174 common words.

Decoding of text was done by actually using code from a speech recognizer with 100 underlying "single node" models (corresponding to the topics), each of which was represented by a unigram model as described above. As in speech, the text was scored against these models one *frame* at a time –

a frame corresponding, in these experiments, to a sentence. The topic-topic transition penalties were folded into a single number, the topic-switch penalty, which was imposed whenever the topic changed between frames/sentences.

The topic-switch penalty was tuned to produce the correct average number of words per segment on the first 100 stories from the test set. There are no other parameters to tune except the search beam width, which was set large enough to avoid search errors in the experiments.

Results

TDT Corpus. The segmentation error metric computed for Dragon's system on the full TDT Corpus was 12.9%. The segmenter produced 16,139 story boundaries, compared to the 15,863 actual boundaries in the test set. Of these, 10,625 were exact matches, yielding a recall rate of 67.0% and a precision of 65.8%.

CNN vs. Reuters. One might expect that, because the data used to train the segmenter's background models was taken entirely from CNN broadcasts, the performance of the segmenter on the CNN portion of the TDT Corpus would be significantly better than its performance on the Reuters portion. To explore this, Dragon ran the evaluation separately on the two subcorpora. The system returned a segmentation error of 16.8% (worse than for the corpus as a whole!) on CNN, and an error of 12.3% (better!) on Reuters.

The most likely explanation for this anomaly is that the CNN is more difficult than Reuters for a content-based segmenter such as Dragon's. For example, written news tends to be more concise than broadcast news, with none of the typical "broadcast fillers", such as introductions, greetings, and sign-offs. It is also the case that the length of CNN stories varies much more widely than Reuters stories, a problem for this segmenter, which has a single parameter controlling for length.

TWA Corpus. The closed-caption version of the corpus contains punctuation marks, making it possible to introduce sentence breaks in the usual way. The recognized transcriptions, of course, contain no punctuation, so breaks were introduced at arbitrary points in the segments in such a way as to produce approximately the same number of "sentences" as in the closed-caption case.

On the closed-caption data, the segmenter returned a segmentation error of 25.5%. On the recognized data the error was 33.6%. The size of these numbers suggests that the problem of segmenting broadcasts may be harder than the TDT Corpus leads us to believe. In any event, it would be interesting to calibrate these error rates against the result on a clean transcription of the TWA Corpus.

The Future It is remarkable that this simple application of HMM techniques to segmentation and tracking achieves such promising results. This work represents just the beginning of what can be achieved with this approach; many improvements are possible, both by incorporating ideas found in implementations at the other sites and from generalizations of the techniques already employed.

In particular, some form of story modeling that attempts to recognize features around boundaries, which both UMass and CMU incorporate into their systems, should be incorporated into Dragon's framework. One way to do this, which continues in the spirit of the speech recognition analogy, is to use "multi-node" story models, in which a story is modeled as a sequence of nodes (for example, one which models the story start, one which models the middle, and one which models the end) rather than a single topic model.

It is also possible to improve the topic modeling that already forms the basis of the segmenter. Some methods of achieving this include using bigram models in place of unigram models for topics, including a "trigger model" of the kind employed by CMU, and adaptively training the background during segmentation. It is also likely that the basic speech-inspired language models can be improved by incorporating information retrieval measures that are more informed about topic information, such as the local context analysis used by UMass.

2.3. UMass Approach

Content Based LCA Segmentation UMass has developed two largely complementary segmentation methods. The first method makes use of the technique of local context analysis (LCA) [16]. LCA was developed as a method for automatic expansion of ad hoc queries for information retrieval. It is somewhat like the method of local feedback [5] but has been shown to be more effective and more robust. For the segmentation task, LCA can be thought of as an association thesaurus which will return words and phrases which are semantically related to the query text and are determined based on collection-wide co-occurrence as well as similarity to the original sentence. Each sentence is run as a query against the LCA database and the top 100 concepts are returned. The original sentence is then replaced with the LCA concepts and the effect is that sentences which originally had few or perhaps no words in common will typically have many LCA concepts in common.

The original LCA method was derived from that described in [12]. The text is indexed at the sentence level using offsets to encode the positions of the LCA features. For example, suppose the feature "O. J. Simpson" occurs in sentence 1, 3, and 10. The index will encode these positions as 1, 2 and 7, the offset from the previous occurrence of the concept. The main idea of the LCA segmenter is to use these offsets to measure shifts in vocabulary over time. The original method, which was tested on the Wall Street Journal, used a simple function of the offsets as a heuristic measure of the "surprise"

of seeing a particular concept in a particular sentence. In a homogeneous collection such as the Wall Street Journal, this heuristic, in conjunction with LCA expansion, worked quite well. However, the TDT Corpus has stories from several sources and so it often happens that several stories on the same topic will occur in close proximity. Moreover, since the TDT Corpus consists of transcribed speech, there is far more off-topic language than in the Wall Street Journal. For example, throughout the corpus, one finds social interaction between speakers which does not relate to the current topic. These two difficulties were circumvented by means of an exponential length model. Rather than looking at the total size of the offset, a model of the average segment size was used. The model was used to determine the probability that an occurrence of a concept was in the same segment as the previous occurrence. This method is more robust with respect to multiple stories on same topic and to "social noise" than the original method and performance is improved.

The LCA method can be thought of as a content-based method. It works by looking at changes in content-bearing words. It is somewhat similar to the topic models used in Dragon's method and to the relevance features in CMU's method. The strong point of the LCA method is that, other than the length model estimation, it is completely unsupervised. One weakness of this method is that the current implementation is somewhat slow since it requires a database query per sentence. However, it could be sped up considerably using standard information retrieval query optimization techniques. A second weakness is that performance of the LCA expansion currently requires sentence breaks. A modification of this approach would be to use a fixed-sized window rather than sentences as the atomic unit for expansion.

Discourse Based HMM Segmentation The second segmentation method uses a Hidden Markov Model to model "marker words," or words which predict a topic change. The model consists of one or more states for the first N sentences of a segment, one or more for the last N sentences, and one or more for the remainder of the segment. So while the LCA segmenter relies on shifts in content, the HMM segmenter is relying on words which predict the beginning or end of a segment without regard to content. This is somewhat similar to CMU's use of vocabulary features. The model is trained using segmented data. Unknown word probabilities were handled with a very simple smoothing method.

Additional Features. In addition to the word probabilities, other features were modeled. These included sentence length (which would be implicit in a word based segmenter), serial clustering tendency [3], and distance from previous occurrence. Each of these features was measured as a standard score, and state probabilities were estimated from the training data. These three features yielded a very slight improvement over the words alone. Part of the reason why they did not help more is that, in the first place, the distributions of the features are far from normal and, secondly, most of the data points cluster around the mean. This suggests that an adaptive binning technique would work better than using standardized scores.

In order to shed some light on this conjecture, all of the data points lying more than one standard deviation from the mean were discarded and a new mean and standard deviation were computed and the scores restandardized. This admittedly poor modification yielded a modest improvement over the initial standard scores and therefore suggests that adaptive binning would be appropriate. However, it is not known to what extent the results would improve from better binning.

One advantage of the HMM implementation is that it is very fast. Training time is approximately 15 minutes on the TDT training corpus and segmentation is extremely fast as one would expect from an HMM with a small number of states. Also, unlike the LCA method, the HMM method can be used at the word level (although the current implementation works at the sentence level). The disadvantage of the HMM method is that it requires segmented training data.

Results and Discussion

LCA Method. The LCA segmenter achieves a 17.6% error rate on the TDT Corpus. The new method is still heuristic in nature and a more principled use of the LCA concepts would, in all likelihood, improve performance further. Two additional improvements could be made to the LCA approach. First, one difficulty with the LCA method is that when one gives a query to LCA such as "Thank you and good-night," the concepts one gets back are essentially random. The current method is fairly robust with respect to a reasonable amount of random noise, but perhaps a better approach would be to model the noise words and not pass them to LCA at all. The second approach is to make use of the discourse features as well. This is discussed further below.

HMM Method. The HMM segmenter has a 23% error rate on the TDT Corpus. One caveat is that this approach may rely on the similarity of the training data to the test data somewhat heavily. Still, it shows that very simple discourse modeling can provide useful information. This method could be made more robust by explicitly modeling "segues" and other regularities of the source. For example, it would be more general to tag place names and names of reporters and to learn the probability of segment boundaries relative to the tags rather than to the specific names as the current approach does.

The Future One obvious question is to what extent a hybrid approach would improve performance over either method alone. For example, one could use an HMM based segmenter and sample the LCA concepts at locations where the distribution is less peaked, i.e. use LCA in places where one least sure about a break. A second reasonable hybrid would be to

combine the content-based HMM segmenter used by Dragon with a simple discourse-based HMM segmenter.

It may also be possible to leverage the strengths of the two approaches as follows. The LCA segmenter works in an unsupervised manner but is somewhat slow. The HMM segmenter is very fast, but requires training data. Over time, one could use the LCA segmenter on a sample of the incoming data in order to provide "up to the minute" training data for the faster HMM segmenter in order to keep the distributions up to date as the language use shifts over time.

2.4. CMU Approach

Motivation The original motivation for the CMU segmentation research arose in the context of multimedia information retrieval applications. In particular, both the News-on-Demand and video library projects within Informedia Digital Libraries project require segmentation of the video stream for accurate and useful indexing, retrieval, browsing, and summarization.

In order to find natural breaks in a video stream, it is important to make use of the concurrent and often complementary information in the text (closed captions or speech output), audio, and image streams. The CMU approach was designed around the idea that various "features" of these multiple media sources should be extracted and then combined into a statistical model that appropriately weighs the evidence, and then decides where to place segment boundaries. For multimedia, the relevant features might include questions such as: *Does the phrase* COMING UP *appear in the last utterance of the decoded speech? Is there a sharp change in the video stream in the last* 20 *frames? Is there a "match" between the current image and an image near the last segment boundary? Are there blank video frames nearby? Is there a significant change in the frequency profile of the audio stream in the next utterance?*

There are several key ingredients in this basic approach applied to the subproblem of text segmentation:

1. Content-based features derived from a pair of language models that are used to help gauge "large scale" changes of topic.
2. Lexical features that extract information about the local linguistic and discourse structure of the context.
3. A new machine learning algorithm that incrementally selects the best lexical features and combines them with the information provided by the language models to form a unified statistical model.

The use of language models, as described below, is geared toward finding changes of topic—whether within or across segment boundaries. This component is similar in spirit to Dragon's use of unigram language models trained on clusters of segments, and the UMass local context analysis technique. The lexical features complement this information by making more fine-grained judgments about those words that correlate—both positively and negatively—with segment boundaries. The feature selection algorithm automatically "learns" how to segment by observing how segmentation boundaries are placed in a sample of training text. This algorithm incrementally constructs an increasingly detailed model to estimate the probability that a segment boundary is placed in a given context. Each of these ingredients is described in more detail below.

Language Models. In the CMU approach the relative behavior of an *adaptive* language model is compared to a *static* trigram language model in an on-line manner. The basic idea is that the adaptive model generally gets better and better as it sees more material that is relevant to the current "topic" of a segment. However, when the topic changes, the performance of the adaptive model degrades relative to the trigram model since it is making its predictions based upon the content words of the previous topic. These language models are essentially the same as those employed for the speech recognition system used in CMU's entry in the recent TREC evaluation for spoken document information retrieval.

Two static trigram models are used—one for the CNN experiments and one for Reuters experiments. The CNN experiments use a static trigram model $p_{\text{tri}}(w \mid w_{-2}, w_{-1})$ with a vocabulary of roughly 60,000 words that is trained on approximately 150 million words (four and a half years) of transcripts of various news broadcasts, including CNN news, but excluding those Journal Graphics transcriptions that overlap with the time frame of the TDT Corpus. The Reuters experiments use a trigram model that has a vocabulary of 20,000 words and is trained on approximately 38 million words of Wall Street Journal data. Both models use the Katz backoff scheme [9] for smoothing.

The method used to construct the adaptive model is to treat the static trigram model as a default distribution, and then to add certain features based on semantic word classes in order to form a family of conditional exponential models. The details of this model are described in [1]. Since the adaptive model should improve as it sees more material from the current topic (or event), a segment boundary is likely to exist when the adaptive model suddenly shows a dip in performance—a lower assigned probability to the observed words—compared to the short-range model. Conversely, when the adaptive model is consistently assigning higher probabilities to the observed words, a partition is less likely.

Lexical Features. The use of simple lexical features is intended to capture words or phrases that are commonly used to begin or end a segment in a particular domain, as well as to extract simple linguistic and discourse clues that a boundary

is near.

As an example, in the domain of CNN broadcast news, a story often will end with a reporter giving his or her name and the location of the report: THIS IS WOLF BLITZER REPORTING LIVE FROM THE WHITE HOUSE. In the domain of Reuters newswire, on the other hand, which originates as written communication, a story is often introduced by recording the day on which the event occurred: A TEXAS AIR NATIONAL GUARD FIGHTER JET CRASHED FRIDAY IN A REMOTE AREA OF SOUTHWEST TEXAS.

The lexical features enable the presence or absence of particular words in the surrounding context to influence the statistical segmenter. Thus, the presence of the word REPORTING in the broadcast news domain, or the presence of the word FRIDAY in the newswire domain might indicate that a segment boundary is nearby. The way in which the learning algorithm actually chooses and uses these features is described briefly in the next section.

Feature Induction The procedure for combining the evidence in the language models and the lexical features is based on a statistical framework called *feature induction* for random fields and exponential models [2, 4]. The idea is to construct a model which assigns to each position in the data stream a probability that a boundary belongs at that position. This probability distribution is incrementally constructed as a log-linear model that weighs different "features" of the data. For simplicity, it is assumed that the features are binary questions.

One way to cast the problem of determining segment boundaries in statistical terms is to construct a probability distribution $q(b \,|\, \omega)$, where $b \in \{\text{YES}, \text{NO}\}$ is a random variable describing the presence of a segment boundary in context ω. Consider distributions in the *linear exponential family* $\mathcal{Q}(f, q_0)$ given by

$$\mathcal{Q}(f, q_0) = \left\{ q(b \,|\, \omega) = \frac{1}{Z_\lambda(\omega)} e^{\lambda \cdot f(\omega)} \; q_0(b \,|\, \omega) \right\}$$

where $q_0(b \,|\, \omega)$ is a prior or *default* distribution on the presence of a boundary, and $\lambda \cdot f(\omega)$ is a linear combination of binary features $f_i(\omega) \in \{0, 1\}$ with real-valued *feature parameters* λ_i:

$$\lambda \cdot f(\omega) = \lambda_1 f_1(\omega) + \lambda_2 f_2(\omega) + \cdots \lambda_n f_n(\omega) \,.$$

The normalization constants $Z_\lambda(\omega) = 1 + e^{\lambda \cdot f(\omega)}$ insure that this is indeed a family of conditional probability distributions.

The judgment of the merit of a model $q \in \mathcal{Q}(f, q_0)$ relative to a reference distribution $p \notin \mathcal{Q}(f, q_0)$ during training is made in terms of the Kullback-Leibler divergence

$$D(p \,\|\, q) = \sum_{\omega \in \Omega} p(\omega) \sum_{b \in \{\text{YES,NO}\}} p(b \,|\, \omega) \log \frac{p(b \,|\, \omega)}{q(b \,|\, \omega)} \,.$$

Thus, when p is chosen to be the empirical distribution of a sample of training events $\{(\omega, b)\}$, the maximum likelihood criterion is used for model selection. The training algorithm for choosing the parameters to minimize the divergence is the *Improved Iterative Scaling* algorithm presented in [4].

This explains how a model is chosen once the features $f_1, \ldots, f_n$ are known, but how are these features to be found? One possibility is a greedy algorithm akin to growing a decision tree, although the models are closer to the form of certain neural networks. In brief, the *gain* of a candidate is estimated as the improvement to the model that would result from adding the feature and adjusting its weight to the best value. After calculating the gain of each candidate feature, the one with the largest gain is chosen to be added to the model, and all of the model's parameters are then adjusted using iterative scaling. In this manner, an exponential model is incrementally built up using the most informative features. See [4] for details.

Results The exponential models derived using feature induction give a probability $p(b = \text{YES} \,|\, \omega)$ that a boundary exists at a given position in the text. In order to actually segment text, this probability is computed in an "on-line" manner, scanning the text sequentially. It is assumed that sentence boundaries have been identified, and segment boundaries are only placed between sentences. A segment boundary is hypothesized if (1) the probability $p(b = \text{YES} \,|\, \omega)$ exceeds a pre-specified threshold α, and (2) a boundary has not been previously placed in the immediately preceding ϵ sentences. The parameters α and ϵ were chosen on a portion of heldout training data to minimize the error probability, and were set to $\alpha = 0.15$ and $\epsilon = 5$ for CNN data, and $\alpha = 0.25$ and $\epsilon = 2$ on newswire data.

As described in the TDT Evaluation Plan, the direct evaluation for a hypothesized segmentation `hyp` with respect to the reference segmentation `ref` is computed as a probability $p(\text{error} \,|\, \texttt{ref}, \texttt{hyp}, k)$. This is the probability that a randomly chosen pair of words a distance of k words apart is inconsistently classified; that is, for one of the segmentations the pair lies in the same segment, while for the other the pair spans a segment boundary.

The CNN segmentation model was trained on approximately one million words of broadcast news data not included in the TDT Corpus, using the broadcast news language models described above as the basis for language model features. A total of 50 features were induced, and the model was trained using the Improved Iterative Scaling algorithm. The selection of each feature from the pool of several hundred thousand candidates takes on the order of 30 minutes, and then training all of the weights takes roughly five minutes, on a high-end workstation. No cross-validation was done to determine the best stopping point, nor was the resulting model smoothed in any way. One of the advantages of feature induc-

tion for exponential models, versus more standard machine learning techniques such as decision trees, is that the procedure is quite robust against over-fitting. When the resulting 50 feature model was then evaluated on the CNN portion of the TDT Corpus, the error rate was 12.5%. The exact match precision and recall were 72.2% and 72.3% respectively.

The segmentation model for the Reuters portion of the TDT Corpus was built using a collection of approximately 250,000 words of AP newswire data, Wall Street Journal articles, and Reuters headline news segments extracted from the Internet. The language models used were trained on 38 million words of Wall Street Journal data. Because of the lack of training data from the Reuters domain, as well as the general absence of strong cue phrases for story transitions in this written domain, it was expected that the resulting segmentation performance would be inferior to that obtained for broadcast news, and this was indeed what happened in the CMU results. A 50 feature model was induced on the training set, and when evaluated on the Reuters portion of the TDT Corpus, the resulting error rate was 15.5%.

The Future The CMU segmentation research carried out under the TDT project is clearly only a beginning, and there are many directions in which this work can be extended, improved, and made more practical. There is current work going on at CMU to build on these results to develop segmentation algorithms for multimedia data, making use of parallel text, audio, and video streams.

The CMU approach has an economy of scale since the language models that are used are identical to those that are used for speech recognition systems constructed in the same domain. Improved language models for speech recognition can be expected to yield improved performance for segmentation. The exponential models resulting from feature induction are very "concrete" in the sense that only a handful of specific features are extracted, and the behavior of the resulting segmenter can be well understood—there are specific "explanations" of the decisions that it makes. Moreover, since the model directly assigns a probability distribution to boundaries, a confidence in the decisions is easy to assign.

The challenge of future work is to preserve these strengths while integrating the complementary strengths of the Dragon and UMass approaches.

2.5. Discussion

One of the remarkable outcomes of the TDT study on segmentation is the diversity of ideas and techniques that have been brought to bear on this problem. Broadly speaking, these ideas and techniques fall into two classes: those that focus on story *content* and those that focus on story structure or *discourse*. In the details, however, there is very little similarity between approaches. Dragon's content-based system models stories as instances of topics described by simple unigram statistics; UMass, in one approach, treats stories as collections of similar queries to an information retrieval system, and in another approach, as a set of words bounded by marker words and phrases; and CMU's system exploits both content and discourse features simultaneously, training an exponential model to combine information from a trigger/trigram language model with features that are associated with story boundaries.

				TWA	
	TDT	CNN	Reuters	cc	rec
Dragon	12.9	16.8	12.3	25.5	33.6
UMass	17.6	—	—	—	—
CMU	—	12.5	15.5	—	—

Table 1: Segmentation error rates (percentages).

This variety of approaches bodes well for the future of work on segmentation. It not only means that improvements on the current task are likely to be realized by combining some of these different ideas, but also that a variety of different tasks can be addressed by selecting the approach with the appropriate strengths. For example, on the CNN task, for which a large amount of well-matched training data was available, CMU's feature-learning mechanism proved to be very effective; on the Reuters task, for which well-matched training material was not available, Dragon's content-based system was more robust (see Table 1).

The indirect evaluation of segmentation (described in Section) shows that carefully transcribed broadcast data can probably be segmented well enough with the current methods that subsequent processing (tracking, at least) will not suffer much. It remains to be seen if the same can be said of not-so-carefully transcribed data, such as that produced by closed-captioning or recognition. The one small test that has been done using the TWA Corpus indicates that this may be a hard problem. On the other hand, in TDT segmentation of the broadcast stream is not an end in itself, but an enabling technology for subsequent tracking and detection processes, and it may prove to be the case that methods of the type developed here will be adequate to support these technologies.

3. New Event Detection

Event *detection* is the problem of identifying stories in several continuous news streams that pertain to new or previously unidentified events. In other words, detection is an unsupervised learning task (without labeled training examples). Detection may consist of discovering previously unidentified events in an accumulated collection ("retrospective detection"), or flagging the onset of new events from live news

feeds or incoming intelligence reports in an on-line fashion ("on-line detection"). Both forms of detection by design lack advance knowledge of the new events, but do have access to (unlabeled) historical data as a contrast set.

In the TDT study, the input to *retrospective detection* is the entire corpus. The required output by a detection system is a partition of the corpus, consisting of story clusters which divide the corpus into event-specific groups according to the system's judgment. (CMU's and UMass's methods exhibit considerably better performance when they are allowed to place stories within multiple event groups.)

The input to *on-line detection* is the stream of TDT stories in chronological order, simulating real-time incoming news events. The output of on-line detection is a YES/NO decision per story made at the time when the story arrives, indicating whether this story is the first reference to a newly reported event. A confidence score per decision is also required. These scores are used later to investigate potential trade-offs between different types of errors(misses and false alarms) by applying different thresholds on these scores and thus shifting the decision boundary.

How to use the above information to detect unknown events presents new research challenges. There are multiple ways to approach the problem.

- The CMU approach to retrospective event detection is to cluster stories in a bottom-up fashion based on their lexical similarity and proximity in time. The CMU approach to on-line detection combines lexical similarity (or distance) with a declining influence look-back window of k days when judging the current story, and determine NEW or OLD based on how distant of the current story from the closest story in the k days window.
- The UMass approach to on-line detection is similar to the extent that it uses a variant of single-link clustering and builds up (clusters) groups of related stories to represent events. New stories are compared to the groups of older stories. The matching threshold is adjusted over time in recognition that an event is less likely to be reported as time passes. UMass' retrospective detection method focuses on rapid changes by monitoring sudden changes in term distribution over time.
- The Dragon approach is also based on observations over term frequencies, but using adaptive language models from speech recognition. When prediction accuracy of the adapted language models drops relative to the background model(s), a novel event is hypothesized.

3.1. Detection evaluation

The detection task used the entire TDT study corpus as input. However, detection performance was evaluated only on those stories which discuss only one of the 25 target events and which are flagged as such with a YES flag for that story. There are 1131 such stories.[6]

Retrospective Event Detection System output for the retrospective event detection task is the clustering information necessary to associate each of the stories with a cluster. (Each story is constrained to appear in only one cluster.) This information is recorded in a file, one record per story, with records separated by newline characters and with fields in a record separated by white space. Each record has five fields in the following format: "`Cluster` N_t `Story Decision Score`", where:

- *Cluster* is an index number in the range $\{1, 2, \dots\}$ which indicates the cluster (event) affiliation of the story.
- N_t is the number of stories used to train the system to the event. (Since this is a detection task, $N_t = 0$, but it is kept in the output to maintain format uniformity across different tasks.)
- *Story* is the TDT corpus index number in the range $\{1, 2, \dots 15863\}$ which indicates the story being processed.
- *Decision* is either YES or NO, where YES indicates that the system believes that the story being processed discusses the cluster event, and NO indicates not. (Again, Decision should always be YES since the story is a member of its cluster, but it is retained in the output format so as to maintain format uniformity across different tasks.)
- *Score* is a real number which indicates how confident the system is that the story being processed discusses the cluster event. More positive values indicate greater confidence.

The performance of retrospective detection is evaluated by measuring how well the stories belonging to each of the target events match the stories belonging to the corresponding cluster. This presents a problem, because it is not known which of the clusters corresponds to a particular target event. Thus it is necessary to associate each target event with (exactly) one cluster to determine this correspondence. This was accomplished by associating each target event with the cluster that best matches it. The degree of match between an event and a cluster is defined to be the number of stories that belong to both the event and the cluster.

Note that retrospective detection uses the entire TDT corpus of 15,863 stories, but is evaluated only on the manually labeled stories of 25 events (containing about 7% of the total stories).

[6] There are a total of 1382 non-NO event flags and 1372 flagged stories. (10 stories were flagged by two events.) However, 240 of these stories were flagged as BRIEF, and one was flagged as YES by two events.

On-line New Event Detection The on-line new event detection task is to output a new event flag each time a story discusses a new event. Since evaluation is performed only over the set of target events, the small number (25) of type I trials presents a problem for estimating performance. This problem is addressed by artificially changing the corpus so as to multiply the number of type I trials by N_{skip} (with $N_{skip} = 10$). This is done in the following way:

- The corpus is processed once after deleting all stories with BRIEF event tags.
- The corpus is processed a second time after further deleting the first story which discusses each of the target events.
- The corpus is processed $N_{skip} - 1$ more times, each time further deleting the subsequent first (next) story which discusses each of the target events, until the first N_{skip} stories discussing each of the target events have been skipped.

System output for the on-line detection task will be a declaration for each story processed. This output is to indicate whether or not the story discusses a new event. This information was recorded in a file in ASCII format, one record per story, with records separated by newline characters and with fields in a record separated by white space. Each record has six fields in the following format: "`Event` N_t `Story Decision Score` j_{skip}", where:

- *Event* is an event index number. (Since there is no event affiliation for the on-line detection task, Event is set to zero, but it is retained in the output format so as to maintain format uniformity across different tasks.)
- N_t is the number of stories used to train the system to the event. (Since this is a detection task, N_t is identically zero, but it is retained in the output format so as to maintain format uniformity across different tasks.)
- *Story* is the TDT corpus index number in the range {1, 2, ... 15863} which indicates the story being processed.
- *Decision* is either YES or NO, where YES indicates that the system believes that this story is the first to discuss the event which the story discusses, and NO indicates not.
- *Score* is a real number which indicates how confident the system is that the story being processed is the first to discuss the event. More positive values indicate greater confidence.
- j_{skip} is the number of initial stories that have been skipped for each of the target events, in the range {0, 1, ... N_{skip} }.

Evaluation measures Given a story and a particular event in consideration, the output of a detection system is a YES/NO decision with a confidence score. The performance average over a set of test stories is used to evaluate the detection system. Five evaluation measures are reported in this study: miss rate, false alarm rate, recall, precision, and the F_1 measure. The miss and false alarm rates were the "official" measures of the pilot study.

The F_1 measure[14] was used as a way of balancing recall and precision, in a way that each of them is given equal weight. A more general form of the *F-measure* is $F_\beta(r,p) = \frac{(\beta^2+1)pr}{\beta^2 p + r}$ where β is the parameter allowing differential weighting of p and r. The F-measure is commonly used as an optimization criterion in binary decision making, when recall and precision are considered as the primary performance measures.

In addition to optimizing binary decisions, another objective of the TDT study is the ability to achieve a tradeoff between different types of performance scores at any level desired. A Decision Error Trade-off (DET) curve between misses and false alarms is used for this part of the evaluation.

3.2. The CMU Approach

Given the lack of knowledge about events, event detection is essentially a discovery problem—i.e., *mining the data* for new patterns, in a new paradigm of *query-free* retrieval. CMU takes an approach based on group average agglomerative text clustering, aiming the discovery of natural patterns of news stories over concepts (lexicon terms) and time. This approach creates a hierarchical tree of clusters, with the top layers representing a rough division into general topics, and the lower ones a finer division into narrower topics and events. CMU also investigated an incremental average-link clustering method that produces a single level partition of the TDT corpus.

Incremental clustering For story and cluster representation, CMU uses the conventional vector space model.[13] A story is presented as a vector whose dimensions are the stemmed unique terms in the corpus, and whose elements are the term weights in the story. By "terms" we mean words or phrases in general. A cluster is represented using a *prototype vector* (or *the centroid*) which is the normalized sum of the story vectors in the cluster. For term weighting in a story vector, CMU tested several typical term weighting schemes which combine the within-story term frequency (TF) and the Inverse Document Frequency (IDF) in different ways. As implementation, CMU uses the mechanisms provided in SMART, a benchmarking retrieval system developed by the Salton group at Cornell [13]. The term preprocessing includes removal of stop words, stemming, and then term weighting. The "ltc" option (in the SMART notation) yielded in the best clustering results in the experiments, where the

weight of term t in story d is defined to be:

$$w(t,d) = (1 + \log_2 TF_{(t,d)}) \times IDF_t / \|\vec{d}\|.$$

The denominator $\|\vec{d}\|$ is the 2-norm of vector $\vec{d}$, i.e., the square root of the squared sum of all the elements in that vector. The similarity of two stories is defined as the cosine value of the corresponding story vectors. Similarly, the similarity of two clusters is defined as the cosine value of the corresponding prototype vectors.

Having stories and clusters represented in vectors, the incremental clustering is straightforward. For each consecutive story, compute the cosine similarity of this story and each cluster centroid in the accumulated set. If the similarity score between this story and the closest cluster is above a threshold (pre-selected), then add this story to the cluster as a member, and update the prototype vector correspondingly. Otherwise, add this story as a new cluster in the set. Repeat the above until the corpus is done.

This algorithm results in a flat partition of the TDT corpus. The number of clusters in the partition depends on the clustering threshold in Step 3. When setting the threshold to a value of 0.23, we obtained a partition of 5,907 clusters which yielded the optimal result evaluated using the 25 events labeled by humans (see Section).

Group-average based clustering The core part of CMU's method is an agglomerative algorithm named *Group Average Clustering*[8, 6] which maximizes the average pairwise similarity between stories in each cluster. This algorithm uses the same vector representation for documents and clusters and produces a binary tree of story clusters in a bottom-up fashion: the leaf nodes tree are single-story clusters; a middle-level node is the centroid of the two most proximate lower-level clusters; and the root node of the tree (if the algorithm is allowed to reach this point) is the universal cluster which contains all sub-clusters will all the stories. The GAC algorithm has a quadratic complexity in both time and space, although envisioned improvements based on [15] and other work at CMU should yield sub-quadratic space complexity, without increasing time complexity. In order to reduce the effective complexity and to exploit natural temporal groupings of events in news-streams CMU used the following modified form of GAC clustering:

1. Sort the TDT stories in chronological order, and use this as the initial partition of the corpus where each cluster starts with a single story.
2. Divide the partition (a cluster series) into non-overlapping and consecutive buckets whose size is fixed in terms of the number of clusters they contain.
3. Apply GAC to each bucket, i.e., combine lower-level clusters into higher-level ones in a bottom-up fashion until the bucket size (number of clusters in it) is reduced by a factor of ρ.
4. Remove the bucket boundaries (assemble all the GAC clusters) while reserving the time order of the clusters. Use the resulting cluster series as the updated partition of the corpus.
5. Repeat Step 2-4, until a pre-determined number of clusters is achieved in the final partition.
6. Periodically (say, once per 3 iterations in Step 2-4) flatten each cluster, and apply GAC internally to each flattened cluster for re-clustering. This is CMU's augmentation to Cutting and Pedersen's algorithm. It enables stories belonging to the same event, but initially assigned to different buckets, to be re-assigned to a common cluster.

On-line Detection Algorithm CMU's on-line detection is implemented as below:

1. The algorithm starts with an empty set ("PAST") of clusters, with the pre-determined values for the following parameters:
   - *the detection threshold* which is the minimum score for the system to say that the current story belongs to a new event;
   - *the combining threshold* which is the minimum similarity score for adding a story as a new member of an existing cluster;
   - *the window size* which is the maximum number of clusters in PAST, or the aging limit (in terms of days) of a cluster to be a member in PAST.
2. Read the next story as "the current". Compute the similarity of this story and all the clusters in PAST.
   - If the largest similarity value is above the *detection threshold*, then announce "YES" as the detection of a new event; otherwise, announce "NO".
   - If the largest similarity value is above the *clustering threshold*, then add the current story to the closest cluster, and update the prototype vector of the cluster correspondingly; otherwise, add the current story an a new cluster in PAST, and remove the oldest cluster from PAST if it exceeded the window size.
3. Repeat the above step until the end of the input series.

This algorithm is similar to the incremental clustering algorithm used for retrospective detection (Section), except for two modifications:

- The PAST reference is restricted to a time window of fixed number of stories or days which are closest to the current story, instead of referring an infinite past.
- A detection threshold, independent from the cluster combining threshold, is used to differentiate NEW from OLD.

3.3. The UMass Approach

Retrospective Detection UMass used two different approaches to retrospective event detection:

In the first approach, the TDT collection was examined and all words and noun phrases that occur very often in the collection that do *not* also occur often in a separate training collection were identified as potential triggers for clusters. Each of those terms was then examined to see if its occurrence in documents was heavily concentrated in some small range of time. If not, the term did not trigger an event.

For a term that did trigger an event, all documents containing the term within a time range (determined by the standard deviation of daily occurrence) were handed to a relevance feedback algorithm and a query representing event was created. UMass applied that query to the collection as a whole to find documents that matched the event. A final trimming step removed outlier stories by considering the concentration of stories over a range of days.

The second approach was a bottom-up agglomerative clustering of the documents similar to CMU's. Document similarity was accomplished using the same queries created by on-line detection (described below). Document i and j are compared by running query i against document j, then query j against document i, and averaging the resulting two belief scores. Only document pairs that are more than two standard deviations away from the mean comparison score are eligible to invoke clustering. This provides a stopping criterion for the clustering.

On-Line Detection The UMass algorithm for on-line event detection follows these steps:

1. For each document, extract the n most important features needed to build a query representation of this document.
2. Calculate a belief threshold for this document's corresponding query by running the query against its source document. That belief value is an upper bound on the threshold; it is adjusted downward as described below.
3. Compare the new document against all previous queries. If the document does not exceed the the threshold of an existing query flag the document as containing a new event.
4. If the document exceeds the the threshold of any existing query flag the document as not containing a new event.
5. Save the document's query (and threshold) in the query set.

For the type of query used in this system, InQuery's belief values can range from 0.40 to 1.00. UMass used a threshold above in step 2 that is somewhere between 0.40 and the belief of the document against its own query. We tried various values, but found that values from 20-30% of the way between the two worked well in general, with a lower threshold was more useful with a larger set of n features.

UMass also applied an aging factor to the thresholds: over time, the threshold for matching grew higher and higher. This was meant to model the idea that an event is less and less likely to be reported as time passes—i.e., it slowly becomes news that is no longer worth reporting. UMass found that the aging factor was an important factor in achieving good results.

3.4. Dragon Approach

Dragon's online and retrospective detection systems are applications of the clustering technology used to train background models for the segmenter. As described in the segmentation report, this technology is an implementation of a k-means clustering algorithm.

Online Detection Dragon followed CMU's lead and approached the online detection task as a clustering problem in which the stories being clustered could be examined only once. With this interpretation, online detection is a natural application of k-means clustering, in which one executes only the first pass of the algorithm. Following this procedure, the first story in the corpus defines an initial cluster. The remaining stories in the corpus are processed sequentially; for each one the "distance" to each of the existing clusters is computed. A story is inserted into the closest cluster unless this distance is greater than a threshold, in which case a new cluster is created. The decision to create a new cluster is equivalent to declaring the appearance of a new event.

The old distance measure Given that several iterations of Dragon's implementation of the k-means algorithm produces good clusters for the segmenter, one would expect that the first pass alone would provide a credible basis for an online detection system. This turns out not to be the case. In fact, the performance of Dragon's clustering algorithm in its first iteration turns out to be horrible, essentially dividing the corpus into chunks of about 50 consecutive stories and declaring these to be clusters.

The problem in the first pass arises due to a subtle property of

the distance measure,

$$d = \sum_n (s_n/S) \log \frac{s_n/S}{(c_n + s_n)/(C + S)} + \sum_n (c_n/C) \log \frac{c_n/C}{(c_n + s_n)/(C + S)},$$

where s_n and c_n are the story and cluster counts for word w_n, with $S = \sum s_n$ and $C = \sum c_n$. The two terms have the following interpretation: the first is the distance between the story and the cluster after the story has been inserted into it, and the second is the distance that the cluster itself moves as a result of incorporating the story.

A problem arises for very small clusters: because of the merging of the story and cluster distributions in the denominator of the log, a story can actually "drag" a small cluster close enough that the distance to it is small, and therefore below threshold. Thus whenever a new cluster is created by the clustering algorithm, all subsequent stories are found to be close in distance until the cluster gets big enough (about 50 stories, given our threshold settings), at which point a new cluster is created and the cycle begins again.

The new measure Dragon fixed the measure for the online task by smoothing the cluster distribution used in the distance computation with a background distribution, and then preventing the cluster from being "dragged" by the story distribution. Two improvements were also made: a story-background distance was subtracted from the story-cluster distance (to compensate for the fact that small clusters tend to look a lot like background after smoothing), and a decay term was introduced to cause clusters to have a limited duration in time. This term is just a decay parameter times the difference between the number of the story represented by the distribution s_n and the number midway between the first and last stories in the cluster.

The new measure has the form

$$d = \sum_n (s_n/S) \log \frac{u_n/U}{c'_n/C} + \text{decay term},$$

where c'_n is the smoothed cluster count for word w_n, and u_n is the background unigram count with $U = \sum u_n$.

Tuning the online detection system means adjusting the decay parameter and the overall threshold. Currently these can only be tuned on the test corpus.

3.5. Results, Analysis, and Future Work

The three sites have obtained results for retrospective and on-line detection, evaluated using the various metrics discussed, including F1 and DET curves. Tables 2, 3 and 4 list the reported results for several of the runs from the various sites. Figure 1 shows the DET curves of the best online runs, one for each site. Figures 3, 2 and 4 are the DET plots of retrospective detection systems.

CMU optimization efforts In order to optimize results, CMU is investigating the following: dealing with out-of-vocabulary (OOV) terms; incremental updating of IDF; using time windows and declining weighting factors; dynamically setting Clustering thresholds; and, unsupervised incremental learning.

The incremental updating of Inverted Document Frequency (IDF) is defined to be:

$$IDF_{(x,t)} = \log_2(N_{(x)}/n_{(x,t)})$$

where t is a term, x is the current story, $N_{(x)}$ is the number of stories in the sequence from the first story in the corpus (TDT or JGI+TDT) to the current point x, and $n_{(x,t)}$ is the number of stories which constrain term t in the sequence to the current point x.

In terms of using time constraints in on-line detection, CMU tried two methods. The first method was to use a time window of k stories, denoted as W_k, which is prior to the current story. The detection decision on the correct story, x, is based on the comparison of this story with each story in the window:

$$score(x) = 1 - \max_{d_i \in W_k} \{cos(\vec{x}, \vec{d_i})\}$$

Another method was to use a decaying weighting function to adjust the influence of stories in the window. The $score(x)$ in this method is modified as

$$score(x) = 1 - \max_{d_i \in W_k} \{\frac{i}{k} cos(\vec{x}, \vec{d_i})\}.$$

This modification makes the decision rely more on the stories which are closer to the current time, than the stories far in the past. In other words, it is a smoother way to use a time window than a uniformly weighted window. CMU found that a window size of 700 is about optimal when not using the decaying weighting function, and a size of 2500 optimal when using the decay weighting. The relative improvement from using decaying weights is about 2% in the F_1 measure over a fixed window.

UMass optimization efforts The word-trigger approach provided reasonably high-precision clusters, but realized bad recall: the cluster sizes were too small. UMass believes that the recall can be improved by relaxing some constraints.

For the bottom-up agglomerative approach, UMass found the unsurprising result that higher-dimensionality query representations were more effective. 100- and 50-term queries were noticeably more effective than 10-term queries, in the same way that they were for on-line detection. However, in this case the 50-term queries outperformed the 100-term queries.

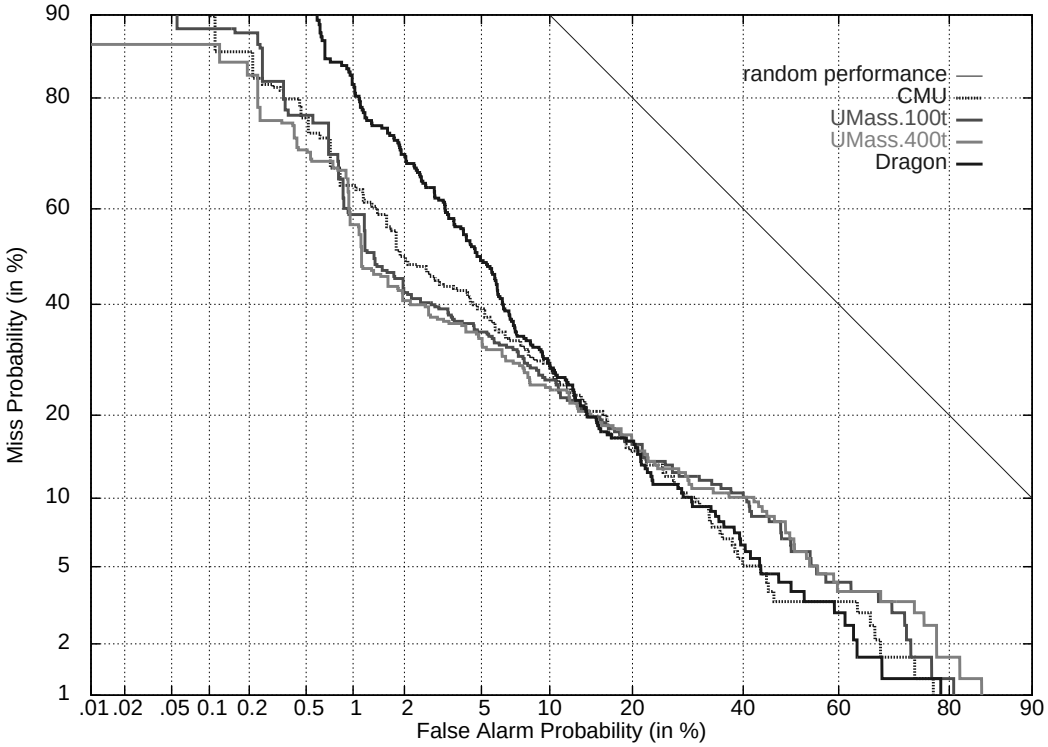

Figure 1: TDT On-line Detection Runs

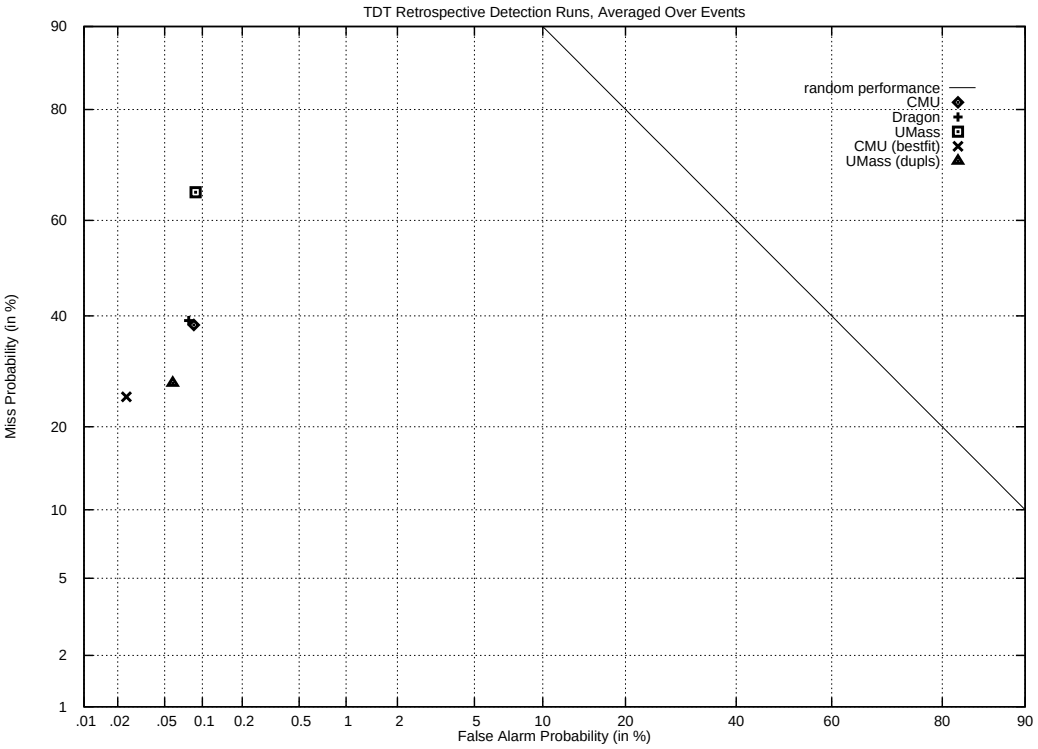

Figure 2: TDT Retrospective Detection Runs, Averaged Over Events

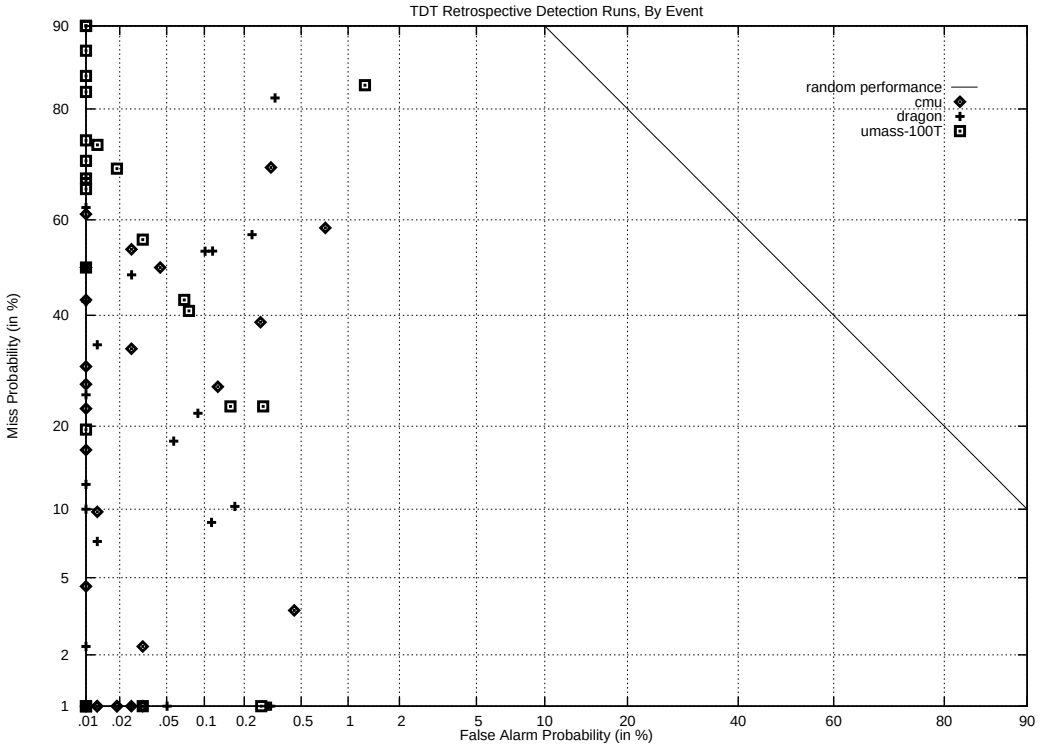

Figure 3: TDT Retrospective Detection Runs, By Event

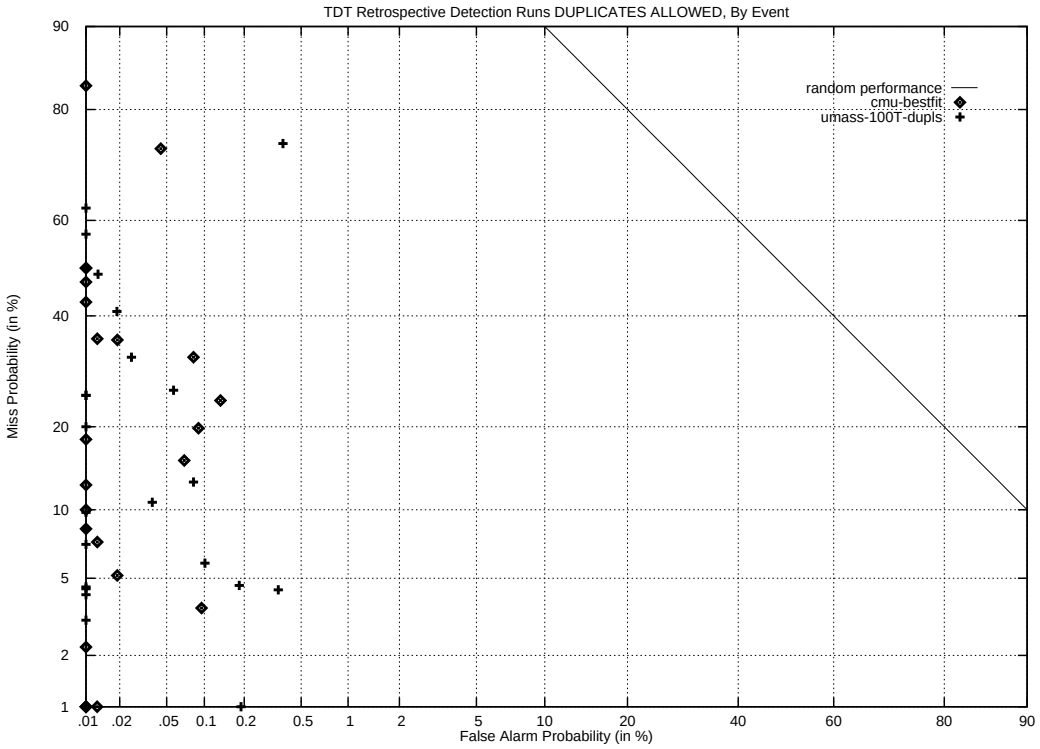

Figure 4: TDT Retrospective Detection Runs, *duplicates allowed*, By Event

Run	%Miss	%f/a	%Recall	%Prec	micro-avg F1	macro-avg F1
CMU incremental	38	0.09	62	67	0.64	.77
CMU gac top-level	17	0.32	83	43	0.56	.63
Dragon	39	0.08	61	69	0.65	.75
UMass 100T	66	0.09	34	53	0.42	.60
UMass 10T	67	0.50	33	16	0.21	.53

Table 2: Retrospective Detection Results: Partition Required. (Official evaluation)

For retrospective detection, UMass did only a small amount of work with alternate types of features (e.g., phrases) for these experiments; preliminary results suggest that multi-word features are helpful.

Similar to CMU's time-windows UMass found that the time sensitive nature of event reporting can be captured by aging the belief thresholds. For a given documents' query, UMass raised the threshold incrementally as each subsequent story was processed, making it more and more difficult for later stories to pass the threshold. This aging of the thresholds provided substantial improvements in precision without impacting recall noticeably. Note, however, that the aging helps performance of unexpected events (e.g., disasters) but hurts performance of long-running events such as the O.J. Simpson trial.

Dragon's Optimization Directions Dragon believes that further careful research on the clustering measure can produce performance gains in its system. The fact that the retrospective evaluation indicates that the new distance measure does better than the old one suggests that the clustering of the background topics used by the segmenter should be revisited, and the segmentation experiments rerun with topics based on the new measure. This is an area for future work.

3.6. Open Issues

Some related issues pertaining to event detection have not been addressed in the pilot TDT study, but evolve naturally therefrom, including:

- How to provide a global view of the information space to users and navigation tools for effective and efficient search?
- Some approaches generate a cluster hierarchy automatically. How to choose the right level of clusters for user's attention that best fits the information need of the user?
- How to summarize the information at different degrees of granularity, i.e., at a corpus level, a cluster level, a story level, and a sub-story level? How to provide user-specific or query-specific summaries? How to remove redundant parts and maximize the information in a summary?
- How to make a better use of temporal information in event detection and tracking than we have done? In the case of on-line detection, for example, we have only taken the simplest approach of imposing a time window to the data stream.
- How to improve the accuracy of on-line detection by introducing limited look-ahead? For instance, noting that two or three stories arriving very close in time are highly related to each other but different than anything in a previous time interval would be a very good indicator of a new breaking event.

4. Event Tracking

The TDT event tracking task is fundamentally similar to the standard routing and filtering tasks of Information Retrieval (IR). Given a few sample instances of stories describing an event (i.e., stories that provide a description of the event), the task is to identify any and all subsequent stories describing the same event. Event tracking is different from those IR tasks in that events rather than queries are tracked, and in that events have a temporal locality that more general queries lack. These differences shift the nature of the problem slightly but at the same time shift the possible solutions significantly. The narrowing of the scope of information filtering encourages modifications to existing approaches and invites entirely new approaches that were not feasible in a more general query-centric setting.

This report discusses approaches to Event Tracking by research teams from CMU, the University of Massachusetts, and Dragon Systems.

4.1. Tracking evaluation

Each event is to be treated separately and independently. In training the system for any particular target event, allowable information includes the training set, the test set, and event flags for that target event only. (No information is given on any other target event).

Evaluation will be conducted for five values of N_t , namely

Run	%Miss	%f/a	%Recall	%Prec	micro-avg F1	macro-avg F1
CMU gac hierarchy	25	0.02	75	90	0.82	.84
UMass 100T-dups	27	0.06	73	78	0.75	.81
UMass 10T-dups	31	0.05	69	80	0.74	.81

Table 3: Retrospective Detection Results: Duplicates Allowed.

$\{1, 2, 4, 8, 16\}$. In training, N_t will count just the number of YES tags for the target event and will exclude the BRIEF tags. However, the full classification of each story in the training set (i.e., YES, BRIEF, and NO) may be used in training.

All of the stories in the test set must be processed, but there will be two evaluations – one over all of the test data (for each value of N_t), and one over a fixed set of test data, namely the test set corresponding to $(N_t)_{Max} = 16$. (Tabulating performance for a fixed test set as N_t varies will allow a more stable comparison of performance across the various values of N_t , because the test set will be the same for all values of N_t .)

The test stories are to be processed in chronological order, and decisions must be made "on line". That is, the detection decision for each test story must be output before processing any subsequent stories. Decisions may not be deferred.

The event tracking system may adapt to the test data as it is processed, but only in an unsupervised mode. Supervised feedback is not allowed. (Evaluating over a set of N_t's provides essentially equivalent information.)

In calculating performance, those stories tagged as BRIEF for the target event will not be included in the error tally.

There will be a TDT tracking trial for each story, and for each event, and for each value of N_t . This will make a total of about 1,000,000 trials. This number is derived by multiplying the number of target events (25) by the average number of test stories (assumed to number about 8,000) by the number of values of N_t (5). Of these trials, less than one percent will be type I (true) trials.

For each trial, there will be two outputs - first, a logical detection indication, YES or NO, indicating whether the system believes that the story discusses the target event; and second, a score indicating how confident the system is in this decision. This confidence indicator will be used to compute a detection error trade-off (DET) between misses and false alarms.

The trials for the tracking task will be recorded in a file in ASCII format, one record per trial, with trials separated by newline characters and with fields in a record separated by white space. Each record will have 5 fields in the format, "`Event` N_t `Story Decision Score`", where:

- *Event* is an index number in the range $\{1, 2, \ldots, 25\}$ which indicates the target event being detected.
- N_t is the number of stories used to train the system to the target event.
- *Story* is the TDT corpus index number in the range $\{1, 2, \ldots 15863\}$ which indicates the story being processed.
- *Decision* is either YES or NO, where YES indicates that the system believes that the story being processed discusses the target event, and NO indicates not.
- *Score* is a real number which indicates how confident the system is that the story being processed discusses the target event. More positive values indicate greater confidence. (Large negative numbers indicate lack of confidence, while large positive number indicate high confidence.)

Indirect Evaluation of Segmentation Segmentation (see Section) will be evaluated indirectly by measuring event tracking performance on stories as they are defined by automatic segmentation means. A straightforward three-step procedure will be used:

1. Segment the whole corpus using the segmentation system under test.
2. Using an event tracker that has been evaluated on the TDT corpus previously, run this system on the auto-segmented corpus. Follow the standard event tracking rules, with the following exceptions:
   - Train the event tracking system on the original correctly segmented stories.
   - Begin evaluation on the first auto-segmented story which follows the last training story and which is disjoint from it.
3. Evaluate the event tracker results and compare these results with results on the original correctly segmented stories.

The evaluation is complicated by the fact that there are no event flags for the auto-segmented stories. This problem will be solved by creating scores for the original stories from those

Run	%Miss	%f/a	%Recall	%Prec	micro-avg F1	macro-avg F1
CMU decay-win2500	59	1.43	41	38	.40	.39
CMU fixed-win700	55	1.80	45	35	.39	.39
Dragon	58	3.47	42	21	0.28	.28
UMass 100T	50	1.34	50	45	0.48	.45
UMass 50T	51	1.31	49	45	0.47	.47
UMass 10T	59	1.19	41	43	0.42	.42
UMass 10T notime	73	1.53	27	28	0.28	.27

Table 4: On-line Detection Results: Average Over 11 Runs.

computed for the auto-segmented stories.[7] The evaluation will then be performed on the original stories using these synthetic scores. The synthetic score for each original story will be a weighted sum of the scores for all overlapping auto-segmented stories, where the weight prorates each score according to how many words the overlapping story contributes:

$$Score_{orig(i)} = \frac{\sum_{j \in overlap(i)} w_{ij} \cdot Score_{autoseg(j)}}{\sum_{j \in overlap(i)} w_{ij}}$$

where w_{ij} is the number of words in the j^{th} auto-segmented story that overlap with the i^{th} original story.

The output record format will be the same as for the conventional event tracking task. The decision will be computed in the standard way and will be based on the synthetic score.

Speech Tracking – the TWA 800 crash event In addition to the TDT study corpus, an additional corpus will be processed to explore the tracking task for different representations of speech, including machine recognition of speech. The corpus consists of CNN recordings and spans the period during 1996 when the TWA 800 crash occurred. This corpus contains a total of 1029 stories, of which 35 discuss the TWA crash. Two different representations of the speech will be processed: (1) Closed captioning taken from the CNN broadcast; and, (2) Speech recognition output provided by CMU. There were no JGI transcripts for the TWA corpus, so there is no "accurate" representation of the speech.

4.2. UMass approach

All efforts by UMass to attack this problem have focused on its similarity to information filtering. For that reason, UMass used the training data (positive and negative) to create a short query intended to represent the event being tracked. The training data were also used to derive a threshold for comparison with that query. That query was applied to all subsequent stories—if they matched the query, they were "tracked."

UMass tried two approaches. The first was based on simple "relevance feedback" methods of IR. The N_t positive training examples and up to $100N_t$ negative training examples were handed to a relevance feedback routine that built queries of 10 to 100 words that were intended to represent the event. The query was run against the training set to select a threshold.

A second approach used a shallow parser to extract nouns and noun phrases (rather than all single terms), weighted features in two different ways—one that gave features a higher weight if they occurred frequently within at least one training story, and the other that weighted features based upon the number of training stories it occurred in.

4.3. CMU approach

CMU developed two methods for tracking events: a *k-Nearest Neighbor* (kNN) classifier and a *Decision-Tree Induction* (dtree) classifier.

kNN is an instance-based classification method. All training instances (positive and negative) of each event being tracked are stored and efficiently indexed. The approach proceeds by converting each story into a vector as it arrives and comparing it to all past stories. The k nearest training vectors (measured by cosine similarity) to the incoming story each "vote" for or against the new story, based on whether or not they are themselves members of the event. For a binary decision, we set a threshold on the scores to vote YES or NO for d_i being an instance of the tracked event. For instance, vote YES iff score > 0.

CMU ran some variation on kNN in an effort to find approaches that yielded high-quality results across the entire miss vs false-alarm tradeoff spectrum, as exhibited in the DET curves in the evaluation section. The alternate approaches were based upon using *two* nearest-neighborhoods (not necessarily of the same size), one for positive instances of the event and one for negative, computing scores S^+ and

[7] There are two possible ways of solving this problem - either by mapping the event flags for the original stories onto the auto-segmented stories, or by mapping the decisions on the auto-segmented stories onto the original stories. Mapping event flags onto the auto-segmented stories might seem to represent the actual application scenario more accurately. Mapping scores was chosen, however, to facilitate a clearer comparison with results on the original stories and to avoid conceptual and mechanical difficulties involved in mapping the event flags.

S^- respectively, using the same similarity-weighted voting as before. The overall score was a linear combination or a ratio of the two neighborhoods' scores.

Decision Trees Decision trees (dtrees) are classifiers built based on the principle of a sequential greedy algorithm which at each step strives to maximally reduce system entropy. Decision trees select the feature with maximal information gain (IG) as the root node, dividing the training data according to the values of this feature, and then for each branch finding the next feature f_k such that $IG(S(f_j), f_k|f_j)$ is maximized, and so on recursively. Decision trees are typically good when there are sufficient training instances belonging to each class. One disadvantage of dtrees is that they cannot output continuously varying tradeoff scores and thus are unable to generate meaningful DET curves (some efforts were made to produce DET curves from the dtrees, but they were not highly successful).

4.4. Dragon approach

Dragon's event tracker is an adaptation of its segmenter, which is described in more detail in the segmentation report. As discussed there, the segmentation algorithm does segmentation and topic assignment simultaneously. In general, the topic labels assigned by the segmenter (which are drawn from the set of automatically derived background topics) are not useful for classification, as they are few in number and do not necessarily correspond to categories a person would find interesting. However, by supplementing the background topic models with a language model for a specific event of interest, and allowing the segmenter to score segments against this model, it becomes possible for the the segmenter to output a notification of an occurrence of that event in the news stream whenever it assigns that event model's label to a story. In this implementation, the topic models have the role of determining the background against which the event model must score sufficiently well to be identified.

In this incarnation, the segmenter is not asked to identify story boundaries. Its job is merely to score each story against its set of background models, as well as against the event model, and report the score difference between the best background model and the event model. A threshold is applied to this difference to determine whether a story is about the event or not, and this threshold can be adjusted to tune the tracker's output characteristics. For example, a low threshold means that a story does not have to score much better in the event model than it does in the best background model (or, perhaps, may even fail to score as well by a specified amount) for it to be declared an instance of the event. This tuning tends to result in missing very few stories on the event, but probably will generate a high number of false alarms.

Event models were built from the words in the N_t training stories, after stopwords were removed with some appropriate smoothing. In this case, in order to provide a more accurate smoothing for the event model, we take as the backoff distribution the mixture of the background topic models that best approximates the unsmoothed event model. There is therefore a different backoff model for every event and every value of N_t.

Run	%Miss	%F/A	F1	% Prec
CMU kNN	29	0.40	0.66	61
Dragon	71	0.12	0.39	60
UMass nonRF-comb	55	0.10	0.60	88
UMass nonRF-20T	13	2.35	0.41	27
UMass RF100	39	0.27	0.62	62

Table 5: Tracking results for $N_t = 4$, pooled average across all 15 events evaluated (evaluation at $N_t = 16$). (Note that recall is 1 minus the miss rate.)

4.5. Evaluation methodology

The first 16 stories on each event[8] are set aside for training purposes. A system's ability to track events is tested on all stories from the one immediately following the 16^{th} training story through the end of the TDT corpus. Note that this means that the test sets for each event are different.

A system is evaluated based on varying amounts of training data. Each system is allowed N_t positive training examples, where N_t takes on values 1, 2, 4, 8, and 16. The system is permitted to train on all N_t positive stories as well as all stories that occur in the corpus *prior* to the N_t^{th} story. Note that the training subset may include some stories that were judged BRIEF for a particular event; those stories may be used (along with the knowledge that it was judged BRIEF). The test set is always the collection minus the $N_t = 16$ training data.

Evaluations may be averaged across events within N_t values. It is not particularly meaningful to average across N_t values. Results are reported using the standard TDT evaluation measures and the Detection Error Tradeoff curves.

4.6. Evaluation results

Basic results Table 5 lists the reported results for several of the runs from the various sites. These report the exact evaluation of error rates at the thresholds chosen by the sites, averaged across all events, at $N_t = 4$ (averaging is by pooling all the results), but evaluated using the $N_t = 16$ test set. The table shows that the sites are able to generate results that vary widely in their error rates. The preferred run from UMass is

[8] Only stories that are judged YES count; those judged BRIEF do not count as part of the 16, nor as part of the test set.

nonRF-comb. Comparing it to the run from the other sites shows dramatic differences in miss rates. The UMass and Dragon runs have similar false alarm rates (the 0.02% difference is the difference between 4 and 7 false alarms per event on average). CMU's substantially lower miss rate comes at the expense of a much larger false alarm rate.

These results are not particularly surprising. CMU tuned its decision points on the false alarm/miss tradeoff based upon the F1 measure that attempts to balance recall and precision values. UMass, on the other hand, tuned its approach using average precision numbers. The effect is clear in the numbers, where UMass achieves very high precision for the task, but CMU attains a better balance between the two.

DET curves Figure 5 shows the DET curves for three sample runs, one from each site. To make some comparison possible, only the $N_t = 4$ run is given for each. A single point is plotted on the curve to represent the specific detection error tradeoff made by the threshold values each system chose. The detached point is associated with the CMU decision tree approach: this is the result of a confidence threshold that does not entirely conform to the YES/NO decisions made for tracking.

The UMass RF2 run and the Dragon run are very similar in effectiveness. The graph shows only $N_t = 4$, but when all N_t values are plotted, the UMass run turns out to be the quickest approach to converge to "good" values as N_t increases. In fact, the use of additional training stories appears to *harm* the overall tradeoff between the errors. UMass hypothesizes that the stability is a result of using noun phrases as features. Dragon's event models did not work as well with very small values of N_t

The UMass RF run performs less well, primarily because it uses a small number of features. It is shown to make it clear that minor variations in the query formation process can result in substantial differences in effectiveness.

The CMU k-NN run is fairly insensitive to N_t at low false alarm rates, but when the miss rate drops below 10%, the training instances become more and more important. This result is not surprising, because as the size of the neighborhood needed to match grows (in order to reduce the miss rate), it is very likely that mismatches will occur and the supporting evidence of other training examples will help prevent that. (It is more surprising that UMass's run does not degrade in this fashion, than that CMU's does.)

The CMU decision tree approach results in an unusual DET curve because its decisions result in a very small number of confidence scores. When the curve makes huge jumps to the right, that indicates a large number of stories with the same confidence value: when the threshold hits that point, all stories at that value get included and the false alarm rate leaps. The decision tree approach is tuned to a specific point on the curve: here, the leftmost knee at about 0.1% false alarm rate was the goal.

	N_t value				
	1	2	4	8	16
Dragon	-55%	-26%	–	+12%	+40%
CMU, Dtree	-90%	-30%	–	+12%	+15%
CMU, kNN	-25%	-9%	–	+11%	+32%
UMass	-39%	-5%	–	+5%	+5%

Table 6: Shows changes in pooled F_1 measure for several systems as N_t varies, with $N_t = 4$ as the baseline. Actual effectiveness numbers for $N_t = 4$ are reported in Table 5.

The DET curves also show the tradeoffs that each site made for selecting a threshold for YES/NO decisions. UMass and Dragon both show decision points in the extreme upper left of the curves, reflecting an emphasis on precision or low false alarms. CMU, on the other hand, selected points much closer to the middle of the graph, illustrating their goal of balancing recall and precision.

Varying Values of N_t The results presented above are at a single value of N_t (i.e., four). That limited presentation simplifies some points of comparison, but also ignores the interesting question of how the number of training stories (N_t) affects performance. Rather than present DET curves for every run discussed in the previous section, we will consider just the increase in effectiveness each system achieves as N_t changes.

Table 6 shows the impact that varying N_t has on the effectiveness of the systems, as measured by pooled values of F_1, a measure that balances recall and precision. Only a few of the systems made an effort to optimize for F_1 values, so

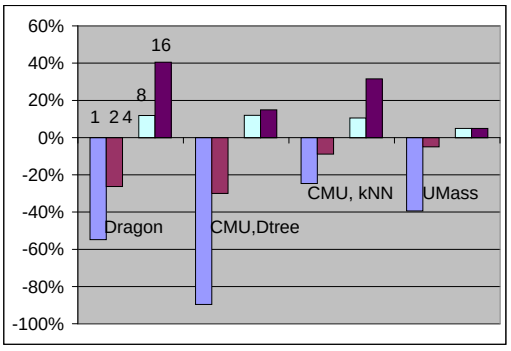

Figure 6: Graph of data in Table 6, showing impact of various values of N_t on a pooled F_1 measure for several systems.

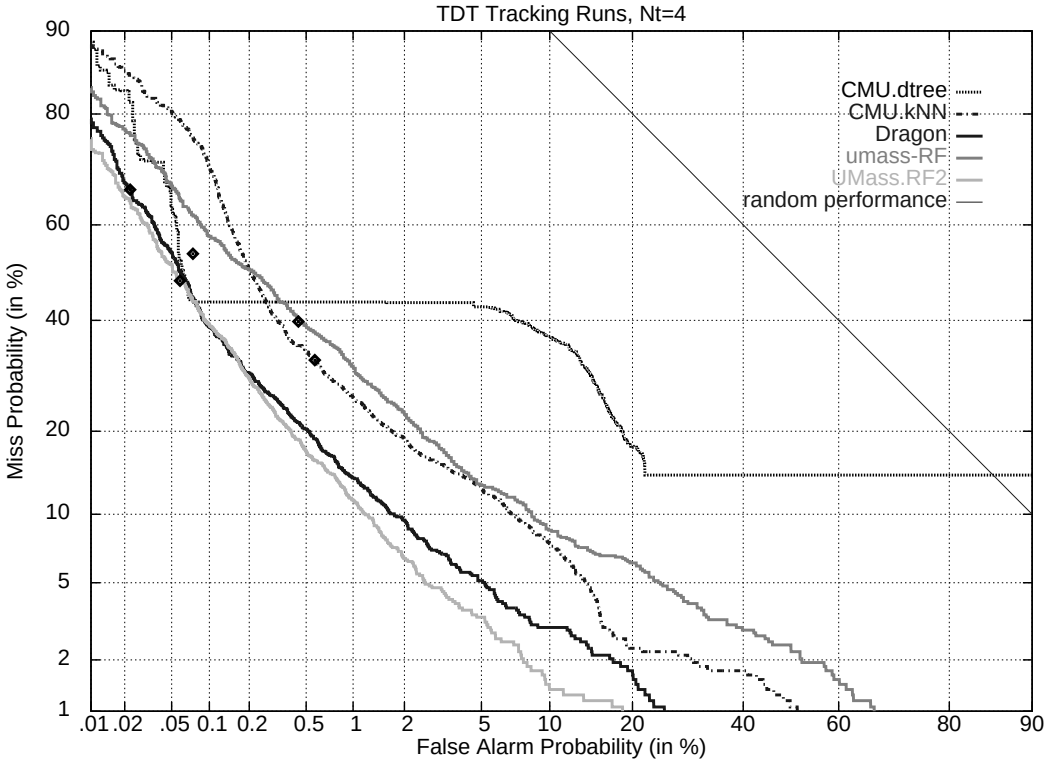

Figure 5: DET curve for sample tracking runs from each site. All runs were performed with $N_t = 4$ training stories, and evaluated at $N_t = 4$.

the actual effectiveness numbers are less important than the changes that varying N_t causes. For that reason, only percent changes from the $N_t = 4$ value are reported (Table 5 reports the baseline effectiveness for that N_t value for those who are concerned).

It is clear from the figure that the decision tree approach is extremely sensitive to the amount of training data. At $N_t = 1$, it has very poor effectiveness, but it learns rapidly, with only modest gains after four training instances. The Dragon approach and the kNN approach both show more consistent gains with each additional training example, though Dragon's approach appears to continue to benefit from learning the most. The UMass approach stands out as the most stable after $N_t = 2$ training instances: it learns a good event representation very rapidly and gains almost nothing in effectiveness beyond that point.

4.7. Indirect Evaluation of Segmentation

Figure 7 shows a comparison of a tracking run done with actual TDT stories and one with stories generated by a segmentation run. The runs were both done by Dragon, though are based on an earlier version of their tracker than that presented in Figure 5, so the baseline performance is slightly lower.

The two runs are nearly identical, except that the segmented corpus has noticeably degraded performance below a false alarm rate of 0.04% and above a rate of roughly 15%. It shows a modest loss in effectiveness in the 2-15% range. Another indirect evaluation done by UMass (not presented here) showed a similar effect, except that the degradation was slightly larger and consistent everywhere except in the 10-20% false alarm rate were the two runs were almost identical.

Those two sets of runs suggest that segmentation of the quality reported in Section has only a modest impact on tracking effectiveness.

4.8. Speech-Recognized Audio Transcripts

The tracking methods developed and discussed above worked relatively well for accurate transcripts and newswire stories: transcribed CNN and Reuters, respectively. However, can tracking also be performed on a much noisier channel, such as the automatically-recognized audio track of broadcast news? If so, at what price in accuracy? In order to investigate these questions the CMU Informedia group provided TDT with about 1,000 CNN news stories, including their close captions and their speech-recognized audio. Speech recognition was performed with the SPHINX-II system, which generates about 50% word accuracy for raw broadcast news. The low accuracy of CSR is due in part to the quality of the news audio (often there is background interference: music, noise, other voices) and a significant number of out-of-vocabulary words.

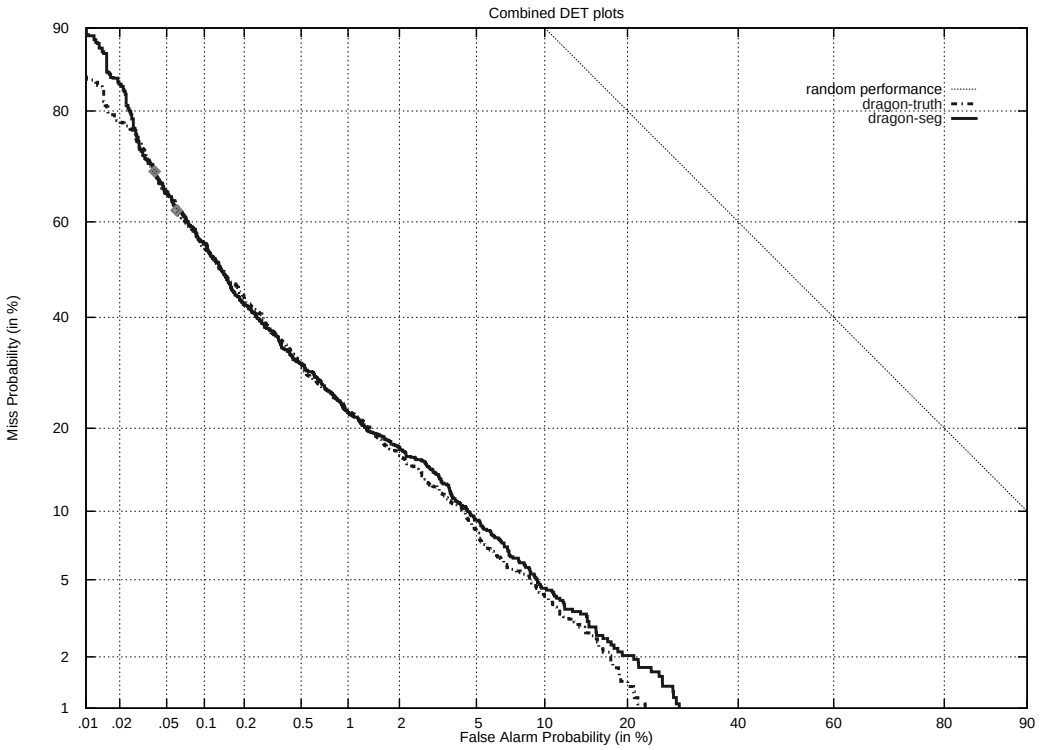

Figure 7: DET curves for an early Dragon tracking run using true story boundaries compared to the same run using calculated segment boundaries.

One event, the crash of TWA flight 800, was heavily represented in this small corpus, and was used for the preliminary investigation. All the TWA-800 events were hand-labeled by Informedia, and made available to the TDT research groups. CMU tried both kNN and dtrees on close-captions and speech data. Dragon (with less preparation time than CMU) also tried their trackers, producing the following summary results, as measured by F1 micro-average across N_t values.

Classifier	CC transcript	CSR Output
D-tree	0.34	0.20
kNN	0.31	0.21
Dragon	0.29	0.08

These results indicate a drop in accuracy from perfect transcripts to imperfect close captions, and a further drop in accuracy to speech recognized audio. However, tracking still works under conditions of 50% error rates. This is encouraging as speech recognition accuracy on broadcast news will only improve, and tracking technology will also improve. Moreover, the latter may be tunable to different expected noise levels in the input channel. These results are quite preliminary, especially given the small data set and single-event tracking.

4.9. Analysis and Conclusions

The tracking task is difficult to analyze because it is somewhat vaguely stated. There was no preference specified for minimizing misses or false alarms, so it was difficult for any of the sites to tune their systems appropriately. (The DET curve shows a tradeoff for a particular tracking algorithm not across algorithms.)

For this task, the events had an average of 54 stories that could have been tracked, and an average of 8377 stories that should *not* have been tracked. Achieving a miss rate of 50% and a false alarm rate of less than 0.25% would mean 27 correctly tracked, and less than 20 incorrectly tracked.

As reported in Table 5, each of these systems falls into approximately that range, but just barely. At a miss rate of 50%, the systems achieve about 0.15%, 0.20%, and 0.05%, meaning that on average 12, 16, or 4 uninteresting stories are tracked in order to get 27 interesting ones. To that extent, then, these systems are successful.

However, for low-miss (high-recall) applications, these results are less impressive. At a miss rate of 20% (43 of 54 tracked), anywhere from 42 to 320 false alarms will arise (assuming at least 2 training examples).

These numbers are not out of line with typical IR ranked re-

trieval tasks, though the comparison is not necessarily obvious. 50% precision at 80% recall would be quite good for a search system, and suggests that this problem or this corpus is simpler than basic IR.

What works The tracking task works by creating some form of model of the event being tracked. The above experiments suggest the following:

- If the event is modeled by a set of single terms (and weights), the evidence indicates that 20-50 terms is preferable. Smaller sets of terms provide higher precision, but do not cast a wide enough net to bring in much relevant material. Very large sets appear to cast too wide and undifferentiated a net, bringing in more relevant stories, but swamping it with unrelated material.
- A better set of features (e.g., noun phrases) is even more effective at producing high quality results. UMass believes that it is the higher quality features used in its nonRF-comb20 run that gave it superior performance.
- Combining multiple approaches to deciding that a story should be tracked can be helpful. The evidence combination applied by UMass substantially stabilized the algorithm's handling of very small numbers of training stories.
- One idea addresses the problem of small event models. To smooth an event model consisting of one story, use that story as a query into a training database, and use the stories retrieved as smoothing material. Given that Dragon's performance improves rapidly with more training examples, this might dramatically improve the behavior of the system at small N_t. In general, Dragon believes that this task requires a smoothing algorithm that aggressively preserves topic, something that is much more suited to information retrieval techniques.

These results are consistent with IR searching results and are not particularly surprising for that reason. However, it does mean that methods that have helped IR are likely to help in this task, too: for example, query expansion techniques based on pseudo-relevance feedback may be a fruitful means of addressing the problem of tracking with a very small set of positive stories. For example, the following may be appropriate areas to explore: (1) evidence combination beyond that explored briefly by UMass; unsupervised learning; interactive tracking (supervised learning).

This study has shown that fairly simple techniques can achieve very high quality results, but that substantial work is needed to reduce the errors to manageable numbers. Fortunately, that the TDT problem focuses on Broadcast News and not on arbitrary forms of information, means that there is hope that more carefully crafted approaches can improve the tracking results substantially.

5. Conclusions

This section presents some broad conclusions that can be drawn from the Topic Detection and Tracking pilot study. It was not known at the start of the TDT pilot study whether the state of the art could effectively and efficiently address any of the TDT tasks. The conclusions below show that the technologies applied solve large portions of the problem, but leave substantial room—and hope—for improvement.

The success of existing approaches has two implications. First, because quick efforts yielded good results, continued and more concentrated work on these problems is likely to yield even better results. Second, because the current approaches are adequate, it is possible to move forward and investigate the more complicated problems suggested by TDT: handling of degraded text (from automatic speech recognition), differences between "topics" and "events," building descriptions of the events being tracked or detected, and so on.

General conclusions. The reporting pattern for a typical event is a rapid onset followed by a gradual decline over period ranging from 1 week to 4 weeks. Some events "re-ignite" (such as Hall's Helicopter, upon his release and homecoming). A few atypical events are "sagas" with sporadic reporting over long periods (such as OJ's DNA).

Segmentation conclusions. Segmentation is a tractable task using known technologies (HMM, IR, machine learning). This fact was not at all certain when the pilot study began.

Segmentation is possible by several methods, each of which has strengths and weaknesses. This suggests (a) that future work will yield improvements as different ideas are merged, and (b) different kinds of segmentation problems can be addressed.

The tracking task shows negligible degradation when applied to segmented text rather than "correct" segmentation, suggesting that automatic segmentation technologies may require little improvement *for this task*.

Detection conclusions. Pure retrospective detection can be performed quite reliably for most events (except OJ, etc.) by clustering methods, with significant differences attributable to the clustering methods used. Permitting overlapping clusters improves performance over strict partitions, though presents some evaluation concerns.

Online detection cannot yet be performed reliably. Whereas the onset of some events are detected well, others (e.g., different airline disasters) are confused with earlier similar events and thus frequently missed. Further basic research is needed, not just tuning or incrementally improving existing methods.

Intermediate points of detection, such as on-line with a variable deferral period offer interesting intermediate solutions

between retrospective and immediate detection.[9]

Tracking conclusions. Tracking is basically a simpler version of the classic Information Retrieval (IR) "filtering" task, but one should *not* therefore conclude that it is uninteresting because it is already "solved". Rather, the fact that it lies in a slightly more restricted domain than IR deals with, means that some more domain-specific techniques can be applied (from IR, speech, and machine learning) to possibly give better performance than one might expect from unrestricted approaches.

Tracking of typical events can be accomplished fairly reliably if at least 4 instance documents are provided. Some events can be tracked with fairly reliably with only 1 or 2 training instances.

Different technologies for tracking (kNNs, decision trees, probabilistic queries, language-model differentials, etc.) show remarkably similar performance on aggregate, but substantial differences on specific events.

Degraded text conclusions. A preliminary study by CMU and Dragon indicated that tracking with automated speech recognition output may prove more difficult than with perfect transcriptions, especially with small numbers (under 4) training instances. Results show a 50% or more drop in effectiveness,[10] suggesting that this area is ripe for further research. Note that the TDT2 study will focus centrally on CSR-generated text for segmentation, detection, and tracking.

References

1. D. Beeferman, A. Berger, and J. Lafferty, *A model of lexical attraction and repulsion*, In Proceedings of the ACL, Madrid, Spain, 1997.
2. A. Berger, S. Della Pietra, and V. Della Pietra, *A maximum entropy approach to natural language processing*, Computational Linguistics, 22(1):39–71, 1996.
3. A. Bookstein and S.T. Klein, *Detecting content-bearing words by serial clustering*, Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 319–327, 1995.
4. S. Della Pietra, V. Della Pietra, and J. Lafferty, *Inducing features of random fields*, IEEE Trans. on Pattern Analysis and Machine Intelligence, 19(4):380–393, April 1997.
5. W.B. Croft and D.J. Harper, *Using probabilistic models of document retrieval without relevance information*, Journal of Documentation, 37:285–295, 1979.
6. D.R. Cutting, D.R. Karger, J.O. Pedersen, and J.W. Tukey, Scatter/Gather: a Cluster-based Approach to Browsing Large Document Collections, In 15th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'92), 1992.
7. M.A. Hearst, *Multi-paragraph Segmentation of Expository Text*, in Proceedings of the ACL, 1994.
8. T. Feder and D. Greene, Optimal Algorithms for Approximate Clustering. In Proceedings of the 20th Annual ACM Symposium on the Theory of Computing (STOC), pp. 434-444, 1988.
9. S. Katz, *Estimation of probabilities from sparse data for the language model component of a speech recognizer*, IEEE Transactions on Acoustics, Speech and Signal Processing, ASSP-35(3):400–401, March, 1997.
10. H. Kozima, *Text Segmentation Based on Similarity between Words*, in Proceedings of the ACL, 1993.
11. D.J. Litman and R.J. Passonneau, *Combining Multiple Knowledge Sources for Discourse Segmentation*, in Proceedings of the ACL, 1995.
12. J.M. Ponte and W.B. Croft, *Text Segmentation by Topic*, in Proceedings of the First European Conference on Research and Advanced Technology for Digitial Libraries, pp. 120–129, 1997.
13. G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley, 1989.
14. C.J. van Rijsbergen, Information Retrieval (2nd edition), Butterworths, London, 1979.
15. E.M. Voorhees, Implementing agglomerative hierarchic clustering algorithms for use in document retrieval, *Information Processing & Management*, **22**:6, 465-476, 1986.
16. J. Xu and W.B. Croft, *Query Expansion Using Local and Global Document Analysis*, in Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 4–11, 1996.

[9] The TDT2 study will focus on this mode of detection.

[10] Effectiveness numbers should be viewed skeptically because of the very small sample size of the test corpus.