UMass at TDT 2000

James Allan, Victor Lavrenko, David Frey, and Vikas Khandelwal

Center for Intelligent Information Retrieval
Department of Computer Science
University of Massachusetts
Amherst, MA 01003

We spent a fair amount of time this year rewriting our TDT system in order to provide more flexibility and to better integrate the various components. The time spent rearchitecting the code, learning to deal with its peculiarities, and correct bugs detracted substantially from research this year. As a result, the major approaches used on this evaluation are very similar to those used in TDT 1999.

We had two thrusts to our research, neither of which was ready to be deployed in this evaluation. We report here on the results from the training data, in all cases explored within the link detection task. In the first direction, we looked more carefully at score normalization across different languages and media types. We found that we could improve results noticeably though not substantially by normalizing scores differently depending upon the source language. In the second direction, we considered smoothing the vocabulary in stories using a "query expansion" technique from Information Retrieval to add additional words from the corpus to each story. This resulted in substantial improvements.

1. BASIC SYSTEM

The core of our TDT system uses a vector model for representing stories—i.e., we represent each story as a vector in term-space, where coordinates represent the frequency of a particular term in a story. Terms (or features) of each vector are single words, reduced to their root form by a dictionary-based stemmer. This system is based on one that was originally developed for the 1999 summer workshop at Johns Hopkins University's Center for Language and Speech Processing.[1] It was substantially reworked to provide improved support for "language model" approaches to the TDT tasks, though that functionality was not used significantly for TDT 2000.

1.1. Detection algorithms

Our system supports two models of comparing a story to previously seen material: centroid (agglomerative clustering) and nearest neighbor comparison.

Centroid In this approach, we group the arriving documents into clusters. The clusters represent topics that were discussed in the news stream in the past. Each cluster is represented by a *centroid*, which is an average of the vector representatives of the stories in that cluster.

Incoming stories are compared to the centroid of every cluster, and the closest cluster is selected. If the similarity of the story to the closest cluster exceeds a threshold, θ_{match}, we declare the story old; if the similarity exceeds a second threshold, $\theta_{certain}$, we add the new story to the topic and adjust the cluster centroid. If the similarity does not exceed θ_{match}, we declare the story new, and create a new singleton cluster with the story as its centroid. Both thresholds are set globally and apply to all clusters.

k-nearest neighbor The second approach, k-NN, does not attempt to explicitly model a notion of a topic, but instead declares a story to on the topic of the existing story most similar to it. That is, incoming stories are directly compared to all the stories we have seen before. The most similar k *neighbors* are found, and if the story's similarity to the neighbors exceeds a threshold, the story is declared to be on the same topic. Otherwise, if the story does not exceed that similarity with any existing story, the incoming story is declared the start of a new topic. In this work, we focused primarily on $k = 1$.

1.2. Similarity functions

One important issue in our approach is the problem of determining the right similarity function. We considered four functions: cosine, weighted sum, language models, and Kullbach-Leiblar divergence. The critical property of the similarity function is its ability to separate stories that discuss the same topic from stories that discuss different topics. For TDT 2000 we used only the cosine function, since our previous work had shown it provided substantial advantages and was more stable. Descriptions of the other techniques are provided for comparison.

Cosine The cosine similarity is a classic measure used in Information Retrieval, and is consistent with a vector-space representation of stories. The measure is simply an inner product of two vectors, where each vector is normalized to unit length. It represents the cosine of the angle between the two vectors $\vec{d}$ and $\vec{q}$.

$$\left(\sum q_i d_i\right) / \sqrt{\left(\sum q_i^2\right)\left(\sum d_i^2\right)}$$

(Note that if $\vec{q}$ and $\vec{d}$ have unit length, the denominator is 1.0 and the angle is calculated by a simple dot product.) Cosine similarity tends to perform best at full dimensionality, as in the case of comparing two long stories. Performance degrades as one of the vectors becomes shorter. Because of the built-in length normalization, cosine similarity is less dependent on specific term weighting, and performs well when raw word counts are presented as weights.

Weighted sum The weighted sum is an operator used in the InQuery retrieval engine developed at the Center for Intelligent Information Retrieval (CIIR) at the University of Massachusetts. InQuery is a Bayesian inference engine with transition matrices restricted to constant-space deterministic operators (e.g., AND, OR, SUM). Weighted sum represents a linear combination of evidence with weights representing confidences associated with various pieces of evidence:

$$\left(\sum q_i d_i\right) / \left(\sum q_i\right)$$

where q represents the *query* vector and d represents the *document* vector. (InQuery does not include a notion of vectors, but we have mapped the InQuery ideas into our vector-based implementation.) For instance, in the centroid model, cluster centroids represent *query* vectors which are compared against incoming *document* vectors.

Weighted sum tends to perform best at lower dimensionality of the query vector q. In fact, it was devised specifically to provide an advantage with short user requests typical in IR. The performance degrades slightly as the number of entries in q grows. In addition, weighted sum performs considerably better when combined with traditional tf·idf weighting (discussed below).

Language model Language models furnish a probabilistic approach to computing similarity between a document and a topic (as in centroid clustering) or two documents (nearest neighbor). In this approach, previously seen documents (or clusters) represent models of word usage, and we estimate which model M (if any) is the most likely source that could have generated the newly arrived document D. Specifically, we are estimating $P(D|M)/P(D)$, where $P(D)$ is estimated using the background model $P(D|GE)$ corresponding to word usage in General English.

By making an assumption of term independence (unigram model), we can rewrite $P(D|M) = \prod_i P(d_i|M)$, where d_i represent individual tokens in D. We use a maximum likelihood estimator for $P(d_i|M)$, which is simply the number of occurrences of d_i in M divided by the total number of tokens in M. Since our models may be sparse, some words in a given document D may have zero probability under any given model M, resulting in $P(D|M) = 0$. To avoid this problem we use a smoother estimate $P(d_i|M) = \lambda P_{ml}(d_i|M) + (1-\lambda)P(d_i|GE)$, that allocates a non-zero probability mass to the terms that do not occur in M. We set λ to the Witten-Bell[3] estimate $N/(N+U)$ where N is the total number of tokens in the model and U is the number of unique tokens. (Note that since detection tasks are online tasks, we may encounter words not in GE, and so we smooth GE in a similar fashion using a uniform model for the unseen words.)

Kullbach-Leiblar divergence Instead of treating a document D as a sample that came from one of the models, we could view D as a distribution as well, and compute an information-theoretic measure of divergence between two distributions. One measure we have experimented with is the Kullbach-Leiblar divergence, $KL(D, M) = -\sum_i d_i \log(m_i/d_i)$, where d_i and m_i represent relative frequencies of word i in D and M respectively (both smoothed appropriately).

1.3. Feature weighting

Another important issue is weighting of individual features (words) that occur in the stories. The traditional weighting employed in most IR systems is a form of tf·idf weighting.

Inquery The *tf* component of the weighting—the number of times a term occurs in a document—represents the degree to which the term describes the contents of a document. The *idf* component—the inverse of the number of documents in which a term occurs—is intended to discount very common words in the collection (e.g., function words) since they have little discrimination power. Below is the particular tf·idf scheme used in the InQuery engine:

$$tfcomp = \frac{tf}{tf + 0.5 + 1.5\frac{len_d}{len_{avg}}}$$

$$idfcomp = \frac{\log(N/df)}{\log(N+1)}$$

The *tf-comp* component has a general form of $\mathrm{tf}/(\mathrm{tf}+K)$, where *tf* is the raw count of term occurrences in the document, and *K* influences the significance we attach to seeing consecutive occurrences of the term in a particular document. The functional form is strictly increasing and asymptotic to 1.0 as *tf* grows without bounds. The effect is that we assign a lot of significance to observing a single occurrence of a term, and less and less significance to consecutive occurrences. This is based on the observation that documents that contain an occurrence of a given word w are more likely to contain successive occurrences of w.

The parameter *K* influences how aggressively we discount successive occurrences, and in InQuery is set to be the document length over average document length in the collection. This means that shorter documents will have more aggressive discounting, while longer stories will not assign a lot of significance to a single occurrence of a term. This form of the *tf* component is generally referred to as "Okapi tf" since it was first introduced as part of the Okapi system.[2]

The *idf-comp* component is the logarithm of the inverse probability of the term in the collection, normalized to be between 0 and 1. *N* denotes the total number of documents in the collection, while *df* shows in how many of those documents the term occurs. This particular *idf* formulation arises naturally in the probabilistic derivation of document relevance under the assumption of binary occurrence and term independence.

tf This weighting scheme is simply the actual *tf* value used in the *tfcomp* formula above—i.e., the number of times the term occurs in the story. The intuition behind omitting the *idf* component is that feature selection at other points in the process will choose only medium- and high-idf features with good discrimination value. As a result, the *tf*-only weighting scheme is less likely to work at high dimensionality when low-idf features will appear and need to be down-weighted.

tf·idf This weighting scheme is simply the raw *tf* component times the *idf* component of the tf·idf scheme. This weighting method boosts the importance of multiple occurrences of a feature over that given in the tf·idf scheme. This approach turns out to be the most successful in our TDT 2000 research.

2. TRACKING

Our research was focused on Story Link Detection (Section), so we did not try anything unusual for tracking this year. We spent time rechecking our parameter choices by sweeping a range of values. In the end, we settled on centroid representation of topics (i.e., average all N_t training stories together), and cosine comparison of stories to topics. The other parameters (weighting, number of features, adapting thresholds) were chosen by a parameter sweep as shown in Table 1.

It is interesting to note that difference between effectiveness of Inquery's weighting function (Okapi tf component) compared to just using the tf count directly. This difference is surprising because the Okapi tf function has been widely adopted in IR—yet here it appears to be less useful. We posit this is because the Okapi tf function is valuable for high-precision (low false alarm) tasks such as information retrieval. In the TDT tracking task, the optimum score is in a part of the error tradeoff curve that is less significant for IR.

We normalized the scores by comparing all N_t training stories to the centroid and then finding the average of those N_t similarities. During tracking, all subsequent story similarities were divided by that average score. So an "average on-topic story" would have a score of 1.0.

If the topic was adapted, the average was recalculated using the original N_t training stories as well as the stories that had been included in the topic. This year, adapting did not provide any reduction in the cost, and usually helped. This is consistent with results from TDT 1998, though continues to surprise us.

We selected using 1000 features (the full story), tf·idf weighting of those features, and no adapting. The threshold was selected depending on the task, as follows:

$N_t = 1$	manual boundaries	0.07
$N_t = 1$	auto boundaries	0.13
$N_t = 4$	manual boundaries	0.07
$N_t = 4$	auto boundaries	0.13

The threshold was chosen by sweeping through the scores on the training data and finding the threshold that yielded the best normalized tracking cost.

3. CLUSTER DETECTION

Our clustering approach used 1-NN story comparison, so that a story was added into the topic that contained a *single* story to which it was very similar. Comparison was done using the cosine measure. Idf values were calculated using a retrospective corpus (the six-month TDT-2 collection).

Table 2 shows the result of the parameter sweep for selecting the comparison function, the weighting, and the threshold θ_{match}.

As part of a cooperative project with BBN's Oasis system, we have begun looking at cluster detection on "real world" data and in a "real world" evaluation setting. It is obviously from the very first attempts that 1-NN cluster formation will not be appropriate. The created clusters have a property that is common among algorithms of the "single link" genre: they tend to be "stringy" with stories that are linked together in long chains, but that may not hold together as a group. Using the optimal settings trained on the TDT-2 corpus (i.e., our TDT 2000 parameters), we found clusterings containing 100s of at best marginally related stories.

The evaluation measure currently used in TDT rewards a system for getting the bulk of a topic's stories together, and does not appear to penalize enough for mistakes. At a minimum that means that the cost values for detection need to be different for the Oasis task. At worst, it means that the detection cost function is inappropriate.

Weighting	#Terms	Adapting	min $(C_{track})_{norm}$
Reference boundaries, $N_t = 4$			
tf·idf	1000	no	0.2255
tf·idf	100	no	0.2560
tf·idf	50	no	0.2992
tf·idf	20	no	0.3718
tf·idf	10	no	0.4082
Inquery	1000	no	0.6038
Inquery	100	no	0.2663
Inquery	50	no	0.3102
Inquery	20	no	0.3761
Inquery	10	no	0.5879
Reference boundaries, $N_t = 1$			
tf·idf	1000	no	0.2673
tf·idf	100	no	0.2906
tf·idf	50	no	0.3311
tf·idf	20	no	0.3751
tf·idf	10	no	0.4487
tf·idf	1000	1.0	0.2673
tf·idf	1000	0.9	0.2673
tf·idf	1000	0.8	0.2673
tf·idf	1000	0.7	0.3550
Inquery	1000	no	0.5301
Inquery	100	no	0.7825
Inquery	50	no	0.6675
Inquery	1000	1.0	0.5301
Inquery	1000	0.9	0.5301
Inquery	1000	0.8	0.5301
Inquery	1000	0.7	0.5301
Automatic boundaries, $N_t = 4$			
tf·idf	1000	no	0.2586
tf·idf	1000	1.0	0.3146
tf·idf	100	no	0.2840
tf·idf	100	1.0	0.3451
Automatic boundaries, $N_t = 1$			
tf·idf	1000	no	0.9533
Inquery	1000	no	0.9720
Inquery	1000	1.0	0.9816
Inquery	1000	0.9	0.9730

Table 1: Result of parameter sweep for tracking run on TDT-2 training data.

4. FIRST STORY DETECTION

Our first story detection system was run identically to the cluster detection system, except that we selected a different threshold because of the different evaluation measure. The emitted score was one minus the detection score—i.e., the confidence that this story is new (rather than on a topic).

Idf was calculated from a retrospective corpus (the six-month TDT-2 collection), we chose the tf·idf weighting scheme, cosine comparison, and 1000 features per story (all features). We selected 0.20 as

Compare	Weight	Threshold	Cost
cosine	tf·idf	0.04	0.9253
cosine	tf·idf	0.06	0.7707
cosine	tf·idf	0.08	0.5981
cosine	tf·idf	0.10	0.4673
cosine	tf·idf	0.16	0.2604
cosine	tf·idf	0.18	0.2334
cosine	tf·idf	0.20*	0.2193
cosine	tf·idf	0.22	0.2212
cosine	Inquery	0.02	1.0000
cosine	Inquery	0.04	1.0000
cosine	Inquery	0.06	0.9904
cosine	Inquery	0.14	0.6219
cosine	Inquery	0.16	0.5289
cosine	Inquery	0.18	0.4383
wsum	tf·idf	0.02	0.9804
wsum	tf·idf	0.04	0.9804
wsum	tf·idf	0.06	0.9569
wsum	tf·idf	0.08	0.9569
wsum	tf·idf	0.10	0.9569
wsum	tf·idf	0.12	0.9569
wsum	tf·idf	0.16	0.9560
wsum	tf·idf	0.18	0.9560
wsum	tf·idf	0.20	0.9246
wsum	tf·idf	0.22	0.9035
wsum	tf·idf	0.24	0.8934
wsum	tf·idf	0.26	0.8835
wsum	Inquery	0.02	0.9245
wsum	Inquery	0.04	0.9245
wsum	Inquery	0.06	0.8393
wsum	Inquery	0.08	0.5422
wsum	Inquery	0.10	0.3560
wsum	Inquery	0.12	0.2932
wsum	Inquery	0.14	0.2713
wsum	Inquery	0.16	0.2832
wsum	Inquery	0.18	0.3101
wsum	Inquery	0.20	0.3624
wsum	Inquery	0.22	0.3872
wsum	Inquery	0.24	0.4192

Table 2: Result of parameter sweep for cluster detection run on TDT-2 training data.

the threshold—the same value as used in clustering, despite the different measures. We are somewhat surprised by this result, but have not yet investigated it.

5. STORY LINK DETECTION

Our link detection submission did not include any novel results. However, we report here on some preliminary results that were showing us improvements in link detection. We exploring how a query expansion technique from information retrieval could smooth the compared stories, and how score normalization depending on language mix can improve results.

Weight	Thresh	Norm(C_{link})
tf·idf	0.02	1.6619
tf·idf	0.04	0.6322
tf·idf	0.045	0.5362
tf·idf	0.05	0.4591
tf·idf	0.055	0.4099
tf·idf	0.06	0.3769
tf·idf	0.065	0.3523
tf·idf	0.07	0.3412
tf·idf	0.075	0.3289
tf·idf	0.08*	0.3200
tf·idf	0.085	0.3235
tf·idf	0.09	0.3216
tf·idf	0.10	0.3248
tf·idf	0.12	0.3583
tf·idf	0.14	0.4084
tf·idf	0.16	0.4641
Inquery	0.02	4.2889
Inquery	0.04	3.3705
Inquery	0.045	3.1142
Inquery	0.05	2.8463
Inquery	0.055	2.5871
Inquery	0.06	2.3356
Inquery	0.065	2.1033
Inquery	0.07	1.8761
Inquery	0.075	1.6715
Inquery	0.08	1.4895
Inquery	0.085	1.3109
Inquery	0.09	1.1864
Inquery	0.10	0.9522
Inquery	0.12	0.6969
Inquery	0.14	0.5994
Inquery	0.16	0.6063

Table 3: Result of parameter sweep for link detection run on TDT-2 training data.

5.1. Submitted SLD

Here we are comparing two stories. We ran a parameter sweep to select the weighting scheme and the threshold for comparison. We found that cosine comparison of tf·idf weights with a threshold of 0.80 worked best. Idf scores were taken from a retrospective corpus (TDT-2's six-month corpus). Table 3 shows the cost function varying over a range of parameter values.

5.2. LCA smoothing

In SIGIR 1996, the CIIR presented a query expansion technique that worked more reliably than previous "pseudo relevance feedback" methods.[4] That technique, Local Context Analysis (LCA), locates expansion terms in top-ranked passages, uses phrases as well as terms for expansion features, and weights the features in a way intended to boost the expected value of features that regularly occur near the query terms.

Because LCA has been so successful in IR tasks, we felt it was appropriate to explore it as a smoothing technique in TDT's story link detection task. That is, each story is treated as a "query" and expanded using LCA. Additional words that occur in the corpus very near the words in the story are added into each story and the resulting, larger, stories are compared as before.

We first provide some details about how LCA works, and then discuss its explicit use and results in TDT.

LCA used for SLD We used LCA query expansion to replace the original story vector with a different, smoothed one. We first converted the story to a vector as before, selecting either Inquery or tf·idf as a weighting function. We then select the n most highly weighted features from that vector and discard all other features.

Those n features are used as a query to find the s stories from the TDT-3 corpus that are most similar to features (as vectors). Except where noted otherwise below, we only allow those stories to come from stories that appeared *before* the story being expanded. (We could have used any stories up until the later of the two stories, but have not yet explored that adjustment.)

We extract all features from those s stories and weight them based upon their proximity to the original n "query" features. The LCA weighting function is a complex heuristic that gives higher weights to features that occur with many query words.[4] We select the top n LCA expansion features and add them to the vector. Note that it is possible for some of the *original* n features to re-appear as LCA features. The resulting vector has anywhere from n to $2n$ unique features.

The new features are added in with weights that start at 1.0 and smoothly drop down to $1.0 - (n-1)0.9/n$. This is the common weighting function for LCA features, and may not be the best choice for adding into the vector.

The result is that a story's vector is replaced by n to $2n$ features with weights that are a combination of Inquery or tf·idf weights, and LCA weights.

For this study we used $s = 20$ stories for expansion, used $n = 100$ features from each story, and added $n = 100$ expansion features.

LCA/SLD experiments Figure 1 shows the impact of story smoothing using LCA on the link detection task. The curve that is consistently worst is the DET plot for no smoothing at all: our base case. The next curve toward the original (it moves closest to the origin at both ends) is the result of using LCA as described above. The curve that comes closest to the origin is a "cheating" run that uses the *entire* TDT-3 corpus for expansion, meaning that a story could be expanded by stories that follow it and not just those in the past. Even without looking ahead, the value of LCA smoothing is apparent.

For our experiments, we used either the Inquery or the tf·idf weighting function both for determining the top n features of the story, and for finding the best-matching stories for expansion. Our best results in non-LCA SLD were obtained with the tf·idf weighting function, but with LCA, Inquery weights performed better. Why?

We hypothesize that the reason is that query expansion requires highly accurate retrieval of the type that is typical in an IR system. The cost of expanding using non-relevant passages is very high: the query will be expanded in a direction that is not related to th original request. Our tf·idf weight is well known to be less effective in IR, so we expect it generates less relevant expansion terms. Since those terms account for up to half of the story's representation, it is very important that they be accurate.

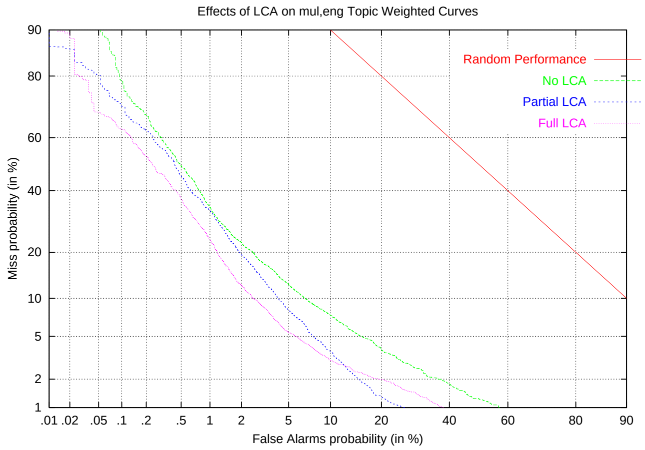

Figure 1: Results of LCA smoothing on SLD task. Experiments were done on the TDT-2 corpus.

5.3. Cross-language score normalization

Effects of SYSTRAN translations During our experiments we stumbled upon an interesting effect of Mandarin documents on performance. We observed that the performance of our story-link detection system was noticeably worse on a multi-lingual dataset than it was on the English-only data. We hypothesized that the drop in performance could be due to lexical differences between the use of language in native English stories and in SYSTRAN translations of Chinese stories.

To test this hypothesis we performed the following post-hoc experiment. We partitioned our set of story pairs into three subsets: (1) pairs where both stories are native English stories, (2) pairs where both stories are SYSTRAN translations of Chinese, and (3) pairs where one story is a native English story and the other is the SYSTRAN translation. Then we analyzed the distributions of similarities of stories in the pair for each subset. Figure 2 presents distribution plots separately for on-target (both stories discuss the same topic) and off-target (stories discuss different topics) pairs in each subset.

It is evident that similarity distributions are very different for different subsets of pairs. On average, two SYSTRAN stories have a higher expected similarity than do two native English stories; the expected similarity of a SYSTRAN story to a native English story is even lower. Note that this observation holds for both on-target and off-target story pairs, but the effect is much more pronounced for on-target pairs.

We suspect the differences are due to the limited vocabulary of SYSTRAN translations. Any machine translation system, including SYSTRAN, has a relatively small vocabulary, whereas native English authors tend to use a much wider range of words. Also, SYSTRAN uses words consistently from story to story, whereas different human authors tend to use different words to describe the same idea. Inconsistent use of words leads to smaller expected word overlap be-

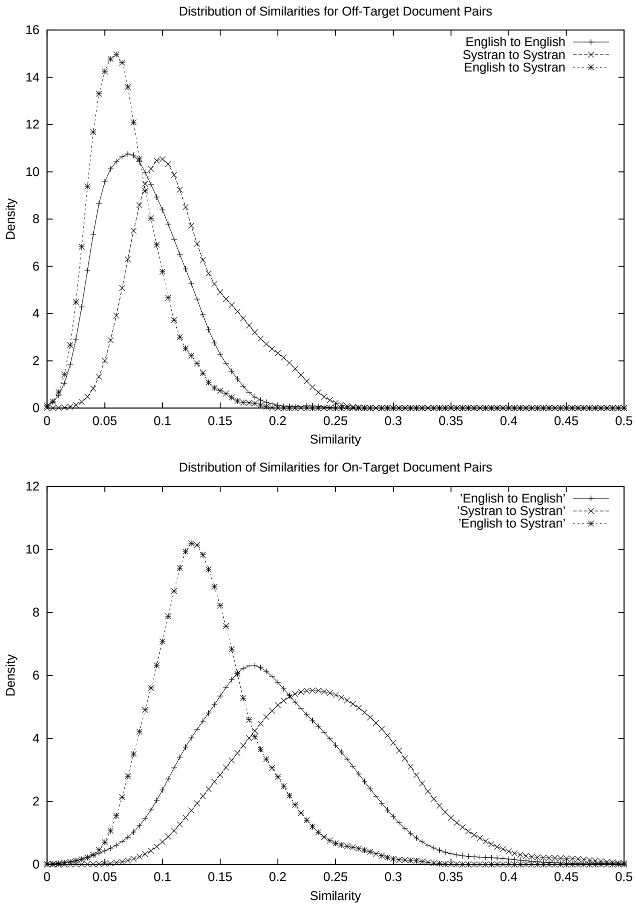

Figure 2: Effect of language on distributions of story similarities. Top: off-target story pairs. Bottom: on-target story pairs

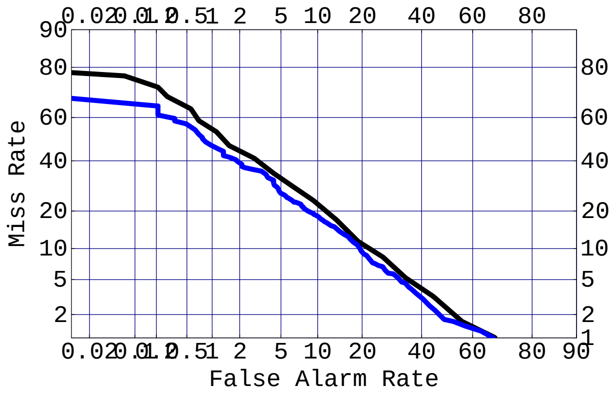

Figure 3: Improvement in performance resulting from normalization of similarities. Lower curve represents normalized system.

tween any two stories, which translates to lower expected similarity between two stories.

Whatever the cause, the differences in similarities present a serious challenge to effective cross-lingual story-linking. Suppose two given stories have a similarity of 0.1. If we know that both stories are SYSTRAN translations, the pair is most-likely off-target (from Figure 2 we see that probability of getting a 0.1 similarity in an on-target SYSTRAN pair is extremely low). However, if we know that one story is native English, and the other is a SYSTRAN translation, the pair is most-likely on-target, since the probability of getting 0.1 is higher for on-target pairs (Figure 2). This example implies that our similarity values are not directly comparable when pairs of stories involve multiple languages. To make them comparable, we need to normalize the similarities with respect to the source of stories in the pair.

Compensating translation effects There exist a number of normalization techniques, ranging from simple range normalization and linear scaling (used in our tracking approach) to more elaborate techniques. We consider a probabilistic normalization technique where we replace the similarity x of a pair from subset S with the posterior probability that the pair is on-target $P(T|x,S)$, given the similarity x and subset S. If we have access to distributions of on-target similarities $P(x|T,S)$ and off-target similarities $P(x|N,S)$, we can use Bayes rule to derive the posterior:

$$P(T|x,S) = \frac{P(x|T,S)P(T,S)}{P(x|T,S)P(T,S) + P(x|N,S)P(N,S)}$$

Note that estimating the posterior requires knowledge of relevance judgments for each pair (to estimate $P(x|T,S)$ and $P(x|N,S)$). What we would do in practice is estimate the probabilities from the training data and then apply the transformation to the similarities in the testing data.

A number of parametric and non-parametric techniques could be used to estimate the conditional densities $P(x|T,S)$ and $P(x|N,S)$. In this work we chose non-parametric *kernel* density estimators because they can provide an arbitrarily close fit to the training data ("Applied Smoothing techniques for Data Analysis" A.Bowman, A.Azzalini). The conditional probability of x is a function of every story pair in the training set S:

$$P_{\phi,h}(x|S) = \frac{1}{h|S|}\sum_{y\in S}\phi(\frac{x-y}{h})$$

Here ϕ is the *kernel*, which can be any probability density function, and h is the *bandwidth* parameter, representing the desired degree of smoothness. For kernel estimators the choice of ϕ has very little effect, as long as it is unimodal, symmetric and smooth. We selected Gaussian kernels:

$$\phi(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}$$

Bandwidth h, on the other hand, has very strong effects on the final distribution. We uses automatic bandwidth selection technique (described in on p.31 of "Applied Smoothing techniques for Data Analysis" A.Bowman, A.Azzalini).

Figure 3 shows the effects of applying our normalization to the training set of story-link pairs. System that used normalized similarities shows a small but consistent improvement over no normalization.

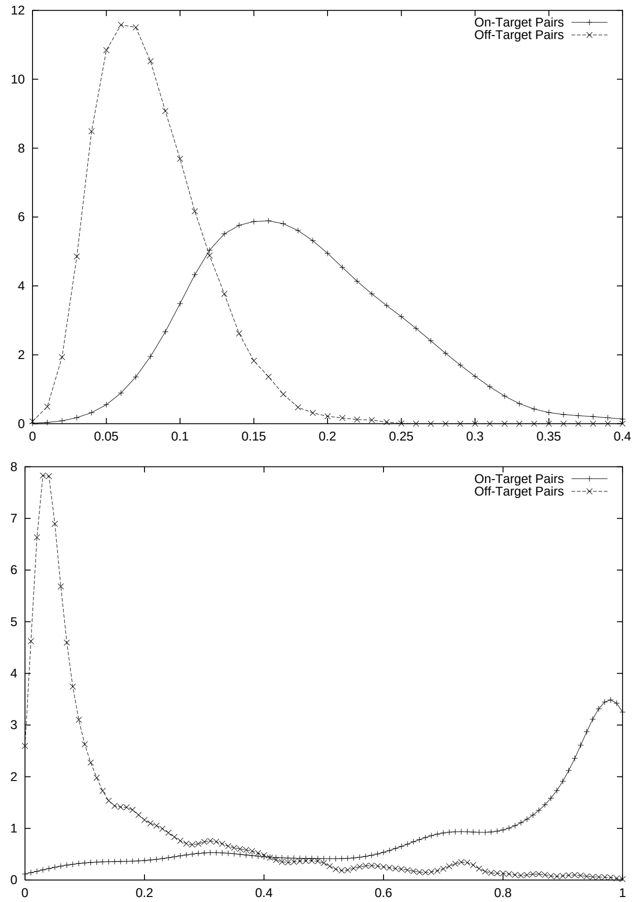

Figure 4: Effect of score normalization on similarity distributions. Top: distributions before normalization. Bottom: after normalization.

In this case we performed a cheating experiment, using the training data to normalize itself.

To better understand the effects of our normalization we plotted the overall densities of the original similarities (top half of Figure 4), and normalized similarities (bottom half). The main effect is in spreading the distributions apart. However, our normalization also introduces very “heavy” tails in both densities on the bottom half of Figure 4, and the tails are “bumpy”, which means that our normalization is non-monotonic (higher similarities don’t always mean higher probability of being on-target). We suspect that bumpiness is the result of over-fitting the density. Possible ways to avoid this problem would be to increase the bandwidth h or use a parametric density estimator instead of kernel estimator described above.

6. CONCLUSION

The bulk of our effort this half year was spent re-engineering our TDT system so that it could better support our longer-term research goals. In particular, we are modifying the system to provide better capabilities in the area of language modeling, consistent with our broader goals of formally modeling information organization tasks.

We have some preliminary work that shows the value of smoothing stories by other, related stories in the corpus. We are simultaneously working on improved formal models for query expansion, and anticipate incorporating that approach into our language modeling ideas.

Score normalization is a key task within TDT that has not been important in areas such as information retrieval. We have been using distribution plots to recognize when normalization is likely to be helpful, and have shown that definitely helps within and across languages.

Acknowledgments

This work was supported in part by the National Science Foundation, Library of Congress, and Department of Commerce under cooperative agreement number EEC-9209623, in part by SPAWAR-SYSCEN-SD grant number N66001-99-1-8912. The opinions, views, findings, and conclusions contained in this material are those of the authors and do not necessarily reflect the position or policy of the Government and no official endorsement should be inferred.

References

1. J. Allan, H. Jin, M. Rajman, C. Wayne, D. Gildea, V. Lavrenko, R. Hoberman, and D. Caputo. Topic-based novelty detection: 1999 summer workshop at CLSP, final report. Available at http://www.clsp.jhu.edu/ws99/tdt, 1999.
2. S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In D. K. Harman, editor, *The Third Text REtrieval Conference (TREC-3)*. NIST, 1995.
3. I.H. Witten and T.C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37:1085–1094, 1991.
4. J. Xu and W. B. Croft. Query expansion using local and global document analysis. In *Proceedings of the 19th annual international ACM SIGIR conference on research and development in information retrieval*, pages 4–11, Zurich, 1996. Association for Computing Machinery.