AT&T at TREC-7

Amit Singhal John Choi Donald Hindle David D. Lewis
Fernando Pereira

AT&T Labs–Research
{singhal,choi,hindle,lewis,pereira}@research.att.com

Abstract

This year AT&T participated in the ad-hoc task and the Filtering, SDR, and VLC tracks. Most of our effort for TREC-7 was concentrated on SDR and VLC tracks. On the filtering track, we tested a preliminary version of a text classification toolkit that we have been developing over the last year. In the ad-hoc task, we introduce a new tf-factor in our term weighting scheme and use a simplified retrieval algorithm. The same weighting scheme and algorithm are used in the SDR and the VLC tracks.

The results from the SDR track show that retrieval from automatic transcriptions of speech is quite competitive with doing retrieval from human transcriptions. Our experiments indicate that document expansion can be used to further improve retrieval from automatic transcripts. Results of filtering track are in line with our expectations given the early developmental stage of our classification software. The results of VLC track do not support our hypothesis that retrieval lists from a distributed search can be effectively merged using only the initial part of the documents.

1 Introduction

Spoken document retrieval (SDR) and retrieval from very large collections (VLC) are the main areas of interest for AT&T at TREC-7. For most of our work, we use an internally modified version of the SMART retrieval system developed at Cornell University. [1, 9] We are in the process of developing a text classification toolkit called ATTICS. The filtering track gave us an opportunity to stress test an early version of this toolkit on a large task.

For speech retrieval, we believe that parallel text corpora, for example printed news from the same time period, can be successfully exploited to improve retrieval effectiveness of a system. This is especially true for the news material currently being used in the SDR track. We use these ideas in our SDR track participation. Initial results from the use of a parallel corpus are quite encouraging.

As the amount of data available electronically (for example on the Web) grows exponentially, it is becoming increasingly hard to maintain a single centralized index to the data. Distributed retrieval is a solution; however most distributed retrieval algorithms expect some cooperation from the individual servers, which is often hard or even impossible to get. In the VLC track we simulate "totally independent" distributed collections, and use a new result merging algorithm that leverages the summary text for retrieved documents, usually provided by the search engines, to merge results from various servers.

2 Ad-hoc Runs

Over the last year, we have modified the SMART retrieval system with the following minor changes: 1) we use a shorter stop-list of 410 stopwords instead of the standard SMART stop-list of 571 stopwords, 2) we use a new tokenizer that handles hyphens and ampersands (as in AT&T) in a better manner, 3) phrase formation is governed by a new set of rules, and 4) we have generated a new set of statistical phrases (187,908 phrases) from the news corpora included in disks 1–5 (AP, WSJ, SJMN, FT, FBIS, LATimes). These phrases are used in ad-hoc, SDR, and the SMART run of the filtering track. We do not use any phrases in the VLC track.

d *tf* factor:	$1 + ln(1 + ln(tf))$	$0 \ if \ tf = 0$
t *idf* factor:	$log(\frac{N+1}{df})$	
b pivoted byte length normalization factor:	$\frac{1}{0.8 + 0.2 \times \frac{length\ of\ document\ (in\ bytes)}{average\ document\ length\ (in\ bytes)}}$	

where, tf is the term's frequency in text (query/document)
N is the total number of documents in the (training) collection
df is the number of documents that contain the term, and
the average document length depends on the collection.

dnb weighting: **d** factor $\times$ **b** factor
dtb weighting: **d** factor $\times$ **t** factor $\times$ **b** factor
dtn weighting: **d** factor $\times$ **t** factor

Table 1: Term Weighting Schemes

Poor performance of the logarithmic *tf*-factor—$1 + ln(tf)$—on the TREC-6 ad-hoc task forced us to revisit our term weighting methodology. The main reasoning for use of a logarithmic *tf*-factor instead of (say) the raw *tf* of the terms is to ensure that high *tf* for one query term in a document does not place that document ahead of other documents which have multiple query terms but with low *tf* values. By using a logarithmic *tf*-factor, the effect of *tf* dampens with increasing *tf* values, and a single match usually doesn't outweigh multiple matches. We found that for short queries (our main interest in the ad-hoc task) a logarithmic *tf*-factor does not adequately reduce the effect of large *tf* values. Therefore we need a *tf*-function that reduces the *tf* contribution even more than the logarithmic *tf*-factor as *tf* increases. Double logarithm—$1 + ln(1 + ln(tf))$—is an obvious choice. We also like double logarithm since it doesn't introduce any new parameters into our weighting scheme.

Next we revisited our pivoted unique document length normalization function. [12] When both words and phrases are used as terms in a document vector, the definition of the number of unique terms in the document is not elegant, especially since phrases are formed from words but are counted as independent unique terms. Earlier work has shown that byte length of a document can be successfully used in a pivoted formula for document length normalization. [12] We switch to using *pivoted byte size normalization* in our weighting scheme. As a side benefit, the weighting code inside SMART is simplified when pivoted byte size normalization is used. With these changes, we use the term weighting schemes shown in Table 1.

2.1 Approach

This year we use a simple two pass *pseudo-feedback* based approach in the ad-hoc task. Many groups have used such an approach in the last few TRECs. Here are the steps in the process:

- **Pass-1**: Using *dtn* queries and *dnb* documents, a first-pass retrieval is done.
- *Expansion:* Top ten documents retrieved in the first pass are *assumed* to be relevant to the query and documents ranked 501–1000 are assumed to be non-relevant. Rocchio's method (with parameters $\alpha = 3,\ \beta = 2,\ \gamma = 2$) is used to expand the query by adding twenty new words and five new phrases with highest Rocchio weights. [8] To include the *idf*-factor in the expansion process, documents are *dtb* weighted.
- **Pass-2:** The expanded query is used with *dnb* documents to generate the final ranking of 1,000 documents.

Task Title+Desc	Baseline *dnb.dtn*	Expansion from target collection	Expansion from TREC D12345	Conservative Collection Enrichment
TREC-6 (AvgP)	0.2290	0.2661 (+16.2%)	0.2781 (+21.4%)	0.2906 (+26.9%)
# Q better/worse		0/0	31/19	34/16
TREC-7 (**att98atdc**)	0.2182	0.2830 (+29.7%)	0.2861 (+31.1%)	**0.2961** (+33.7%)
# Q better/worse		0/0	25/25	32/18

Table 2: Effect of conservative collection enrichment

2.2 Conservative Collection Enrichment

At TREC-6 some groups (*e.g.*, City University and University of Massachusetts) used a technique that Kwok calls "collection enrichment". [5] The main idea is to run the first pass on a much larger collection than the target collection. The hope is that a larger text collection will have more relevant documents for the query and our methods will pull more relevant documents in the top ten or twenty documents, thereby strengthening the assumption of relevance employed in the query expansion stage. We employ such a technique in our ad-hoc runs using all TREC disks (1–5) as the large collection.

One obvious problem with this approach is possible "domain mismatch". If the top documents retrieved from the large collection are from a completely different domain than the top, presumably query-related, documents in the target database, collection enrichment can cause the query to drift away from target relevance. During our test with collection enrichment, we found that this indeed was a problem. Therefore we devised a conservative collection enrichment technique which forces the expanded query to remain in the target domain by not allowing any expansion terms proposed only by expansion on the large collection.

For example, for query 354: *journalist risks*, the first pass from the large corpus (TREC disks 1–5) retrieves several stories that talk about a list of missing or detained U.N. workers around the world, some of them were killed, including a journalist. Even though these documents are reasonably relevant to the topic, the focus of these documents is different than the topical documents in the target database. These documents add terms related to the U.N. and its missing employees to the query and drift the query away from relevance in the target database which does not have reports on missing U.N. journalists.

To fix this problem, we allow only those new terms to be added to the query that are proposed by top documents retrieved from the target database. To capture some effect of collection enrichment, we allow the large collection to impact the term selection and the final weights of the terms. Here are the steps involved in our conservative collection enrichment:

1. Use the query to retrieve the top ten document from the target collection and build a list of potential expansion terms along with their proposed Rocchio weights. Only terms that appear in at least two of the ten documents and have a positive Rocchio weight are considered.

2. Use the query to retrieve ten document from the large collection, and compute the Rocchio weights for all the expansion terms proposed in step 1.

3. Add term weights proposed in steps 1 and 2 and add the top weighted twenty words and five phrases to the original query. This allows for terms that are weakly proposed (*i.e.*, with a low weight) by the target collection but are strongly proposed by the larger collection to enter the query and vice-versa.

2.3 Results

We submitted two fully-automatic runs based on the above described algorithms. One run, **att98atc**, was based on title-only queries and the other run, **att98atdc**, used title+description queries. We do not use the *narrative*-portion of the TREC topics in any of our runs since we don't believe that real users will ever provide us with such long queries. To highlight the benefits of conservative collection enrichment, we first present the results of running this algorithm on last year's (TREC-6) adhoc task (title+description) and this year's task (our run att98atdc) in Table 2. The second column in Table 2 shows the baseline average

Run	Average Precision	Best	>= Median	< Median
att98atc (title only)	0.2488	0	32	18
att98atdc (title+desc)	0.2961	1	41	8
att97atde (title+desc)	0.2940	0	44	6

Table 3: Results for adhoc runs

precision of a straight retrieval when documents are *dnb* weighted and queries are *dtn* weighted. The third column shows the results when only the target database is used in the query expansion process. For the TREC-6 task, we get a 16.2% improvement in average precision. This improvement is almost 30% for this year's task. If we use the large collection for query expansion, column four shows that the performance improves some more. The last column shows that our conservative collection enrichment further improves the performance some.

Even though in terms of average precision, the conservative method is not much better than expanding purely from the large collection, but as the rows labeled *# Q better/worse* show, it is more stable with respect to the number of queries that improve or deteriorate in comparison to expansion from the target collection (the sensible baseline for this comparison since if we compare to unexpanded queries, which is the baseline used in the average precision rows, all expansion strategies will show large gains and the relative performance of different expansions will be hard to judge). For the TREC-6 task, when expansion is done from D12345, 31 queries improve but 19 queries deteriorate (column 4) in comparison to base expansion. But if we do conservative collection enrichment, we cut our losses to 16 queries instead, (column 5) and we gain some in average precision. These numbers are even more striking for this year's task. Expansion from the large collection is worse than expansion from the target collection for half the queries, it is better for the other half. But the conservative expansion reduces our losses from 25 queries to 18 queries only, and the losses are, in general, smaller. These results show that conservative collection enrichment is more stable than pure enrichment, even though the gains in term of average precision are not much.

The official results for our ad-hoc runs are shown in Table 3. Both the runs att98atc and att98atdc did well, especially given that the medians are drawn from a pool which includes runs that use the full topic text (rich with content words from the "narrative" portion of the topics) to construct the query. Our runs just use either very short queries (title only) or short queries (title+description). We also submitted an experimental run **att97atde** which enriched the term set by using word cooccurrence pairs as we have used in the past in the routing task. [11] We were hoping that these pairs would be useful in the ad-hoc setting too, but using word-pairs didn't improve our retrieval effectiveness.

3 SDR Runs

We use our own speech recognizer to process the SDR track data. One of our submitted runs **att-s1** uses the one-best recognizer transcript and does retrieval using an algorithm quite similar to the algorithm we have used in the ad-hoc task. The other run **att-s2** uses word-lattices generated by our recognizer and a parallel text corpora to do *document expansion*. The expanded documents are then used for retrieval instead of the one-best recognizer transcript.

3.1 Speech Recognizer

Our speech recognition process involves the following steps. Prior to recognition, each speech story is segmented into approximately one minute long prosodically well-formed segments using a CART based classifier. [2] The resulting segments are submitted to another wideband/narrowband classifier for selection of the acoustic model to be used in recognition of that segment.

The recognizer is based on a standard time-synchronous beam search algorithm. The probabilities defining the transduction from text-dependent phone sequences to word sequences are estimated on word level grapheme-to-phone mappings and are implemented in the general framework of weighted finite-state transducers. [7] Transducer composition is used to generate word lattice output.

We use continuous density, three-state, left-to-right, context-dependent hidden Markov phone models. These models were trained on 39-dimensional feature vectors consisting of the first 13 mel-frequency cepstral coefficients and their first and second time derivatives. Training iterations included eigenvector rotations, k-means clustering, maximum likelihood normalization of means and variances and Viterbi alignment. The output probability distributions consist of a weighted mixture of Gaussians with diagonal covariance, with each mixture containing at most 12 components. The training data were divided into wideband and narrowband partitions, resulting in two acoustic models.

Language Models

We used a two pass recognition process. In the first pass, we built word lattices for all the speech using a minimal trigram language model and a beam that we had determined heuristically to provide manageable word lattices. These word lattices were then rescored, by removing the trigram grammar weights while retaining the acoustic weights and intersecting these lattices with a 4-gram language model. The 1-best path was extracted from the rescored lattices.

Both the first pass trigram language model and the rescoring 4-gram model are standard Katz backoff models [4], using the same 237 thousand word vocabulary. For choosing the vocabulary, all of the words from the SDR98 training transcript were used. This base vocabulary was supplemented with all words of frequency greater than two appearing in the New York Times and LA Times segments of LDC's North American News corpus (LDC Catalog Number: LDC95T21, see `www.ldc.upenn.edu`), in the period from June 1997 through January 1998. The vocabulary includes about 5,000 common acronyms (e.g "N.P.R."), and the training texts were preprocessed to include these acronyms.

The language model training was based on three transcription sources (the SDR98 training transcripts, HUB4 transcripts, transcripts of NBC nightly news) and one print source (the LDC NA News corpus of newspaper text). The first-pass trigram model was built by first constructing a backoff language model from the 271 million words of training text, yielding 15.8 million 2-grams and 22.4 million 3-grams. This model was reduced in size, using the approach of Seymore and Rosenfeld [10], to 1.4 million 2-grams and 1.1 million 3-grams. When composed with the lexicon, this smaller trigram model yielded a manageable sized network. The second pass model used 6.2 million 2-grams, 7.8 million 3-grams, and 4.0 million 4-grams. For this model, the three transcription sources (SDR, HUB4, NBC) were in effect interpolated with the text source (NA News), with the latter being give a weight of 0.1.

3.2 Retrieval System

For the SDR track, we use the NA News corpus (also used in the language model training described above) as the large collection for conservative collection enrichment (see Section 2.2). Since the test data is dated from June 1997 to January 1998, we used news dated from May 1997 to February 1998 (one month before and after) from the NA news corpus.

Algorithm

We use the following algorithm in our reference run—**att-r1**, our two baseline runs—**att-b1** and **att-b2**, and our first full SDR run—**att-s1**. This algorithm is exactly the algorithm we have used in the ad-hoc track with some parameters changed to deal with the small size of the speech database.

- **Pass-1**: Using *dtn* queries and *dnb* documents, a first-pass retrieval is done.

- *Expansion:* Top five documents retrieved in the first pass are *assumed* to be relevant to the query and documents ranked 101–200 are assumed non-relevant. The query is expanded by adding ten new words and two new phrases using Rocchio's formulation (parameters $\alpha = 2,\ \beta = 1,\ \gamma = 1$). To include the *idf*-factor in the expansion process, documents are *dtb* weighted.

- The above expansion step is performed once on the target collection and once on the large NA news collection. Conservative collection enrichment is done as described in Section 2.2.

- **Pass-2:** The expanded query is used with *dnb* documents to generate the final ranking of 1,000 documents.

One of the main motivations for using the above algorithm is to keep our ad-hoc algorithm uniform across tasks.

Lattice Based Document Expansion

The one-best transcript from a recognizer misses many content words and adds some spurious words to the spoken document. The misses reduce the *word-recall* (proportion of spoken words that are recognized) and the spurious words reduce the *word-precision* (proportion of recognized words that were spoken). We believe that information retrieval algorithms would benefit from a higher word recall and are robust against poor word precision. An approach to enhance word recall is to add new words that "could have been there" (words that were probably spoken but weren't the top choice of a speech recognizer) to the automatic transcriptions of a spoken document.

Several techniques are plausible for bringing new words into a document. An obvious one from an IR perspective is *document expansion* using similar documents: find some documents related to a given document, and add new words from the related documents to the document at hand. And from a speech recognition perspective, the obvious choice is to use word lattices which contain multiple recognition hypotheses for any utterance. A word lattice contains words that are acoustically similar to the recognized words could have been said instead of the words recognized in the one-best transcription.

We use both these techniques to do controlled document expansion for our second full SDR run **att-s2**. In our experiments we found that each method when used alone adds more spurious words to a document than is desirable. However, a controlled document expansion that incorporated information from both the sources helps in reducing the spurious words, allowing the good words to still be added to a document.

We take the one-best recognition for a story and look for similar stories in the print media (NA news). This is done by simply running the one-best recognition for the story as a raw-$tf \times idf$ weighted query on the NA news database. The idea being that important news would also be reported in the print media, and we can leverage words from there to enrich our spoken documents. We do not enforce any conditions like 'the returned stories from NA news should be from the same day or near the same date as the spoken document', even though one can imagine that this could possibly help.

We found that for speech stories that are not reported in the print media, marginally related stories are retrieved in response to the query (speech story), and unrelated words are brought into the story. To contain this problem, we force our expansion algorithm to choose only those words that are also present in the word lattice generated by our recognizer for the speech story. This restriction guarantees that the words being added to a document are also proposed by the speech recognizer, albeit with a low confidence.

The parameters for document expansion were chosen somewhat arbitrarily based on a quick inspection of the expansion terms and our experience with relevance feedback. We didn't have any testbed to tune our parameters. The following steps are used:

1. Twenty documents are retrieved from the NA news corpus for a given speech document. The one-best transcription for a speech document weighted using raw-$tf \times idf$ is used as a query.
2. 25% of the unique words, at most 50, new words are added to the speech document using Rocchio's formula. I.e., if the original one-best recognition has 80 unique (not counting repetitions) words, then 20 new words are added; if it has 200, then 50 new words are added; and if it has 300, then also only 50 new words are added. Following are some details of the expansion process:
   - Rocchio parameters $\alpha = 4,\ \beta = 1,\ \gamma = 0$ are used.
   - Only words that occurred in at least 10 of the 20 documents retrieved in step 1 are used as expansion terms.
   - Both the original speech document and the top documents from step 1 are *dtb* weighted (see Table 1) for Rocchio's formula. This yields expanded document vectors that have *idf* in the weights.

Transcription	Used in Run	WER	Term Recall	Term Precision
Reference	att-r1	0	100	100
Baseline-1	att-b1	34.1%	78.98%	78.02%
Baseline-2	att-b2	46.9%	68.40%	68.04%
Full SDR-1	att-s1	32.4%	81.79%	81.58%
Expanded Docs	att-s2	—	83.74%	67.50%

Table 4: Analysis of recognition and document expansion

Retrieval Condition	Baseline *dnb.dtn*	Expansion from		Collection Enrichment	Best	Above Median	Below Median
		target collection	NA News				
Reference **att-r1**	0.4548 —	0.5083 +11.7%	0.4864 +6.9%	**0.4992** +9.8%	5	16	2
Baseline-1 **att-b1**	0.4115 —	0.4925 +19.7%	0.4493 +9.2%	**0.4700** +14.2%	9	10	4
Baseline-2 **att-b2**	0.3358 —	0.3941 +17.4%	0.3983 +18.6%	**0.4065** +21.1%	6	14	3
Full SDR-1 **att-s1**	0.4371 —	0.5069 +16.0%	0.4839 +10.7%	**0.5065** +15.9%	4	16	3
Full SDR-2 **att-s2**	0.4535 —	**0.5300** +16.0%	0.4981 +10.7%	**0.5120** +15.9%	7	13	3

Table 5: Results for SDR track

3. *Idf* is removed from the expanded document vectors by dividing by component words' *idf*s. Now we have expanded document vectors that are *dnb* weighted. These expanded documents are used instead of the one-best transcriptions in the same retrieval algorithm as used in the run att-s1, and this run was submitted as our second SDR track run att-s2.

Results and Analysis

The recognition word error-rate, average term recall (proportion of spoken words that are recognized), and average term precision (proportion of recognized words that are spoken) for various automatic transcriptions are shown in Table 4. To compute the word recall and the word precision, we take all the stories that have more than ten index terms (non-stop word-stems as indexed by SMART, multiple occurrences of a stem counted as one term) in the human transcription and compute how many of the original index-terms (non-stop word-stems only, repetitions not counted) are recognized in the automatic transcription. We also count the number of spurious terms recognized by a recognizer. We compute per-document term recall and term precision, and average these values across documents to get the numbers reported in Table 4. We omit all storied that have less than ten terms since the recall/precision figures for them are quite unstable. The main reason behind discounting multiple occurrence of a term is that the *tf*-factor of our weighting scheme has a similar effect. Recognizing a word once is more important than correctly recognizing multiple occurrences of the same word.

Table 4 shows that the word error rate for our recognizer is 32.4%, slightly better than the 34.6% for the medium error transcriptions provided by NIST. This is also reflected in the higher term recall and precision for our recognizer. Document expansion is doing its job, though not as well as we would like it to. It does increase the term recall by another 2% on an average but it significantly reduces term precision from 81.58% to 67.5%. But as discussed in the following results, the deterioration in term precision does not hurt retrieval effectiveness. This supports our hypothesis that term recall is more important for our retrieval systems.

The results for various runs are shown in Table 5. The official numbers are presented in bold in column 5, along with various other numbers. The baseline retrieval results—*dnb* documents, *dtn* queries, no query expansion—are shown in column 2. The average precision on human transcription is 0.4548. The retrieval effectiveness falls when retrieval is done on a speech recognizer's automatic transcription of the speech. The

Code	Provided By	WER
Human	NIST	0%
CUHTK-S1	Cambridge University	24.8%
Dragon98-S1	Dragon Systems	29.8%
ATT-S1	AT&T Labs	31.0%
NIST-B1	Carnegie Melon (CMU)	34.1%
SHEF-S1	Sheffield University	36.8%
NIST-B2	Carnegie Mellon (CMU)	46.9%
DERASRU-S2	DERA	61.5%
DERASRU-S1	DERA	66.2%

Table 6: Different automatic transcriptions.

base effectiveness for the medium-error baseline transcription (B1) is 0.4115, a loss of 9.5%. As expected, the average precision falls further to 0.3358 for the high-error recognition (B2).

When retrieval is done on our own recognition, which has a marginally higher term recall/precision, the retrieval effectiveness jumps from 0.4115 for B1 to 0.4371, a gain of 6.2%, and we are running just 3.9% behind retrieval on human transcriptions. Document expansion removes even this difference and retrieval from expanded documents is at par with retrieval from human transcriptions. This is quite encouraging, especially when the expansion parameters were chosen without any guidance. This also shows that term recall is indeed more important than term precision. The term precision for the expanded documents is noticeably worse than our one-best recognition, and the term-recall is marginally better; however, the retrieval effectiveness is better despite the poor term precision.

When query expansion from the target collection is done, all results improve noticeably (column 3 in Table 5). Retrieval from our transcriptions is still better than retrieval from the medium-error baseline transcription, and is at par with retrieval from human transcriptions. Retrieval effectiveness on expanded documents gets to 0.53 average precision and it actually surpasses the retrieval from human transcriptions (0.5083) by 4.3%. This results is very encouraging. Our conservative collection enrichment hurt us in this environment and our official runs (column 5 in Table 5) are all lower than if we had expanded just using the target collection. However, the official runs are still very competitive, both our full-SDR runs att-s1 and att-s2 are above median for 20 out of 23 queries. The run using expanded documents att-s2 yields 7 best results in 23 which is quite a substantial proportion.

All these results point us to the possible advantages of doing document expansion in speech retrieval environments. These results also tend to confirm our belief that term recall plays a more important role in retrieval effectiveness for speech than term precision. We can afford to lose term precision for better term recall and this should yield improved retrieval effectiveness for speech. A larger query set would have made these results much more robust, but there is a clear pattern in Table 5 indicating that document expansion consistently yields better results than not doing it.

3.3 Cross-Recognizer Analysis

After the official conference, we did a more rigorous study of document expansion. We first discovered that constraining document expansion to allow only terms from the word-lattices generated by our recognizer held no additional benefit over not doing so. *I.e.* we can do document expansion only from NA news and the results were equally good or better. This also allows us to test document expansion for retrieval from the automatic transcriptions provided by other SDR track participants, for which we don't have the word-lattices. Secondly we found that the query expansion parameters used in our SDR runs were sub-optimal. Using the standard set of parameters used in our ad-hoc run yields consistently better results, and has the added advantage of uniformity of parameters.

We test document expansion on different automatic transcriptions provided to NIST by various track participants. Table 6 lists these transcriptions along with their word error rates. We cleaned some timing mistakes in our transcripts and used the correct WER scripts to get the 31% WER reported in Table 6 (as opposed to the 32.4% WER reported in Table 4). Here are the steps involved in document expansion:

Transcript	Unexpanded Documents				Expanded Documents			
	Baseline	Query expn. from		Coll.	Baseline	Query expn. from		Coll.
	dnb.dtn	target	NA News	Enrich.	*dnb.dtn*	target	NA News	Enrich.
ltt	0.4595	0.5300	0.5211	0.5327	0.5108	0.5549	0.5334	0.5614
cuhtk-s1	0.4376	0.5035	0.5202	0.5285	0.5220	0.5372	0.5444	0.5549
dragon98-s1	0.4190	0.5100	0.5227	0.5147	0.5061	0.5284	0.5459	0.5483
att-s1	0.4353	0.5020	0.5128	0.5251	0.5080	0.5343	0.5505	0.5452
nist-b1	0.4104	0.4820	0.4987	0.4954	0.4862	0.5259	0.5314	0.5316
shef-s1	0.4073	0.4890	0.5042	0.5085	0.5068	0.5421	0.5355	0.5399
nist-b2	0.3352	0.3965	0.4602	0.4220	0.4377	0.4743	0.4961	0.4940
derasru-s2	0.3633	0.3962	0.4614	0.4419	0.4585	0.5065	0.5118	0.5199
derasru-s1	0.3236	0.3613	0.4604	0.4188	0.4526	0.4849	0.5045	0.4959

Table 7: Cross-recognizer analysis.

1. Find documents related to a speech document. We do this by running the automatic transcription of the speech document as a query (raw-$tf \times idf$ weighted) on the NA News corpus and retrieving the *ten* most similar documents. In other words, we use the ten nearest neighbors of the speech document in this process. The documents are weighted by raw-$tf \times idf$ when used as a query because we found that nearest neighbors found using raw-$tf \times idf$ weighted documents yield the best expansion results.

2. The speech transcriptions are then modified using Rocchio's formula.

$$\vec{D}_{new} = \vec{D}_{old} + \frac{\sum_{i=1}^{10} \vec{D}_i}{10}$$

   where $\vec{D}_{old}$ is the initial document vector, $\vec{D}_i$ the the vector for the i-th related document, and $\vec{D}_{new}$ is the modified document vector. All documents are *dnb* weighted (see Table 1). New words are added to the document. For term selection, the Rocchio weights for new words are multiplied by their *idf*, the terms are selected, and the *idf* is stripped from a selected term's final weight. Furthermore, to ensure that this document expansion process doesn't change the effective length of the document vectors, and change the results due to document length normalization effects, we force the total weight for all terms in the new vector to be the same as the total weight of all terms in the initial document vector. We expand documents by 100% of their original length (*i.e.* if the original document has 60 indexed terms, then we add 60 new terms to the document).

The results for unexpanded as well as the expanded documents are listed in Table 7. The two main highlights of these results are:

- document expansion yields large improvements across the board, and
- document expansion reduces the performance gap between retrieval from perfect and automatic transcriptions.

These points are highlighted in Figure 1. The left plot shows the average precision on the y-axis, against the WER on the x-axis. All number plotted in Figure 1 are for the unexpanded queries (*i.e.* we use the columns marked *Baseline* in Table 7). This prevents effects of query expansion from affecting these graphs and allows us to study the effects of document expansion in isolation. The horizontal lines are for human transcriptions whereas the other lines are for the different automatic transcriptions. As we can see in the left graph, document expansion (solid lines) yields large improvements across the board for this task over not doing document expansion (dashed lines).

The right graph in Figure 1 plots the %-loss from human transcriptions on the y-axis for unexpanded and expanded documents. The baseline for the expanded documents is the expanded human transcriptions, *i.e.* the solid horizontal line on the left graph. We observe that for the poorest transcriptions (DERASRU-S1) document expansion yields an improvement of an impressive 40% (over 0.3236) and reduces the performance

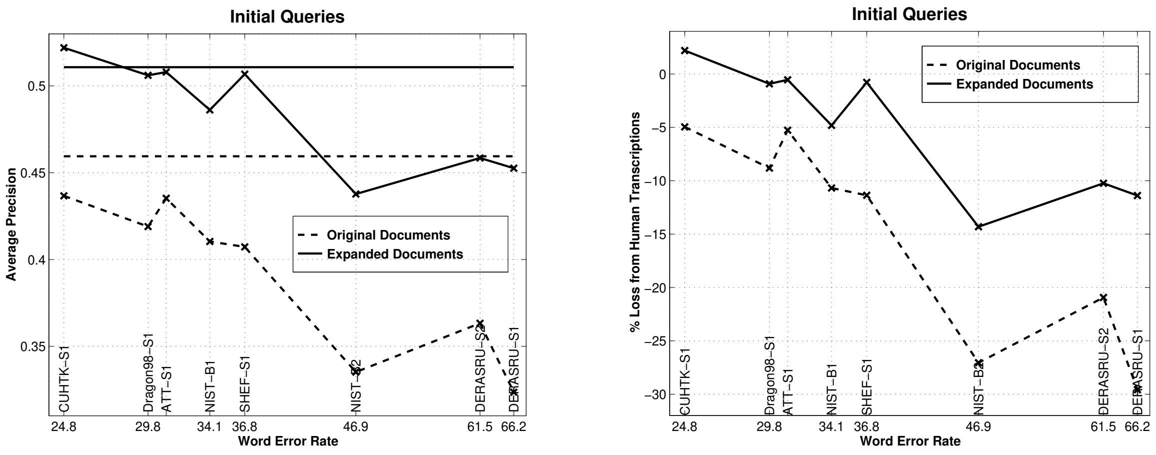

Figure 1: Raw average precision and %-loss from human transcriptions (initial user queries).

gap from human transcription to about 12% instead of the original 30% despite the very high baseline used. The results are similar for other transcriptions.

These results indicate that document expansion is indeed a very useful tool for retrieval from this speech corpus. We should caution that this is a very small test collection and more experimentation is needed, possibly on a larger test collection, before the full effect of document expansion could be analyzed in details.

4 VLC Runs

We come to the VLC track with the belief that with time the collection sizes will outgrow our capability to efficiently maintain a single index for a collection. We already see this happening on the Web. A recently published study reports that any single Web search engine covers less than 35% of the Web. However, if the collections from various engines are somehow pooled, the coverage improves dramatically. [6] For this reason, meta-search systems, systems that search multiple search engines and optionally merge the results are becoming increasingly available on the Web.

Our VLC track participation is modeled like a meta-search system. We divide the 100G collection into twenty independent collections of about 5G each. Each query is submitted in parallel to each of the twenty collections and their retrieval results are processed in various possible ways to obtain a single final ranking for the whole collection. We assume that each small collection is available through a *server*, and a user submits queries on a *client* which passes them to the servers, gathers the results and merges them into a single ranked list for presentation. We further assume that the client has access to some text collection for computation of *idf* values needed in our merging process described below. To make things closer to reality, we force the client to use an entirely different collection (TREC disks 1–5) as its *idf* collection since many clients will not have any Web collection (the target collection for the task) to gather *idf*s.

Four runs were submitted, each with a different set of assumptions to study different effects. Here is a brief description of the runs. Each server returns its top 20 documents to the client and the client generates a single ranked list for the 400 documents (20 each from 20 servers).

- **att98vi**: This run assumes the following:
  1. Each server is running a straight vector match *dnb* documents and *dtn* queries (see Table 1). No pseudo-feedback or query expansion is done at the servers' end.
  2. Each server returns the first 500 bytes (mark-up removed) for each of the top 20 documents to the client. This simulates the Web search environment where it is commonplace for engines to return the initial portion of the documents as a summary on the results page.

3. The client indexes the 400 summaries "on-the-fly" and creates *dnb* documents. The client also indexes the query using *dtn* weighting (*idf* is picked from the TREC collection).
4. The summaries are ranked by the client using a straight vector match, and the rank of a summary defines the rank of the final document. No pseudo-feedback or query expansion is done at the client's end.

Result: Average P@20 0.3570

- **att98vf**: This run is the same as **att98vi** except that servers return full document text instead of the initial 500 bytes and full document text is used by the client to produce the final ranking.

  Result: Average P@20 0.5030

- **att98vie**: This run has the following assumptions:

  1. Each server expands the query using pseudo-feedback.
  2. The client also expands the query in parallel using pseudo-feedback on its local collection (TREC).
  3. Each server returns the first 500 bytes (summary) for documents retrieved using the expanded query from step 1.
  4. The client indexed the 400 summaries "on-the-fly".
  5. The summaries are ranked by the client using the expanded query from step 2.

  Result: Average P@20 0.3750

- **att98vfe**: This run is the same as **att98vie** except that the servers return full document text (instead of the initial 500 bytes) which is used by the client to produce the final ranking.

  Result: Average P@20 0.5870

We wanted to show that merging based on the initial 500 bytes is not much worse from merging based on the full text of the documents. This indeed was the result in our internal studies in which we split the TREC database into several servers and evaluated our merging methods with TREC queries. However, we are disappointed to see that our run att98vi is noticeably worse than the corresponding full-text run att98vf. Similarly att98vie is noticeably worse than att98vfe. We would like to investigate this discrepancy in behavior for the TREC documents and the Web documents. On first thought, one is inclined to believe that Web documents are just different than more traditional TREC documents, and the initial part of a Web document is not a very good representative of the entire document (whereas for the TREC documents it is), but we would like to study this effect further.

5 Filtering Runs

We used the TREC-7 filtering track to test an alpha version of ATTICS, our toolkit for machine learning of classifiers for mixed textual and non-textual information. ATTICS differs from most text retrieval software in its support for numeric and other formatted data, and in its emphasis on classification of streams of data rather than retrieval from a static or slowly changing collection. It differs from most machine learning software in its efficient and flexible support for textual data, particularly in mapping from structured documents to attribute vectors.

ATTICS can be used either as a stand-alone system or as a C++ library that can be embedded in other applications. An extensive API and careful use of the C++ class system enable new preprocessors, data transformations, classifier types, training methods, and output formats to easily be added to the system. We used ATTICS in stand-alone mode for the TREC-7 filtering task. Support in ATTICS for incremental training of classifiers is not yet complete, so only the batch filtering and routing tasks were attempted. This also meant that in the batch filtering task we did not take advantage of training on retrieved test documents.

ATTICS uses XML internally for document mark-up. Since the SGML mark-up for TREC data obeys XML conventions, processing of TREC data was trivial. For the quick experiment reported here we mapped the HEAD and TEXT fields (lower-cased) of the AP documents to a single vector of raw *tf* counts. Stemming and stop-wording have not yet been implemented and so were not used.

Early runs on TREC and other large data sets showed preprocessing to be unacceptably slow. The problem was traced to the extensive use of C++ streams for communication between preprocessor modules, a holdover from a prototype in the Unix pipeline style. A reimplementation achieved a rate of 220MB/hr for creation of on-disk vectors from on-disk text using one R10000 processor of an SGI Challenge XL with 8GB of RAM. This is still well below the comparable rate (about 3GB/hr) for our latest modification of SMART, but ATTICS can process a much richer set of data types, and the version used was not yet fully optimized.

ATTICS development to date has focused almost exclusively on basic systems issues, data modeling, API design, and so on, with little time left for implementing particular learning algorithms. To meet the TREC deadline, we did a quick implementation in ATTICS of the original Rocchio algorithm. [8] We trained linear models for each of the 50 TREC-7 filtering topics using only the judged AP88 documents, with Rocchio parameters $\alpha = 0$ (topic descriptions were not used), $\beta = 1$, and $\gamma = 1$. Negative Rocchio weights were zeroed out, as usual. Within document weights for all training and test data were computed using SMART *Lnu* style normalization with a slope parameter of 0.2. [12]

This first set of linear models was used to retrieve the top 5000 scoring AP88 documents. Then a second set of linear models was trained using the union of the judged AP88 documents and the top 5000 AP88 documents, with unjudged documents in the top 5000 being treated as non-relevant, *i.e.* a query zoning approach. [13] The second set of linear models were run back over the training documents to score them, and a threshold for each model was chosen that optimized the desired effectiveness measure (filtering measure F1 or F3). The models and thresholds were then applied to the test (AP89-90) documents, with the documents exceeding the threshold constituting the submitted set for batch filtering.

Our batch filtering submissions showed reasonable effectiveness. Run att98fb5, optimizing measure F1, was at or above median utility for 36 of 50 topics, with 16 tied for best. Run att98fb6, optimizing measure F3, had 32 of 50 topics at or above median utility, with 11 tied for best. Statistical significance tests showed the effectiveness of these runs to not be significantly different from that of the best submitted batch filtering runs [3]. This was a bit of a surprise to us, given our omission of feature selection, stop lists, *idf* weighting, dynamic feedback optimization, and other modern enhancements to Rocchio-style training.

On the other hand, our routing run using the scores output by the same Rocchio models (att98fr4) was not state-of-art. Overall average precision was 0.275, and per-topic average precision was above median for only 23 of 50 topics, with 2 bests. For comparison, we submitted a routing run (att98fr5) based on a modern augmented Rocchio approach implemented in SMART. The algorithm was a scaled-down version of our TREC-6 run att97rc, and did not use word cooccurrence pairs as features. [11] Overall average precision for this run was 0.419, with per-topic average precision at or above median for 41 of 50 topics, with 5 best. Incorporating more modern training methods into ATTICS is a priority for us, and should improve both filtering and routing effectiveness.

6 Conclusions

We are encouraged by our SDR performance, especially by the possible advantages of document expansion in this environment. We are quite satisfied by our filtering performance given the initial developmental stage of the software we are using. We would like to further study meta-searching as a model for searching very large collections. We simplified our adhoc algorithm over the complex algorithm we used last year and the results are still very good.

Acknowledgments

We are thankful to Anna Litvinova, Mandar Mitra, Marcin Kaszkiel, Yoram Singer, and Dan Stern for their help with various aspects of this work. We are also very grateful to Andrej Ljolje, Mehryar Mohri, and Michael Riley for their help in building the recognizer for the SDR track data.

References

[1] Chris Buckley. Implementation of the SMART information retrieval system. Technical Report TR85-686, Department of Computer Science, Cornell University, Ithaca, NY 14853, May 1985.

[2] Julia Hirschberg and Christine Nakatani. Using machine learning to identify intonational segments. In *Proceedings of the AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, Palo Alto, CA, March 1998.

[3] David A. Hull. The TREC-7 filtering track: Description and analysis. In this volume.

[4] S.M. Katz. Estimation of probabilities from sparse data from the language model component of a speech recognizer. *IEEE Transactions of Acoustics, Speech and Signal Processing*, pages 400–401, 1987.

[5] K.L. Kwok. Improving two-stage ad-hoc retrieval for short queries. In *Proceedings of the Twenty First Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 250–256. Association for Computing Machinery, New York, August 1998.

[6] Steve Lawrence and C. Lee Giles. Searching the World Wide Web. *Science*, 280(5360):98, 1998.

[7] Fernando C. N. Pereira and Michael D. Riley. Speech recognition by composition of weighted finite automata. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*, pages 431–453. MIT Press, Cambridge, Massachusetts, 1997.

[8] J.J. Rocchio. Relevance feedback in information retrieval. In *The SMART Retrieval System—Experiments in Automatic Document Processing*, pages 313–323, Englewood Cliffs, NJ, 1971. Prentice Hall, Inc.

[9] Gerard Salton, editor. *The SMART Retrieval System—Experiments in Automatic Document Retrieval.* Prentice Hall Inc., Englewood Cliffs, NJ, 1971.

[10] Kristie Seymore and Ronald Rosenfeld. Scalable backoff language models. In *ICSLP'96*, volume 1, 1996.

[11] Amit Singhal. AT&T at TREC-6. In E. M. Voorhees and D. K. Harman, editors, *Proceedings of the Sixth Text REtrieval Conference (TREC-6)*, pages 215–226, 1998.

[12] Amit Singhal, Chris Buckley, and Mandar Mitra. Pivoted document length normalization. In *Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 21–29. Association for Computing Machinery, New York, August 1996.

[13] Amit Singhal, Mandar Mitra, and Chris Buckley. Learning routing queries in a query zone. In *Proceedings of the Twentieth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 25–32. Association for Computing Machinery, New York, July 1997.