Learning approaches for Detecting and Tracking News Events

(LaTex version: with minor differences from the IEEE-formatted version)

Yiming Yang, Jaime Carbonell, Ralf Brown,
Tom Pierce, Brian T. Archibald, Xin Liu
{yiming,jgc,ralf,tomp,ba2e,xliu}@cs.cmu.edu
Language Technologies Institute, Carnegie Mellon University,
Pittsburgh, PA 15213-3702, USA

Abstract

This paper studies the effective use of information retrieval and machine learning techniques in a new task, event detection and tracking. The objective is to automatically detect novel events from chronologically-ordered streams of news stories, and track events of interest over time. We extended existing supervised learning and unsupervised clustering algorithms to allow document classification based on both information content and temporal aspects of events. A task-oriented evaluation was conducted using Reuters and CNN news stories. We found agglomerative document clustering highly effective (82% in the F_1 measure) for retrospective event detection, and single-pass clustering with time windowing a better choice for on-line alerting of novel events. We also observed robust learning behavior for k-nearest neighbor (kNN) classification and a decision-tree approach in event tracking, under the difficult condition when the number of positive training examples is extremely small.

1 Introduction

The rapidly-growing amount of electronically available information threatens to overwhelm human attention, raising new challenges for information retrieval (IR) technology. Traditional query-driven retrieval is useful for content-focused queries, but deficient for generic queries such as "What happened?" or "What's new?". Consider, for example, a person who just returns from an extended vacation and needs to find out quickly what happened in the world during her absence. Reading the entire news collection is a daunting task, while generating specific queries without any knowledge of recent events is rather unrealistic. Another example would be a foreign-policy specialist who wants to study the Asian economic crisis including precursor and consequent events in the surrounding time period. A keyword-based search on the query "Asian economy crisis" would most likely miss many relevant stories reporting the stock market crashes in Indonesia or Korea, or the banking-sector insolvency problem in Japan, or the stories about Habibi's rise to power in Indonesia. In other words, query-based retrieval is useful when one knows more precisely the nature of the events or facts one is seeking, and less useful when one wants specific information but can only formulate a larger category-query sharing few if any terms with the potentially most useful texts. In short, retrieval based on immediate-content-focused queries is often insufficient for obtaining a variety of relevant stories and tracking the gradual evolution of events through time.

In the examples above, it would be equally difficult for the user to formulate "the right query" or "the right level of abstraction", or to check through all the stories potentially relevant. It would be desirable for an intelligent system to detect automatically significant events from large volumes of news stories, present the main content of events to the user in a summarized form with multiple levels of abstraction, alert the onset of novel events as they happen, and track events of interest based on user-given sample stories. This is

the goal of a new line of research, namely, Topic Detection and Tracking (TDT) [1]. The task is defined over chronologically-ordered news stories from multiple channels of TV/radio broadcasts or newswire sources. The input data can be in the form of original or transcribed text, or the output of automated speech recognition which is typically with about 25% to 50% word-recognition error. The TDT problem consists of three major sub-problems: segmenting speech-recognized TV/radio broadcasts into news stories, detecting events from unsegmented or segmented news streams, and tracking stories for particular events based on user-identified sample stories. *Topic* in this context is used to mean dynamically changing events, which is different from the traditional sense of topic in the literature, as discussed further in Section 2.

In this paper we report our work on event detection and event tracking on manually segmented documents only; the CMU work at segmentation has been reported in separate papers[4]. The detection work was partially reported in a previous paper[33]. Directly related to our work is the on-going research in the other TDT-member groups, among which UMass and Dragon have published their approaches[1, 2]. UMass adapted their benchmark IR systems (InQuery and InRoute) to the TDT problems, using a combination of statistical phrase finding, part-of-speech tagging, TF-IDF term weighting and a Rocchio classification method[2]. Dragon applied speech recognition techniques, including unigram (and later bigram) language modeling for event representation and a k-means clustering method for document classification. Indirectly related work includes document clustering methods applied to retrieval and corpus navigation problems[25, 26, 12, 29, 24, 8, 34], and supervised learning algorithms applied to text categorization[13, 7, 31, 27]. Those results provide a rich background to our research, but do not directly address the problems of event detection and event tracking in temporal text and audio streams.

2 Event Analysis

Before exploring the solution space, let us observe the properties of events in news stories, which may shed light on what makes event detection and tracking a new challenge to traditional information retrieval and machine learning technology.

The TDT1 corpus, developed by the researchers in the TDT Pilot Research Project, is the first benchmark evaluation corpus for TDT research[2]. Table 1 shows the 25 events manually identified in this corpus. TDT1 consists of 15,863 chronologically-ordered news stories spanning the period from July 1, 1994 to June 30, 1995. Roughly half of these stories are randomly sampled Reuters articles, and the other half are CNN broadcasts which were manually transcribed by the Journal Graphics Institute (JGI). The event identification process consists of randomly sampling from the corpus, defining the events discussed in the sampled stories, and making exhaustive relevance judgements for each of those events. Each story was assigned a label of YES, NO or BRIEF with respect to each of the 25 events. YES indicates that the article focuses on a particular event, while BRIEF indicates that the article mentions the event in passing, but does not discusses it as a major focus. Note that this process resulted in only a subset of the existing events in the corpus, and that the random sampling made larger events (those reported more often) to be more likely to be included, compared to smaller events.

There is a difference between an *event*, as specified in the TDT problems, and a *topic* in the conventional sense. An event identifies *something (non-trivial) happening in a certain place at a certain time*. For example, *USAir-427 crash* is an event but not a topic, and "airplane accidents" is a topic but not an event. One could think events as instances of topics, associated with certain actions. In event detection and tracking, these distinctions must be drawn automatically by the system. Another interesting characteristic of news-story

[1]The Topic Detection and Tracking research was initiated and supported by the U.S. Government since 1996. Three research groups participated the TDT Pilot Research Project (1996-1997), including Carnegie Mellon University (CMU), the University of Massachusetts at Amherst (UMass) and Dragon Systems Corporation (Dragon). A much larger number of research groups participates in the current TDT Project, Phase-2 (TDT2, 1998-1999), including CMU, UMass, Dragon, UPenn, IBM, BBN, SRI, etc.

[2]The TDT1 corpus is made available recently via the Linguistic Data Consortium (LDC). Larger and richer corpora (TDT2 and TDT3) have been and continue to be developed by the LDC – see www.ldc.upenn.edu/TDT. In this paper, we report our experiments on the TDT1 corpus only.

events is that they are often associated with news bursts. Figures 1 and 2 illustrate the temporal histograms of a few events, where the X-axis of each graph is time (numbered from day 1 to 365), and the Y-axis is the story count per day. Figure 2 shows the histograms of all the 25 events in TDT1.

Several patterns emerged from our observations of temporal event distributions:

- News stories discussing the same event tend to be temporally proximate, suggesting the use of a combined measure of lexical similarity and temporal proximity as a criterion for document clustering.
- A time gap between bursts of topically similar stories is often an indication of different events (e.g., different earthquakes, airplane accidents, political crises, etc.). This suggests that monitoring cluster evolution over time is necessary and that using a time window to restrict the temporal extent of an event would be beneficial.
- A significant vocabulary shift and rapid changes in term frequency distribution are typical of stories reporting a new event, indicating the importance of dynamically updating the corpus vocabulary and statistical term weights. A timely recognition of new patterns, including previously unseen proper names and proximity phrases, in the streams of stories is potentially useful for detection of the onset of a new event.
- Events are typically reported in a relatively brief time window (e.g. 1-4 weeks) and contain fewer reports than broader topics. Hence we need learning methods that require few positive training examples to achieve satisfactory tracking performance, and that can exploit the temporal decay inherent in reporting of events.

3 Event Detection Methods

Event detection is a unsupervised learning task, sub-divided into two forms. One is *retrospective detection*, which entails the discovery of previously unidentified events in an chronologically-ordered accumulation of documents (stories). The other is *on-line detection*, where the goal is to identify the onset of new events from live news feeds in real-time. Both forms of detection intentionally lack advance knowledge of novel events, but may have access to unlabelled historical news stories for use as contrast sets.

Given that each event is usually discussed by multiple news stories, document clustering appears to be a natural approach to event discovery. We implemented two clustering methods: a divide-and-conquer version of a *Group-Average Clustering* (GAC) algorithm [29], and a single-pass incremental clustering algorithm (INCR). GAC is for agglomerative clustering, resulting in hierarchically organized document clusters. It is designed for batch processing, and has been used for retrospective detection. INCR is designed for sequential processing, and has been used for both retrospective detection and on-line detection. INCR results in a non-hierarchical partition of the input collection.

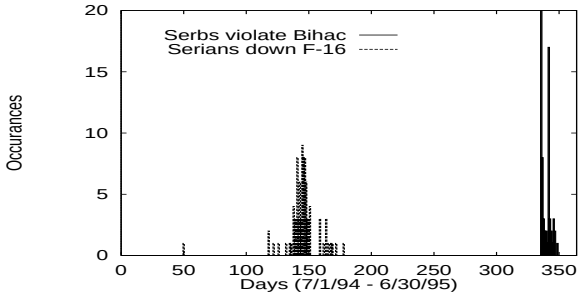


Figure 1: Histogram of *Serbian*-related events

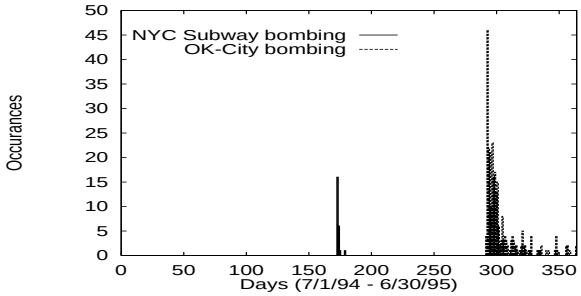


Figure 2: Histogram of *bombing*-related events

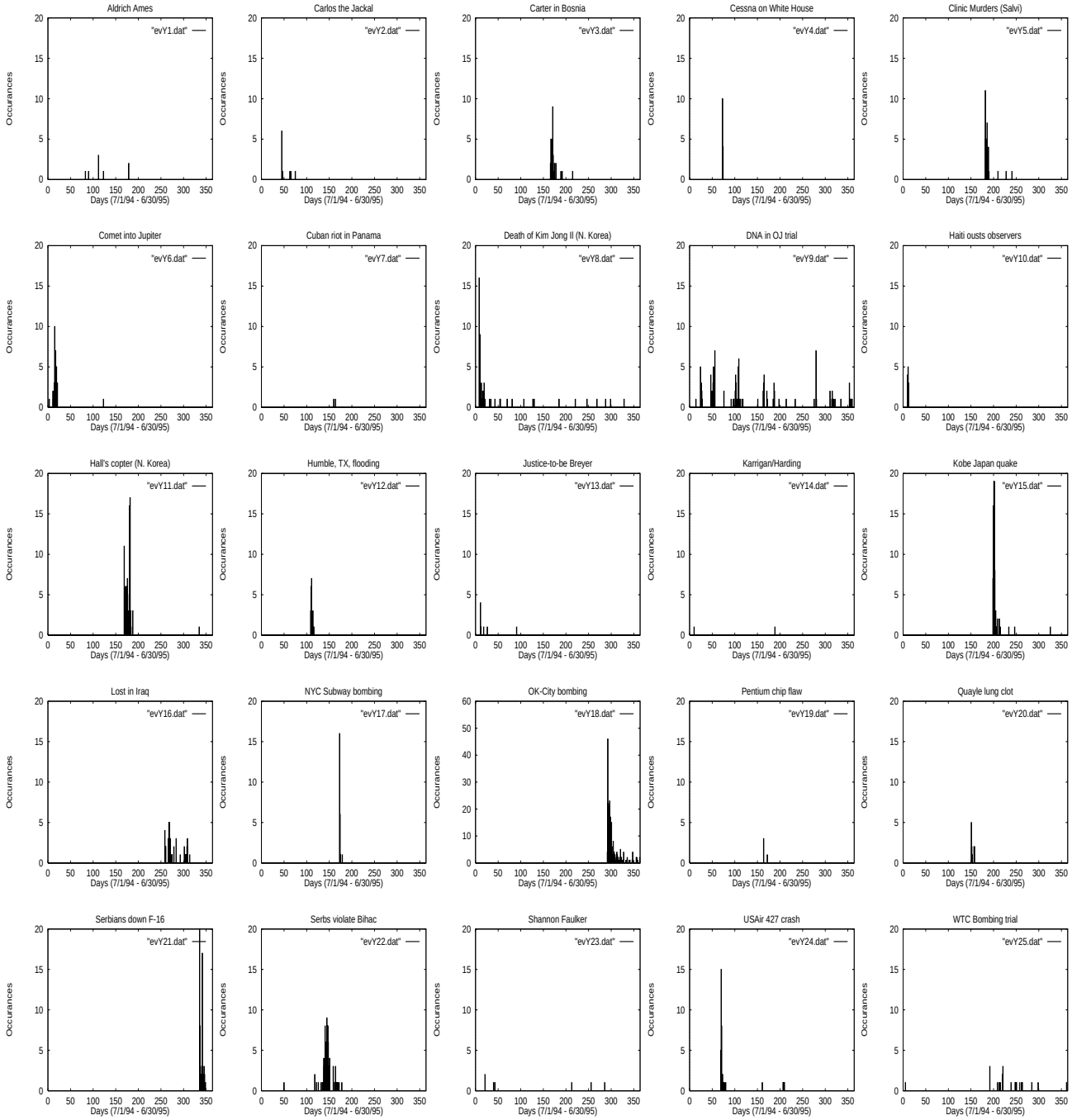


Figure 3: Histograms of the 25 TDT events

EventID	Count	StartTime	Name
1	8	94-02-22	Aldrich Ames
2	10	94-08-15	Carlos the Jackal (his capture)
3	34	94-12-25	Carter in Bosnia
4	14	94-09-12	Cessna on White House
5	41	94-12-30	Clinic Murders (Salvi)
6	45	94-07-16	Comet into Jupiter
7	2	94-12-08	Cuban riot in Panama
8	58	94-07-08	Death of Kim Jong Il (N. Korea)
9	114	94-07-??	DNA in OJ trial
10	12	94-07-11	Haiti ousts observers
11	97	94-12-17	Hall's copter (N. Korea)
12	22	94-10-19	Humble, TX, flooding
13	8	94-07-12	Justice to be Breyer
14	2	94-01-06	Karrigan/Harding
15	84	95-01-17	Kobe Japan quake
16	44	95-03-13	Lost in Iraq
17	24	94-12-21	NYC Subway bombing
18	273	95-04-19	OK City bombing
19	4	94-11-22	Pentium chip flaw
20	12	94-11-29	Quayle lung clot
21	65	95-06-02	Serbians down F-16
22	91	94-11-11	Serbs violate Bihac
23	7	94-07-22	Shannon Faulkner
24	39	94-09-08	USAir-427 crash
25	22	95-01-09	WTC Bombing trial

Table 1: Events in the TDT1 Corpus

3.1 Cluster Representation

Our clustering algorithms are rooted in the conventional vector space model[23] and traditional clustering techniques in information retrieval[8, 25, 29]. Each document is represented using a vector of weighted terms which can be either words or phrases. For term weighting we use a standard version ("ltc") of the TF-IDF scheme [3] :

$$w(t,d) = \frac{(1+\log_2 tf(t,d)) \times \log_2(N/n_t)}{\|\vec{d}\|}$$

where $w(t,d)$ is the weight of term t in document d;
$tf(t,d)$ is the within-document term frequency (TF);
$\log_2(N/n_t)$ is the Inverted Document Frequency (IDF);
N is the size of the training corpus used to compute IDF;
$n(t)$ is the number of training documents where term t occurs;
$\|\vec{d}\| = \sqrt{\sum_t w(t,d)^2}$ is the 2-norm of vector $\vec{d}$.

[3] TF-IDF based term weighting has been intensively studied in the IR literature. The implementation of the standard versions (more than a dozen) are provided in the SMART benchmark retrieval system (developed at Cornell). We tested a few common options and found the *ltc* option yielded the best detection results in our limited experiments. This does not mean that *ltc* is the best possible term weighting scheme for document clustering or for event detection. Finding the best term weighting scheme is an open research question at the current stage.

As for cluster representation, a *prototype* vector (also called the *centroid* of the cluster) is obtained by summing the vectors of the member documents and selecting the k most significant terms per prototype. Each document is treated as an initial cluster with a single member. To measure the distance between two clusters, we use the standard cosine similarity, i.e., the cosine value of the two prototype vectors.

A modification we made to standard TF-IDF term weighting is to use **adaptive IDF** instead of static IDF. Since new stories arrive continuously, how should we deal with the new vocabulary from incoming documents and update the corpus-level IDF statistics (which impact term weighting and vector normalization)? Related work shows that incremental IDF can be effectively used for document retrieval after a sufficient number of "past" documents have been processed[5]. For on-line event detection, we used a retrospective corpus (e.g., a six-month collection of CNN news stories prior to the TDT1 corpus) to compute the initial IDF values, and then incrementally update them with each incoming document. The incremental version of the IDF is defined to be:

$$IDF(t,p) = \log_2(N(p)/n(t,p))$$

where p is the current time point, $N(p)$ is the number of accumulated documents up to the current point (including the retrospective corpus if used), and $n(t,p)$ is the document frequency of term t at time p. The incremental IDF is used in our on-line event detection and tracking systems. For retrospective detection, on the other hand, the static IDF trained on the entire TDT corpus is used.

3.2 Group-average-based hierarchical clustering

GAC maximizes the average similarity between document pairs in the resulting clusters by merging clusters in a greedy, bottom-up fashion[25, 29]. Straightforward GAC algorithms typically have a complexity in time and space quadratic to the number of input documents, which is less economical or tractable for large applications than simpler methods, such as *single-link* clustering or single-pass k-mean clustering. This problem was addressed by Cutting et al. [8] using a divide-and-conquer strategy (referred to as "fractionation") which achieves a compromise between cluster quality and computational efficiency. It grows clusters iteratively; in each iteration, the current pool of clusters is divided into evenly-sized *buckets*, and GAC is applied to each bucket locally, merging smaller clusters into larger ones. We implemented this fractionation algorithm with GAC locally applied, and refer this approach as "GAC" in this paper if not otherwise specified. This algorithm has a time complexity of $O(mn)$ where n is the number of documents in the input corpus, m is the bucket size, and $m \leq n$.

The bucketing strategy is particularly well-suited for the event detection problem; we found that bucketing stories based on the order that they are reported produces gains not only in computational efficiency, but also improves the cluster quality and indeed the detection effectiveness. In other words, this strategy gives a higher priority to grouping temporally-proximate stories rather then temporally disparate ones.

The input to our algorithm is a collection of documents sorted in chronological order, and a set of user-specified parameters (see the Steps below). The output is a forest of binary trees of clusters. The algorithm consists of the following steps:

1. Treat each document in the input collection as a singleton cluster, and set the initial partition to be the full set of the singleton clusters.
2. Divide the current partition into non-overlapping and consecutive buckets of size m (a user-specified parameter).
3. Apply GAC to each bucket, which repeatedly combines the two closest lower-level clusters into a higher level cluster, until the number of clusters in the bucket is reduced by a factor of ρ (a user-specified parameter), or if all the similarity scores between two clusters are below a pre-selected clustering threshold (another user-specified parameter).
4. Remove the bucket boundaries (assemble all the GAC clusters) while preserving the time order of the clusters. Use the resulting cluster series as the updated partition of the corpus.

5. Repeat steps 2-4, until the size of the partition is no larger than m, or stops being reduced due to the minimum similarity constraint.

6. Periodically (once per k iterations in Step 5) re-cluster the stories within each of the top-level clusters, by flattening the component clusters and re-growing clusters internally from the leaf nodes.

The temporal-bucketing and re-clustering are our modification to Cutting's algorithm. Re-clustering is useful when events straddle the initial temporal-bucket boundaries, or when the bucketing caused non-desirable grouping of stories about different events. Re-clustering reduces the systematic bias of the initial bucketing, at a cost of increased computation time.

3.3 Single-pass clustering

The INCR algorithm is straightforward. It sequentially processes the input documents, one at a time, and grows clusters incrementally. A new document is absorbed by the most similar cluster in the past if the similarity between the document and the cluster is above a pre-selected **clustering threshold** (t_c); otherwise, the document is treated as the seed of a new cluster. By adjusting the threshold, one can obtain clusters at different levels of granularity. Suitable choice of clustering threshold, therefore, is important for the effectiveness of retrospective event detection where the granularity levels of document clusters should match the concepts of events.

For the application of INCR to on-line event detection, we introduced an additional threshold, namely, the **novelty threshold** (t_n). If the maximal similarity score between the current document and any cluster in the past are below the novelty threshold, then this document is labelled as "NEW", meaning that it is the first story of a new event; otherwise a flag of "OLD" is issued. By tuning the novelty threshold, one can adjust the sensitivity to novelty in on-line detection.

Both the clustering threshold and the novelty threshold are user-specified parameters to INCR. The choice for one threshold is independent of the choice for the other. Using both thresholds permits better empirical optimization for different tasks. For instance, we found that setting $t_c = t_n$ is appropriate for retrospective clustering (i.e., t_n is not needed), but for on-line detection choosing $t_c = \infty$ is better (i.e., not growing any clusters).

Another functionality we added to INCR is the **time penalty** in documents clustering. The simplest way is to use a uniformly weighted time window. Given the current document (x) in the input stream to INCR, we impose a **time window** of m documents prior to x, and define the modified similarity between x and any cluster (c) in the past to be:

$$sim(\vec{x}, \vec{c})' = \begin{cases} sim(\vec{x}, \vec{c}) & \text{if } c \text{ has any member-document in the time window;} \\ 0 & \text{otherwise.} \end{cases}$$

Alternatively, we can introduce a linear decaying-weight function in the above formula:

$$sim(\vec{x}, \vec{c})' = \begin{cases} (1 - \frac{i}{m}) \times sim(\vec{x}, \vec{c}) & \text{if } c \text{ has any member-document in the time window;} \\ 0 & \text{otherwise} \end{cases}$$

where i is the number of documents between document x and the most recent member-document in cluster c. The decaying-weight function makes a smoother use of the temporal proximity, compared to using a uniformly weighted window[4]. These windowing strategies yielded measurable and consistent improvements in our event detection experiments, enhancing precision with only a small sacrifice in recall, compared to not using time penalty.

[4]For simplicity we only define a linear function for the decay weighting; however, it is easy to generalize this definition to a more elaborate form if necessary, such as the interpolated decay profile extracted from an earlier development or training corpus.

In addition to the binary (New or Old) prediction, a score is also computed for each incoming document, indicating the how *new* this document is as measured by the system. This score is defined to be:

$$score(x) = 1 - \arg\max_{c}\{sim(x, c)'\}$$

where x is the currently new document and c is any cluster in the past. The scores of documents are used for evaluating the potential trade-off between different types of errors. That is, by adjusting the threshold on these scores for binary decisions, one can obtain the trade-off curve between recall and precision, or between miss and false alarm (refer to Section 4.3 for more details).

4 Event Detection Evaluation

4.1 Examples of Resulting Clusters

Table 2: Clusters generated by GAC on TDT1 Corpus, Jan-Feb 1995 Subset

Documents included	Top-ranking Words (stemmed)
330	republ clinton congress hous amend
217	simpson o prosecut trial jury
98	israel palestin gaza peac arafat
97	japan kobe earthquak quak toky
93	russian chech chechny grozn yeltsin
56	somal u mogadishu iraq marin
55	flood rain californ malibu rive
48	serb bosnian bosnia croat u
35	game leagu play basebal season
33	crash airlin flight airport passeng
28	clinic sav abort massachuset norfolk
27	shuttl spac astronaut mir discov
26	patient drug virus holtz infect
24	chin beij deng trad copyright
...	

Table 2 shows a corpus summary obtained by applying our hierarchical clustering algorithm (GAC) to a few thousand news stories, i.e., CNN news and Reuters articles from January to February in 1995, and by presenting a few top-ranking terms for each cluster. As the table shows, domestic politics reigned supreme as usual, the OJ trial received media attention in early 1995, etc. However, the table also reveals that disasters struck Kobe Japan and Malibu California, and unrest in Chechnya has flared up again, events which were not present the months before. The key terms provide content information, and the story counts imply significance, as measured by media attention. New multi-document summarization methods [6, 14] applied to the clusters provide additional information as to the nature of the events. And, if further detail is desired, the clusters, sub-clusters, and individual documents can be examined via query-driven retrieval. The utility of summarization and cluster-based browsing tools is evident from our prototypes, even though some clusters may be imperfect and the current user interface is rudimentary.

Figures 3-6 show the temporal distributions of two events, The upper half of each graph is the histogram of human-labeled documents for an event; the lower half is the histogram of the system-generated cluster for the same event. The absolute value on the Y-axis is the story count for the event or cluster in a particular day. If an event and a cluster are a perfect match, then their histograms will completely mirror each other.

As the figures show, GAC and INCR have complementary strengths and weaknesses. GAC shows almost symmetric graphs for most events, except those with significant temporal extent, and therefore it is particularly suitable for recognition of news bursts. INCR, on the other hand, has less symmetric

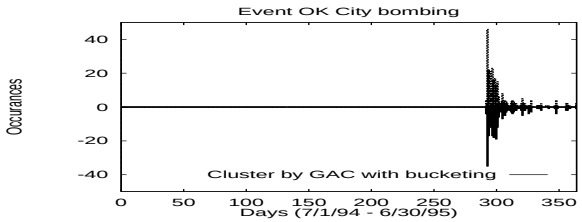


Figure 3: Event *OK City bombing* vs GAC-cluster

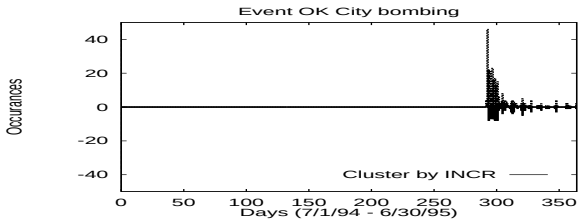


Figure 4: Event *OK City bombing* vs INCR-cluster

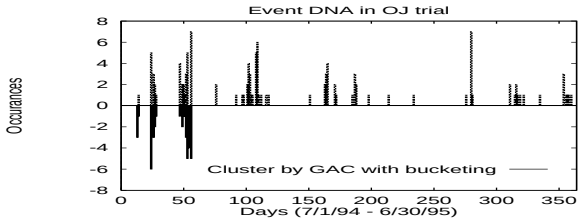


Figure 5: Event *DNA in OJ trial* vs GAC-cluster

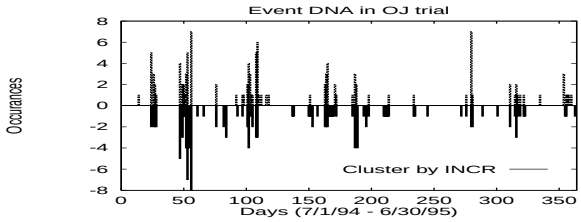


Figure 6: Event *DNA in OJ trial* vs INCR-cluster

performance compared to GAC, but is better at tracking long-lasting events (e.g. *DNA in O.J. trail* and *OK City bombing*). The observed behavior may come partly from the different biases in these algorithms and partly from the parameter settings in the particular experiments.

4.2 Retrospective Detection Results

The TDT1 evaluation in 1998 is the first controlled study[1] where comparative results are available. Therefore we use that evaluation as a reference for the results presented here. The entire TDT1 corpus was used as the test set for evaluating detection systems, although it would be preferable if an additional cross-validation corpus were available for setting global system parameters. Nevertheless, since detection is an un-supervised classification task which does not involve labeled training data, there was no contamination of the test data in that sense, except possibly with respect to setting a handful of system parameters. Subsequent work in progress on independently developed TDT2 corpora indicate that the parameter values can be effectively chosen using a retrospective corpus and cross validation. For example, we found that the clustering threshold optimal on the TDT1 corpus for on-line event detection is nearly optimal on the TDT2 corpus.

Each detection system was run on the entire corpus of TDT1, resulting in system-generated clusters which are either a partition of the corpus (i.e., no overlapping stories between clusters), or a forest of hierarchies (overlapping stories between clusters are allowed). Recall that there are 25 human-labeled events (consisting of 1131 stories; about 7% of the total stories). Each system was evaluated using the 25 clusters that best matched the 25 manually-labeled events. The goodness of matching between each cluster/labeled-event pair was evaluated using a contingency table, as shown in Table 3[5].

Performance measures are defined using the contingency table:

[5]Note that there are more events in the TDT1 corpus beyond the 25 labelled ones; also, there are more than 25 clusters generated by each detection system. However, the match between clusters and events beyond the 25 cluster-event pairs was not measured in the evaluation. I.e. the system detected many potential events, but it was only evaluated on a subset of the system-generated clusters which best matched the manually-labelled events. The fact that there are many clusters corresponding to other potential events suggests a natural usage of a detection system, i.e., as a browsing support to the user for the navigation through the event space. Although we expect hierarchical clustering to be a suitable choice for navigation support, how to evaluate the practical impact of various kinds of navigation support requires future research.

Table 3: Per-event contingency table

	in event	not in event
in cluster	a	b
not in cluster	c	d

Table 4: Retrospective detection results

	Partition required			Cluster overlap allowed	
	CMU (INCR)	UMass (no-dupl)	Dragon (multi-pass)	CMU (GAC)	UMass (dupl)
micro-avg Recall (%)	62	34	61	75	73
micro-avg Precision (%)	82	53	69	90	78
micro-avg Miss (%)	38	66	39	25	27
micro-avg False Alarm (%)	.04	.09	.08	.02	.06
micro-avg F_1	.71	.42	.65	.82	.75
macro-avg F_1	.79	.60	.75	.84	.81

- Miss $m = c/(a+c)$ if $a + c > 0$, otherwise undefined;
- False Alarm $f = b/(b+d)$ if $b + d > 0$, otherwise undefined;
- Recall $r = a/(a+c)$ if $a + c > 0$, otherwise undefined;
- Precision $p = a/(a+b)$ if $a + b > 0$, otherwise undefined;
- $F_1 = 2rp/(r+p) = 2a/(2a+b+c)$ if $(a+b+c) > 0$, otherwise undefined.

where F_1, originially defined by van Rijsbergen[25], is the harmonic mean of recall and precision.

In order to measure global performance, two averaging methods are used. The *micro-average* is obtained by merging the contingency tables of the 25 events (by summing the corresponding cells), and then using the merged table to produce global performance measures. The *macro-average* is obtained by producing per-event performance measures first, and then averaging the corresponding measures. The former introduces a scoring bias towards frequently-reported events, and the latter towards less-reported ones, hence both measures are given to minimize the effect of hidden bias.

Table 4 summarizes the retrospective-detection results. Tables 5 and 6 provide the parameter settings in the two CMU detection systems: *GAC* and *INCR*. For comparison, we indlude the results for approaches developed at theUniversity of Massachusetts (temporal-TF based event detection and agglomerative clustering) and by Dragon systems (multipass k-means clustering)[1]. Algorithms that permit cluster hierarchies (GAC) or potentially-overlapping clusters (dupl) performed better than non-hierarchical algorithms which adhere to the strict partition requirement.

In the results of the partition-producing algorithms, we were surprised that the simplest approach – the single-pass clustering by INCR – worked as well as the multi-pass k-means clustering method by Dragon. This may be partly because of the temporal proximity of events which simplifies the clustering problem. We found time windowing highly effective for the performance of INCR; with other parameters fixed, using a window of 2000 documents (covering about 1.5 months of time) improved the performance score from 0.64 to 0.70 in the F_1 measure, versus no time window.

The better results obtained by hierarchical clustering (CMU's GAC) or overlapping clustering (UMass's *dupl*) is less surprising. We believe the main reason for the better results of GAC is the multi-leveled clusters which enable the detection of events at any degree of granularity. This representational power of GAC comes

Table 5: Parameters used in retrospective GAC

bucket size	= 400
clustering threshold	= .2
terms per vector	= 100
term weighting	= ltc
reduction factor ρ	= 0.5
# of iterations between re-clustering	= 5

Table 6: Parameters used in retrospective INCR

window size	= 2000
clustering threshold	= .23
terms per doc vector	= 125
term weighting	= ltc

with the cost of producing a larger number of clusters (about 12,000 in this particular run), than the number of clusters (5,907) in the partition by INCR. The increase in the number of clusters may not add a significant burden to the end user in scatter-gather navigation or query-driven retrieval[8, 15], where only a small subset of the clusters would actually be visited by the user via selected paths on the hierarchy.

4.3 On-line Detection Results

Online detection performance is evaluated using a contingency table as shown in Table 7. Since there are only 25 events defined, and each event has only one first story, the total number of true *New* stories is 25 for the entire corpus. This is too small a number for a statistically reliable estimation of performance. To improve the reliability, an 11-pass evaluation was conducted. The first pass used the entire corpus; the second pass used the modified corpus after removing ("skipping") the first story of each event; the third pass used the modified corpus after removing the first two stories of each event, and so on. The eleven passes are labeled as $N_{skip} = 0, 1, \ldots, 10$. A contingency table was computed for each value of N_{skip}; a global contingency table then was obtained by summing the corresponding cells in the per-N_{skip} contingency tables. Performance scores were derived from those contingency tables in the same way as described for retrospective detection evaluation.

Table 7: On-line detection contingency table

	New is true	Old is true
Predicted New	a	b
Predicted Old	c	d

Table 8 summarizes the official TDT1 results, including those by UMass and Dragon for on-line detection. Both CMU and UMass conducted multiple runs with different parameter settings in the TDT workshop[1]; here we present the best result for each site with respect to the F_1 measure. A possible reason for the significantly higher F_1 scores by UMass and CMU over those by Dragon is that both UMass and CMU chose to use individual documents instead of clusters to represent the past in on-line detection; Dragon, on the other hand, grew clusters instead[6]. Keeping individual documents without clustering them makes the *novelty test* more difficult for the current story to pass, because this story must be sufficiently different from *all* of the past stories, a stronger condition compared to being different from an *average* of past stories.

In addition to the scores presented in Table 8, a Detection-Error Tradeoff (DET) curve[19] is also used to evaluate each on-line detection system in the TDT evaluation. A DET curve is defined to be the sequence of interpolated values in the false-alarm/miss space, and is obtained by retrospectively thresholding on the system-generated scores for individual documents. At a particular threshold, any document with a score above the threshold is labelled as *New*, and *Old* otherwise. This process results in a set of performance

[6] UMass used a single-pass algorithm to compare a new document with all the past documents, which is similar to our INCR; Dragon used a single-pass version of their k-means clustering method for on-line detection. Multi-pass clustering cannot be used for on-line detection because, by the task definition, future knowledge is not available at the decision making point.

Table 8: On-line event detection results

	CMU	UMass	Dragon
micro-avg Recall (%)	50	49	42
micro-avg Precision (%)	37	45	21
micro-avg Miss (%)	50	51	58
micro-avg False Alarm (%)	1.89	1.31	3.47
micro-avg F_1	.42	.47	.28

Table 9: Parameters used in on-line INCR

window size	= 2500 linear decay
clustering threshold	= ∞
novelty threshold	= .16
terms per vector	= no limit
term weighting	= ltc
IDF	= adaptive*

* initially trained on a retrospective corpus

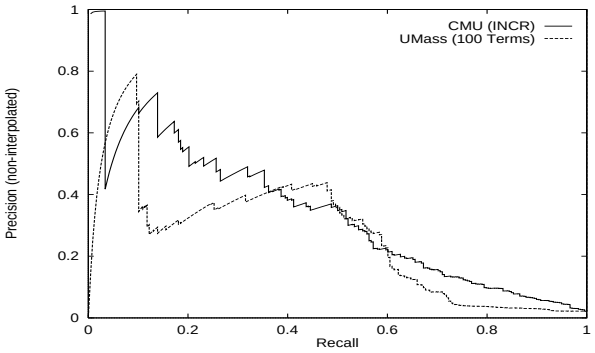


Figure 7: On-line detection Recall-Precision curves

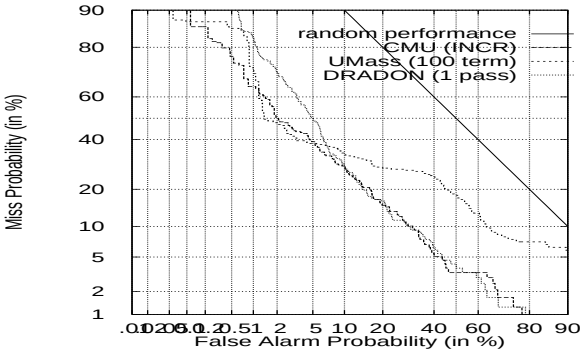


Figure 8: On-line detection DET curves

scores such as those in Table 8 for each threshold value. By changing the threshold value and interpolating the corresponding miss and false-alarm values, one can observe the trade-off between them (Figure 8). Each DET curve axis (miss and false-alarm) is scaled to a Gaussian (thereby compressing the midrange and expanding the extremes) so that a "random"-decision plot is a straight line passing through the 50%-50% error point. As an alternative to DET curves, one can also plot the corresponding non-interpolated recall and precision (Figure 7). In these curves, the CMU results show better performance at the high precision area. As is especially evident in Figure 7, the CMU, UMass and Dragon approaches exhibit very different behaviors, inviting further detailed investigation. Note the recall-precision curves are, in IR terminology[10], *non-interpolated*, and exhibit the typical non-monotonic behavior as analyzed in [23].

5 Event Tracking Methods

Event tracking is a supervised learning task, aiming to automatically assign event labels to news stories at the time they arrive based on a small number of previously identified past stories that define the event. Adaptive learning is needed due to the dynamic nature of events, i.e., they start at certain time points and trail off after a while. Making fine distinctions between topically related events is another task-specific requirement; for example, NYC Subway bombing and OK City bombing should be identified as different events. Moreover, quick learning is highly desirable, which means that the classifier needs only a small number of positive training examples per event in order to achieve satisfactory tracking performance. We found that two well-known learning methods, i.e., *k-Nearest Neighbor* (kNN) classification and a *Decision-Tree Induction* (d-tree) well suited to this task, after some extensions to the standard algorithms.

The kNN algorithm uses the same document representation used for event detection, i.e., a document is represented as a bag of terms with statistical weights. The decision tree algorithm, on the other hand, uses binary term weighting only (1 for terms present and 0 for terms absent).

5.1 K-Nearest Neighbor Classification (kNN)

kNN is an instance-based classification method well-known in in pattern recognition[9] and machine learning[21], and has been applied to text categorization (TC) since the early stages of the research[20, 30, 16]. It has been used as a baseline in recent TC comparative research on the benchmark Reuters corpus of newswire stories, where the top-performing methods include kNN and the Linear Least Squares Fit mapping by Yang et al.[31], Generalized Instance Sets by W. Lam et al.[18], decision trees with boosting by Weiss et al.[27], Support Vector Machines by Joachims[17, 11], and neural networks by Wiener et al. [28] Other methods that performed less well in TC include Naive Bayes classifiers, decision trees without boosting, and rule induction algorithms[31, 32]. We chose kNN for event tracking because, in addition to its generally good performance, it makes the fewest assumptions about terms, stories and optimal decision surfaces for the tracking task.

A requirement in official TDT evaluations is that each event be tracked independently, without any knowledge about other events. That is, for each particular event, the training stories are labelled either YES, NO or Brief; no event labels are given for the negative stories. According to this constraint, we adapted our conventional M-ary classification kNN (developed for text categorization in general)[30, 31] to the binary classification problem of event tracking. We trained a specific kNN classifier for each event. The system converts an input story into a vector as it arrives and compares it to the training stories (see Section 6 for the training-set construction), and select the k nearest neighbors based on the cosine similarity between the input story and the training stories. The confidence score for a YES prediction on the input story is computed by summing the similarity scores for the positive and negative stories respectively in the k-neighborhood, and taking the difference between the two sums:

$$s1(YES|\vec{x}) = \sum_{\vec{d} \in P(x,k)} \cos(\vec{d}, \vec{x}) - \sum_{\vec{d} \in N(x,k)} \cos(\vec{d}, \vec{x})$$

where $\vec{x}$ is the input story;
$P(x,k)$ is the set of positive training stories in the k-neighborhood;
$N(x,k)$ is the negative training stories in the k-neighborhood.

Binary decisions are obtained by thresholding locally on the confidence scores generated by each event-specific classifier. Our experiments showed good results (Section 6) for kNN in making binary decisions when threshold at the zero value of the confidence score. However, when moving the threshold beyond that point, we foundthat it resulted in a somewhat unsatisfactory DET curve. More specifically, it has difficulty gaining a high recall without sacrificing precision significantly. The reason, we believe, is that the positive examples are extremely sparse (for most events) in the training set, and therefore often "blocked away" by densely populated negative examples. One solution to this problem is to discount the influence of negative examples by sampling a small portion in the k-neighborhood, and ignoring the remaining one. This idea leads to a modified version of kNN; for distinction, we refer to the original version as kNN-a, and the modified version as kNN-b. In the modified version, we take $k1$ ($\leq k$) nearest positive examples ($P(x,k1)$) and $k2$ ($\leq k$) nearest negative examples ($N(x,k2)$) from the k-neighborhood, and average the similarity scores of the two subsets respectively. The confidence score for a YES prediction on the input story is defined to be:

$$s2(YES|\vec{x}) = \frac{1}{k1} \sum_{\vec{d} \in P(x,k1)} \cos(\vec{d}, \vec{x}) - \frac{1}{k2} \sum_{\vec{d} \in N(x,k2)} \cos(\vec{d}, \vec{x})$$

By introducing the parameters $k1$ and $k2$ in addition to k, and by suitably choosing the parameter values, we can effectively adjust the DET behavior of our tracking system. In principle, these parameters can be empirically tuned based on the optimization of event tracking on a validation collection of stories. In reality, when only a very small number of positive examples are identified for an events, one would not want to use these positive examples for validation instead of training. The official event tracking evaluation restricted the number (N_t) of positive training examples per event to be 1, 2, 4, 8 and 16, respectively(Section 6).

Under such an condition, we used a *rule of thumb* for determine the values of our parameters:

$$\text{in kNN-a:} \quad k = \min\{N_t, 5\}$$

$$\text{in kNN-b:} \quad k1 = \min\{P(x, 100), N_t\}; \quad k2 = \min\{N(x, 100), 16\}$$

Another heuristic we used in event tracking is a *time window*. That is, any test story which is k-stories away from the last positive training example is assigned a NO decision. We empirically set the window size to 1800-2000 stories (about 1.5 months worth of data), based on the common sense that most events last no longer than one or two months, and on the observation that a 1.5-month window is close to optimal for the on-line detection task discussed earlier.

5.2 Decision Trees

Decision trees are classifiers built based on the principle of a sequential greedy algorithm which at each step strives to maximally reduce system entropy[21, 22]. Decision trees are constructed by selecting the feature with maximal information gain (IG) as the root node, and dividing the training data according to the values of this feature; then for each branch finding the feature which maximizes information gain over the training instances for that branch, and so on recursively. We chose d-trees as an alternative approach to kNN for TDT tracking because it represents a very different technology, and one with relatively reasonable performance in text categorization evaluations on the benchmark Reuters collection[3, 27]. One potential disadvantage of d-trees is that, unlike kNN, they cannot generate a continuously varying tradeoff between miss and false alarm, or recall and precision.

We developed our own d-tree method, rather than using C4.5[22] primarily because we wanted a version that is fast, scalable. and easily tunable for text categorization, though not necessarily optimized for other machine learning tasks, and without extra features such as C4.5's rules-from-trees option. Training 25 decision trees (one per event tracked) each with up to 15,000 stories requires less than two minutes in aggregate on a standard 300 MHz Sun Ultra II. We use the same information gain metric (minimizing total entropy) and greedy root-to-leaves d-tree construction as in C4.5. The primary tunable parameters are:

1. minimal number of training instances at leaf node
2. percentage of positive instances at leaf node
3. whether or not to use word roots/stems instead of surface forms
4. whether or not do distinguish between single and multiple occurrences of a word in a document
5. size of time window for training data (fixed or adaptive)
6. limit on positive or negative training examples
7. limit features to top N (e.g. top 1000) by global information gain
8. single feature nodes (as in C4.5) vs m-of-n feature decisions

These parameters are tuned by cross-validation. Some parameters, such as the time window, make a significant difference. Performance is optimized by using only the most recent 1.5 to 2 months training data. Other parameters such as stemming, or using only the top-N features make only a small difference in overall performance.

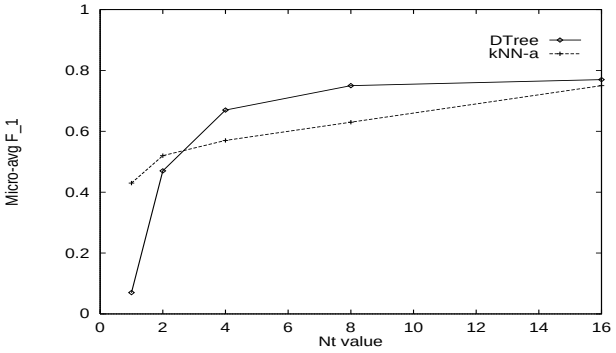


Figure 9: Micro-averaged learning curves of event tracking systems

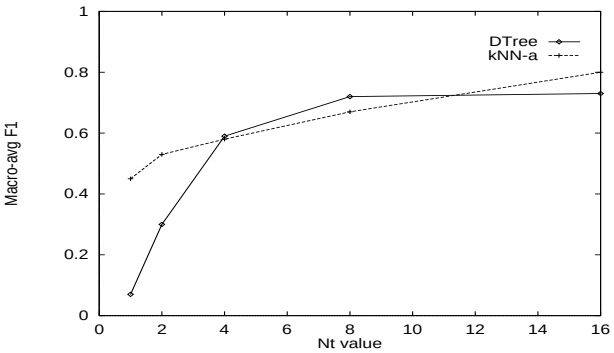


Figure 10: Macro-averaged learning curves of event tracking systems

6 Event Tracking Results

6.1 Tracking results on TDT1

The event tracking systems were evaluated using N_t positive training examples plus all available negative examples, where N_t takes on values 1, 2, 4, 8, and 16, respectively. For each event and a particular N_t value, the TDT1 corpus was split at the point right after the N_t-th positive example of that event; the stories before that split point were used for training, and the remaining stories were used for testing. Fifteen of the 25 events have more than 16 YES stories [7]; those were used for event tracking evaluation. Each system was tested on all the pairs (15×5) of training/test sets, resulting in 75 two-by-two contingency tables (for the cases of the predicted YES or NO versus the true YES or NO given an event). The micro-averaged and macro-averaged performance scores were computed from the 75 contingency tables. The results are shown in Table 10.

Table 10: Event tracking results: performance averaged over all events and all N_t = 1,2,4,8 and 16.

Classifier	kNN-a	d-tree	UMass (RF-10T)	Dragon
micro-avg Recall (%)	89	80	64	65
micro-avg Precision (%)	44	50	51	30
micro-avg Miss (%)	56	50	36	70
micro-avg False Alarm (%)	.04	.08	.39	.10
micro-avg F_1	.59	.61	.57	.41
macro-avg F_1	.62	.53	.63	.42

To observe the learning behavior of kNN-a and d-tree with respect to the number of positive training examples, we present in Figure 9 the interpolated curves of the micro-averaged $F1$ values. Figure 10 shows the corresponding curves for the macro-averaged performance. We see that both methods work reasonably well with the small N_t values. Interestingly, d-tree is not as good as kNN when $N_t = 1$ or 2; also, its curve asymptotes at $N_t = 8$. Our interpretation of this kind of behavior is that d-trees select only a few "good" features, but they (over)generalize quickly. This proves to be problematic when the input data is noisy as evidenced by the speech-recognition-result discussed below. On the other hand, kNN is based on the local training examples surrounding a test story, but uses all the terms in those stories as features.

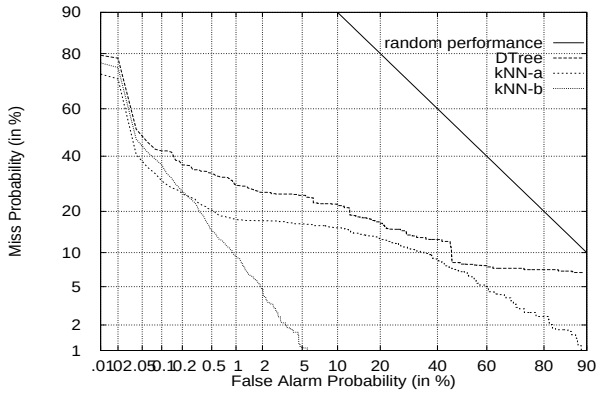


Figure 11: Detection Error Trade-off (DET) curves of d-trees and kNN

6.2 Trade-off improvement

To investigate the trade-off potential, we evaluated the false-alarm and miss rates of kNN-a and kNN-b when varying the decision thresholds on their confidence scores. Decision thresholding was applied locally within each event-specific kNN in the experiment with a fixed value of N_t. This resulted in 75 local DET curves per system for the 15 events and 5 N_t values in total. To observe the trade-off in average, we divided the range of false alarm into 5000 evenly-sized intervals, computed the average miss rate within each interval at the per-event and per-N_t basis, and then averaged these averages over events. Finally, we interpolated the resulting points in all the intervals. Figure 11 shows the average DET curves of d-trees, kNN-a and kNN-b, respectively, at the N_t value of 8. The comparison indicates that kNN-a has a better performance for high-precision (or low false-alarm) event tracking, while kNN-b would be a better choice for high-recall oriented applications. Note that this comparison is under the (crucial) condition of having very small N_t values, and is not necessarily generalizable to different conditions. In other words, our focus here is to evaluate our systems under a task-specific constraint, and we found that kNN-b effectively smoothes the detection error trade-off in event tracking, better than d-trees or kNN-a.

7 Summary

To summarize the main points in this paper, event detection and tracking represent a new family of tasks for information retrieval and machine learning. We studied a set of retrieval techniques and learning algorithms addressing the following challenges:

- analyzing the nature of events in news stories;
- identifying suitable learning algorithms for event detection and tracking;
- suggesting special-purpose changes to standard learning algorithms;
- evaluating the suggested techniques and comparing the results to those by other research groups using different techniques.

More specifically, our empirical evaluations suggest the following points:

- For retrospective detection, we have shown that conventional document representation and relatively simple clustering algorithms (GAC and INCR) can be highly effective, especially when they are adapted to make a combined use of context similarity and temporal proximity in document clustering.

[7] The stories judged as BRIEF were allowed to be used for training; however, they were excluded from testing.

- On-line novel-event detection is somewhat more difficult than retrospective detection. We observed better detection accuracy when using a non-clustering approach instead of clustering, although deeper understanding about this requires further investigation.
- For event tracking, both kNN and d-trees exhibit encouraging performance in *quick learning*, with their performance curve approaching a *plateau* after a very small number ($N_t = 4$ or 8) of positive training examples.
- For better detection error trade-off flexibility in event tracking, kNN-b shows a significant improvement by a suitable nearest-neighbor sampling and score normalization.

Important research questions for further investigation include:

- Are there better learning algorithms for the TDT problems?
- How can we model event evolution over time more accurately than via time windowing or simple linear decay?
- How can we combine document clustering and text summarization for user support in event detection and tracking?

Finally, it should be pointed out that the current TDT research and evaluations have not yet thoroughly address the questions about how to optimally use system-generated clusters, and how to best match these clusters to the "true events". In principle, finding a meaningful mapping without any information (e.g., positive and negative examples given an event) or knowledge about the target events (e.g., event descriptions) is an unrealistic task. Document clustering can be useful only if the user input (or interaction) is taken into account in the loop of the event identification. Further more, the system-generated clusters should be provided along with suggested browsing strategies to the users. Investigating potential strategies for traversing through cluster hierarchies or corpus partitions, and measuring the practical impact in terms of time saving and error reduction for the end users will be a crucial part of the future work in event detection and tracking.

8 Acknowledgments

The authors wish to thank Charles Wayne and George Doddington for their guidance in the TDT task definition and evaluation, and James Allan at UMass and Jon Yamron at Dragon for sharing ideas and results in the research.

The TDT research is sponsored in part by the Department of Defense. However, any opinions or conclusions in this paper are the authors' and do not necessarily reflect those of the sponsors.

References

[1] James Allan, Jaime Carbonell, George Doddington, Jonathan Yamron, and Yiming Yang. Topic detection and tracking pilot study: Final report. In *Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop*, 1998.

[2] James Allan, Ron Papka, and Victor Lavrenko. On-line new event detection and tracking. In *Proceedings of the 21th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98)*, pages 37–45, 1998.

[3] C. Apte, F. Damerau, and S. Weiss. Towards language independent automated learning of text categorization models. In *Proceedings of the 17th Annual ACM/SIGIR conference*, 1994.

[4] D. Beeferman, A. Berger, and J. Lafferty. Statistical models for text segmentation. In *Machine Learning*, volume 34, pages 1–34, 1999.

[5] Jamie Callan. Document filtering with inference networks. In *Proceedings of the 19th Annual ACM/SIGIR conference*, pages 262–269, 1996.

[6] J.G. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98)*, pages 335–336, 1998.

[7] R.H. Creecy, B.M. Masand, S.J. Smith, and D.L. Waltz. Trading mips and memory for knowledge engineering: classifying census returns on the connection machine. *Comm. ACM*, 35:48–63, 1992.

[8] 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)*, pages 318–329, 1992.

[9] Belur V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* McGraw-Hill Computer Science Series. IEEE Computer Society Press, Las Alamitos, California, 1991.

[10] Ed. DK Harman. *The Second Text REtrieval Conference (TREC-2).* US Government Printing Office, Washington, DC, 1994.

[11] S.T. Dumais. Using svms for text categorization. In *Support Vector Learning In: IEEE Intelligent Systems*, pages July–August, 13(4), 1998.

[12] T. Feder and D. Greene. Optimal algorithms for approximate clustering. In *Poceedings of the 20th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 434–444, 1988.

[13] N. Fuhr, S. Hartmanna, G. Lustig, M. Schwantner, and K. Tzeras. Air/x - a rule-based multistage indexing systems for large subject fields. In 606-623, editor, *Proceedings of RIAO'91*, 1991.

[14] Jade Golsdstein and Jaime Carbonell. The use of mmr and diversity-based reranking in document reranking and summarization. In *Proceedings of the 14th Twente Workshop on Language Technology in Multimedia Information Retrieval*, pages 152–166, Enschede, the Netherlands, December 1998.

[15] M. Hearst and J.O. Pedersen. Reexamining the cluster hypothesis: Scatter/gather on retrieval results. In *19th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'96)*, pages 76–84, 1996.

[16] Makato Iwayama and Takenobu Tokunaga. Cluster-based text categorization: a comparison of category search strategies. In *Proceedings of the 18th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'95)*, pages 273–281, 1995.

[17] Thorsten Joachims. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *European Conference on Machine Learning (ECML)*, 1998.

[18] W. Lam and C.Y. Ho. Using a generalized instance set for automatic text categorization. In *Proceedings of the 21th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98)*, pages 81–89, 1998.

[19] A. et al. Martin. The det curve in assessment of detection task performance. In *EuroSpeech 1997 Proceedings*, volume 4, 1997.

[20] B. Masand, G. Linoff, and D. Waltz. Classifying news stories using memory based reasoning. In *15th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'92)*, pages 59–64, 1992.

[21] Tom Mitchell. *Machine Learning.* McGraw Hill, 1996.

[22] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[23] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, Reading, Pennsylvania, 1989.

[24] R.H. Tohompson and B.W. Croft. Support for browsing in an intelligent text retrieval system. In *International Journal of Man-Machine Studies*, pages 30(6)639–668, 1989.

[25] C.J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 1979.

[26] Ellen M. Voorhees. Implementing allgomerative hierarchic clustering algorithms for use in document retrieval. In *Information Processing & Management*, volume 22:6, pages 465–476, 1986.

[27] S.M. Weiss, C. Apte, F. Damerau, D. Johnson, F.J. Oles, T. Goetz, and T. Hampp. Boosting text-mining performance. *IEEE EXPERT, Special Issue on Applications of Intelligent Information Retrieval*, 1999.

[28] E. Wiener, J.O. Pedersen, and A.S. Weigend. A neural network approach to topic spotting. In *Proceedings of the Fourth Annual Symposium on Document Analysis and Information Retrieval (SDAIR'95)*, 1995.

[29] R. Willett. Recent trends in hierarchic document clustering: a critical review. *Information Processing and Management.*, 25(5):577–597, 1988.

[30] Y. Yang. Expert network: Effective and efficient learning from human decisions in text categorization and retrieval. In *17th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'94)*, pages 13–22, 1994.

[31] Y. Yang. An evaluation of statistical approaches to text categorization. *Journal of Information Retrieval (to appear)*, 1:69–90, 1999.

[32] Y. Yang and X. Liu. A re-examination of text categorization methods. In *The 22th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99)*, page (to appear in August 99), 1999.

[33] Y. Yang, T. Pierce, and J. Carbonell. A study on retrospective and on-line event detection. In *Proceedings of the 21th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98)*, pages 28–36, 1998.

[34] Oren Zamir and Oren Etzioni. Web document clustering: A feasibility demonstration. In *Proceedings of the 21th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98)*, pages 46–54, 1998.