Verity at TREC-6: Out-of-the-Box and Beyond

Jan O. Pedersen, Craig Silverstein, Christopher C. Vogt
Verity, Inc.
894 Ross Dr.
Sunnyvale, CA 94089
jpederse@verity.com, csilvers@cs.stanford.edu, vogt@cs.ucsd.edu

February 17, 1998

Abstract

The Verity Trec-6 entry focused on the performance of the built-in search facilities of the commercially available Verity engine and explored the impact of simple enhancements. The *ad hoc* results show that considerable improvements can be achieved through the application of standard and more experimental techniques. The *routing* results show that respectable performance can be achieved simply through careful parameter tuning.

1 Introduction

The focus of Verity's TREC-6 work was to measure the performance of the built-in Verity retrospective search facilities and to explore simple extensions which have the potential to improve performance. Participating in both the *ad hoc* (short query, automatic only) and *routing* Category A tracks, we investigated both standard techniques (e.g., IDF, length normalization, pseudo-feedback, and training routing classifiers on a per-query basis) along with more experimental ones (linguistic parsing, clustering, local region training, and source-sensitive training). None of these are yet built into the Verity engine, although they can be implemented on top of the Verity Query Language. As our routing experiments show, the built-in Verity QBE facility can perform on a par with other TREC entries. Our ad hoc experiments show, as expected, that adding the "standard" techniques boosts performance significantly, whereas most of the other techniques also boost performance, but to a lesser degree.

Our *ad hoc* approach began with a very simple baseline system and progressively added techniques, all built on top of the Verity Query Language (VQL). Each new technique except clustering successfully improved performance. Our *routing* approach emphasized using queries automatically generated by the built-in VQL Query-By-Example facility, tuning the parameters to that facility based on training examples. Thus, our *routing* approach represents what a current Verity user could achieve without adding to built-in functionality, but our *ad hoc* approach represents what VQL is capable of achieving.

All experiments were performed on a Sparc Ultra Enterprise 3000 with four 250 MHz UltraSparc processors and 2 GB of main memory, running Solaris 2.5.1. The experiments were run using release 2.2 of Verity's search engine.

2 The Ad Hoc Task

The Verity search engine supports a rich, fuzzy Boolean query language. However, it is intended for use with hand-crafted queries, not TREC-style natural language queries. This is not a functional limitation; one can supply a query analyzer that converts natural language queries to fuzzy Boolean. However, the built-in Verity query analyzer is missing some components typically used in tandem with natural language queries. In particular, it employs a non-linear scoring scheme optimized for short queries that saturates (achieves a maximum score) too rapidly to be useful for Trec-length queries.

Nevertheless, the underlying Verity query language supports features that allow a state-of-the-art natural language query analyzer to be built on top of the Verity engine. Query words may be weighted arbitrarily, allowing IDF (inverse document frequency) information to be encoded in the query. The weighting scheme may be changed to a linear vector model, ameliorating saturation problems. Summarization facilities provide a novel method for length normalization and also a basis for performing pseudo-feedback. Finally, we can use the ability to cluster result sets to provide a refinement of the typical pseudo-feedback algorithm.

For the *ad hoc* TREC-6 task, we started with the baseline query analyzer that converted each natural language query into a sum of query terms. Then we added each of the following units, in turn, on top of the base analyzer:

1. Phrase identification and part-of-speech-based word elimination;
2. Word elimination based on shallow syntactic analysis;
3. IDF weighting;
4. Length normalization;
5. Pseudo-feedback; and
6. Clustering.

Notice that each of these steps involves merely modifying the query, or, in the case of length normalization, modifying the corpus. Thus they can all be implemented without having access to the internals of the underlying search engine.

The results of these modifications are summarized in Table 1, which looks at the precision at 30 documents for both TREC-6 and TREC-5. Since the algorithms build on each other, we show both the improvement in performance over the previous algorithm as well as over the base algorithm described in Section 2.3. In the following sections, we study each refinement in turn.

2.1 Experimental Setup

The TREC-6 *ad hoc* test bed consists of about 2.5 gigabytes of data, in about 550,000 documents, from the Congressional Record, Financial Register, Financial Times, Foreign Broadcast Information Service, and LA Times collections. This is augmented with 50 topics along with relevance judgements. Each topic consists of three fields: a title (2-3 words), a description (1-2 sentences), and a narrative (a paragraph listing specific criteria for accepting or rejecting a document). We examined only the description of each topic, ignoring the other fields.

For each algorithm studied, we ran the algorithm on each query. The algorithm attached a score to each document based on the query and retrieved the top 1000 documents, sorted by score. We studied two major statistics based on the retrieved document set for each query: the total recall

trec6	prec30	Δprev	Δbase
base	0.0700	—	—
+phrase	0.0987	40%	40%
+syntax	0.1087	10%	55%
+idf	0.1727	58%	146%
+len	0.2207	27%	215%
+feedback	0.2473	12%	253%
+cluster	0.2333	−5%	233%
out-of-box	0.0860	—	22%
base+idf	0.1247	—	78%

trec5	prec30	Δprev	Δbase
base	0.0673	—	—
+phrase	0.1327	97%	97%
+syntax	0.1447	9%	115%
+idf	0.2027	40%	201%
+len	0.2113	4%	213%
+feedback	0.2493	17%	270%
+cluster	0.2513	0%	273%
out-of-box	0.0800	—	18%
base+idf	0.0913	—	35%

Table 1: The performance of various refinements to the baseline Verity search engine. Performance is given in terms of the precision at 30 documents, averaged over all 50 queries in TREC-6 (left table) and TREC-5 (right table). Relative performance is in terms of both the previous algorithm, which the current algorithm builds on, and the original, baseline algorithm. Algorithms below the line are not in the main chain of refinements and are used to examine the effectiveness of certain refinements in isolation, for instance IDF without term selection.

(that is, how many of the 1000 retrieved documents were judged relevant), and the precision at 30 documents (that is, how many of the top 30 retrieved documents were judged relevant).

2.2 The Out-of-the-Box Algorithm

The Verity `<FREETEXT>` operator is intended to handle short natural language queries. This built-in, out-of-the-box algorithm performs poorly over the relatively long TREC-6 queries, as Table 1 attests. In the table, the out-of-the-box algorithm yields only a small improvement over the baseline algorithm. It also performs significantly worse than the "phrase" algorithm, which performs similar term selection. This disparity can be attributed to the scoring function, which, as we mentioned in Section 2, saturates with many query terms. The out-of-the-box algorithm uses this default scheme, while the baseline algorithm uses an alternate scheme that does not have the saturation problem. We conclude that even with relatively aggressive term selection, many documents contain a large number of query terms for most queries, causing score saturation to be an issue.

2.3 The Baseline Algorithm

While the freetext parser is helpful for natural language queries, it is not truly a baseline, at least in the context of the Verity search engine, because it performs syntactic parsing of the query before invoking the search routines. We therefore, instead, build our algorithms on top of the following, simple, baseline algorithm:

1. Considering every word (including stop words) as a separate term in the query, weight each word equally so the weights sum to 1.

2. Combine the query terms using the `<SUM>` operator. Pass this query to the Verity search engine.

The `<SUM>` operator determines a document's score by summing the scores of each query term on that document. The score of a term on a document is determined as follows: A term has score 0 if it does not occur in the document, and a score equal to its weight if it occurs (logically) an infinite number of times in the document. The score is in between if it occurs a finite number of

times in the document, but it quickly grows to the weight. The Verity engine performs stemming on each query term. It is case sensitive only if the query term includes an upper-case letter. (In our experiments, we keep the case of the original TREC-6 description, except we lower-case the first word of each sentence.)

This equal weighting scheme is clearly a bad idea without careful term selection, and indeed the baseline algorithm performs quite badly. Its performance can be seen as the bottom curve in Figure 1 and the first row in Table 1.

2.4 Term Selection I: Phrase and Part-of-speech

One obvious way to improve the baseline algorithm is to perform term selection. In this step we perform a similar analysis as is done by the freetext parser. However, instead of the built-in parser we use ENGCG, an off-the-shelf parser sold by the Lingsoft Corporation (http://www.lingsoft.fi) We use ENGCG because the built-in parser cannot perform the syntactic analysis we do in the next section, and to use the ENGCG parser there we must use it here as well. This parser performs the following functions:

1. Delete words due to their part of speech. Nouns, verbs, adjectives, adverbs, interjections, numerals, abbreviations, and participles are retained. Coordinating conjunctions, subordinating conjunctions, determiners, infinitive markers, prepositions, and pronouns are removed.

2. Collect words into phrases, when appropriate. There are two types of phrases: noun phrases (when one or more nouns modify a noun) and adjective phrases (when one or more adjectives modify a noun or noun phrase).

3. Weight each term or phrase so the weights sum to one. Terms are weighted equally, but adjective phrases are given twice the weight, and noun phrases three times the weight, of terms.

4. Combine the query terms using the <SUM> operator. Pass this query to the Verity search engine.

Identification of phrases is possible because ENGCG is a shallow syntactic parser, so it can determine which word a modifier modifies. If A modifies B, everything between A and B is taken to be a phrase, and the type of phrase is determined by the part of speech of A. Thus, "Food and Drug Administration" is a noun phrase.

Phrases are weighted highly purely due to experimental evidence that it improves recall. This is expected, since previous research has shown that phrases are better markers of relevance than individual terms.

Recall that individual terms are scored based on their frequency in a document. Phrases are scored somewhat differently. A phrase gets full credit if the phrase appears as consecutive words in the document. However, it can get up to half credit if the words in the phrase appear in the document, but not consecutively. Each phrase component found contributes equally to the score. Furthermore, if the words in the phrase appear close to each other but not consecutively, up to 75% credit is possible. This accommodates situations where the phrase is interrupted by a modifier or interjection. Experimental results show that this complicated scoring system improves on the naive scoring system based on considering the phrase merely as a single word with internal spaces.

Term and phrase selection improve the performance dramatically, as can be seen in Figure 1 and the second row in Table 1.

2.5 Term selection II: Shallow Syntactic Parsing

The ENGCG parser is a shallow linguistic parser, which means it can identify phrases, clauses, and the role words play in a sentence (for instance, whether a noun is the subject, whether a verb is transitive or intransitive, and so on). We use this to recognize patterns that are often found in natural language queries, and to delete uninformative terms in the query.

For example, we can detect if the verb is imperative ("Find all documents relating to slavery in Brazil"), in which case the object of the verb is merely a placeholder like "documents." In this case we delete the verb and its object. After part of speech elimination, we will be left with the query `<SUM>(slavery, Brazil)`.

As another example, if there are chained prepositional phrases starting with "of," only the last prepositional phrase is kept. Thus, in sentences like "The exportation of some part of U.S. industry," we would throw out "part." In every query we have examined, the first "of" prepositional phrase holds a contentless word.

This yields only a small addition to the algorithm:

1. Delete words according to their syntactic function.
2. Analyze the remaining words using the algorithm in Section 2.4.

As Figure 1 and the third row in Table 1 demonstrate, this more aggressive term selection improves the quality of the result set, though not dramatically. Since these rules are only applied to specific query forms, the effect of this step is pronounced in some queries but non-existent in most. See Tables 2 and 3 for a per-query analysis.

2.6 Term Weighting

There were two weaknesses of the baseline algorithm: lack of term selection and lack of term weighting. Previous algorithms have attacked the first of these problems; now we approach the second.

One very popular term weighting scheme is tf.idf weighting, that is, weighting by term and inverse document frequency. The Verity engine performs some variant of term weighting by default, so we need only to add IDF weighting.

The IDF weight of word w is defined to be

$$\log \frac{n}{n_w}$$

where n is the total number of documents and n_w is the number of documents containing word w. While the Verity search engine does not support IDF explicitly, it does allow us to query how many documents a term occurs in. Therefore, we can, in a preprocessing step, determine the IDF weight of every word (and phrase) in the query. This yields the following algorithm:

1. Perform term selection as in Section 2.5.
2. Weight each term/phrase so that the weights sum to 1. However, instead of equal weight, each element gets a weight related to to its IDF weight. In particular, terms are weighted proportional to their IDF score, adjective phrases proportional to twice their IDF score, and noun phrases proportional to three times their IDF score.
3. Combine the query terms using the `<SUM>` operator. Pass this query to the Verity search engine.

Since phrases tend to have high IDF weights due to their relative rarity, it may seem no longer necessary to give phrases a bonus of double or triple weight. However, experiments showed that recall was improved when these bonuses were applied.

IDF weighting is also applied inside phrases. Remember that a document could get a partial score for a phrase if its constituent words occurred, even if the phrase did not. The partial score for a given constituent word is made proportional to the IDF score of that word. Thus, in the phrase "Food and Drug Administration," a document would get little bounce from having "and" but a larger jump for having "Administration."

As is consistent with previous research, IDF improved the quality of the result set tremendously (see Figure 1 and the fourth row in Table 1).

IDF is often seen as a replacement of, or improvement on, term selection. Thus, we would expect IDF to give a greater bounce to the baseline algorithm than to the term selection algorithm. As we see in Table 1, however, this is not the case. The last row of the table shows the improvement over baseline due merely to IDF: 78% for TREC-6 and 35% for TREC-5. The "+idf" row shows the advantage of adding IDF to phrase selection: 58% and 40%, respectively. For each data set, the improvement is similar in both cases. This seems to indicate that IDF and term selection actually complement each other. IDF is useful for queries with many nouns and other words term selection algorithms tend to see as identical. On the other hand, IDF may highly weight words that are tangential to the query (such as "documents" in "Find documents that relate to the crime in Europe") that term selection techniques can identify. Thus, despite some overlap in functionality overlap, IDF and term selection are complementary techniques.

2.7 Length Normalization

Collections with both short and long documents, such as the data set used for TREC-6, provide a particular problem for search engines. A canonical problem document is the dictionary, which matches every query word but is relevant to few queries. Traditionally this has been handled via length normalization, which penalizes long documents. Approaches have ranged from automatically splitting long documents to downweighting documents based on their length.

For these experiments we tried a different approach. The Verity search engine includes a summarizer, which ranks each sentence of a document based on its perceived relevance to the document's content. We chose a length (approximately 16 Kbytes, the median length of a document in TREC-6), and used the summarizer to bring documents larger than the cutoff length down to that length in size. We did this by selecting sentences in order of relevance until the cutoff was reached. Here is the resulting algorithm:

1. Replace each document in the corpus by its summarized version. Documents under the cutoff length (16K) remain unchanged.

2. Perform the algorithm of Section 2.6 on the summarized corpus.

This technique behaves differently than other length normalization techniques. If the document logically consists of many parts, one of which is much smaller than the others, then the summarizer is likely to ignore the small part entirely. Thus the search algorithm will fail on queries that match the small section. On the other hand, if the document contains many "throw-away sentences" tangential to the main thrust of the document, the summarizer will do well by ignoring the irrelevant sentences.[1] In our experiments, the summarizer did as well as, and even slightly better than, length

[1]While this might generally be considered a feature, it is not rewarded behavior in the TREC framework, since TREC judges a document to be relevant if *any* portion in the document fits the relevance criteria (as spelled out in the narrative).

normalization based on Singhal's method.

The results for the algorithm with length normalization added can be found in Figure 1 and the fifth row in Table 1.

2.8 Pseudo-Feedback

Pseudo-feedback is a common "accelerator" technique, piggybacked on other techniques. If the original technique has "good" precision, pseudo-feedback improves it; if the precision is "bad," pseudo-feedback usually makes things worse. The reason for this behavior (and one reason "good" and "bad" are not well defined concepts) is that pseudo-feedback augments the query based on terms found in highly ranked documents. If these documents are actually relevant to the query, the new query may better specify the information need, but if they are random documents the new query will be overwhelmed with irrelevant terms.

We use the Verity summarizer feature for pseudo-feedback. Just as for length normalization, the summarizer identifies sentences it deems represent the document, here we have the summarizer identify words it finds relevant. We had the summarizer identify the 5 most representative terms and phrases from each of the 20 documents deemed most relevant (according to the algorithm of the previous section).[2] We then ranked the 100 terms/phrases based on their IDF score, strongly upweighting terms recommended by more than one document. We took the 5 highest scoring terms and used them to augment the query, and performed the algorithm from Section 2.7 on the new query. Formally:

1. Perform the algorithm of Section 2.7 on the query.
2. For the top 20 documents (or for all documents with score at least 0.3, whichever is fewer), use the summarizer to find the 5 most relevant words. Rank the 100 words based on tf.idf weighting. Collect the 5 highest ranking words.
3. Form a new query by adding the 5 words from the previous step to the end of the existing algorithm.
4. Perform the algorithm of Section 2.7 on the new query.

This is a two-step algorithm. Previous research used many more than 20 documents, and many more than 5 terms, for pseudo-feedback. However, these were the parameters that gave the best results in our tests. One reason may be that for many TREC queries, the number of relevant documents is small, so it is advantageous to use few documents for pseudo-feedback. Also, our method of augmenting the query favors using few feedback terms. If we use many terms, with short queries — and many queries are short after term selection — the feedback terms will outweigh the original query terms. This is especially true since the feedback terms are chosen based on their high IDF weight.

Figure 1, and the sixth row in Table 1, show the effect of pseudo-feedback on retrieval quality. A look at the individual queries shows that it helps significantly on some queries and hurts on others; the performance on any given query is highly dependent on the quality of the five terms chosen for feedback.

[2] We actually used fewer than 20 documents if most documents had low scores. Recall that each document gets a score from 0 to 1. We only used documents for feedback whose score was at least 0.3.

2.9 Cluster-based Pseudo-Feedback

One way to judge whether the original retrieval algorithm is "good" or "bad" on a query — that is, whether performing pseudo-feedback would improve or degrade the quality of the result set — is whether the top documents retrieved have similar content. If they do, this content is likely to be related to the query. If not, the search engine likely retrieved random documents. This reasoning is called the *Cluster Hypothesis*.

Verity provides a clustering routine that we can use to judge whether a set of documents has similar content. We cluster all 1000 retrieved documents into 5 clusters. We then count how many of the top 20 documents fall into the same cluster. If it is at least half, we judge the result set coherent and perform pseudo-feedback. If not we return the result set without performing feedback.

We actually use a slightly different technique that guarantees we always perform feedback. For every i between 1 and 20, we mark i if at least half of the i top documents are in the same cluster. We then take the largest i that is marked. Since $i = 1$ is always marked, we always take at least one document for pseudo-feedback.[3]

The results of this cluster-based algorithm are shown in Figure 1 and the last column above the line in Table 1.

2.10 TREC-5 and TREC-6 results

Here we show the results for the various algorithms, both on the TREC-5 collection and the TREC-6 collection. TREC-5 can be thought of as the "training" set, since the parameters of the various algorithms (such as the weighting bonus of phrases, the 0.3 score cutoff for pseudo-feedback, and the 16K length cutoff for length normalization) were optimized for TREC-5. We see, comparing the two tables in Table 1, that in TREC-6 most methods provide a smaller increase, percentage-wise, than in TREC-5. Likewise, the overall improvement over the baseline algorithm is smaller. This may indicate that the parameters were over-trained for TREC-5. It may also merely indicate that the topics in TREC-6 are harder than the topics in TREC-5.

As mentioned in Section 2.1, we are concerned mostly with recall at 1000 documents and precision at 30 documents. In reality we consider the latter statistic to be the single most important, since in our view users are likely to look at only the first two screens of search results for any query. These are the statistics we report in Table 2, for TREC-6, and Table 3, for TREC-5. We show the statistics on a per-query basis, to illustrate that many of these methods work only for queries of a certain structure, or those with many — or few — relevant documents. For comparison with other TREC researchers, we also show the averaged uninterpolated average precision for each algorithm.

In Figure 1 we show the recall-precision curves for TREC-6 and TREC-5.

[3] This is not technically true, since the 0.3 score cutoff still applies: we look at less than 20 documents if not all 20 have score above 0.3 after the original retrieval step.

		base		+phrase		+syntax		+idf		+len		+feedback		+cluster	
Q#	rel	prec30	rel	prec30	rel	prec30	rel	prec30	rel	prec30	rel	prec30	rel	prec30	rel
301	474	0.0333	60	0.0667	45	0.0333	44	0.1000	81	0.3333	94	0.4000	125	0.2333	77
302	77	0.2667	27	0.3333	39	0.3333	39	0.5667	68	0.6000	65	0.6000	71	0.6333	70
303	10	0.0333	9	0.1667	10	0.1667	10	0.1667	10	0.2000	10	0.1667	10	0.1333	10
304	226	0.0333	34	0.0667	63	0.0667	66	0.2333	86	0.4000	88	0.3667	94	0.2667	93
305	35	0.0000	3	0.0000	10	0.0000	10	0.0000	13	0.0000	9	0.0000	6	0.0000	10
306	352	0.2667	75	0.3000	139	0.3000	139	0.2333	122	0.4333	126	0.5333	163	0.4667	145
307	215	0.0000	24	0.0333	22	0.0333	22	0.0667	42	0.2667	50	0.3000	69	0.3000	69
308	4	0.0000	2	0.0667	3	0.0667	3	0.1000	4	0.1000	4	0.1000	4	0.1000	4
309	3	0.0000	1	0.0000	0	0.0000	0	0.0000	0	0.0000	0	0.0000	2	0.0000	2
310	13	0.0333	2	0.0333	3	0.0333	3	0.0667	6	0.0667	6	0.1333	6	0.1333	6
311	186	0.0000	6	0.0000	7	0.0000	9	0.0000	8	0.0333	10	0.0333	10	0.0333	9
312	11	0.0000	1	0.0000	1	0.0333	6	0.0000	5	0.0000	6	0.0000	7	0.0000	7
313	107	0.7333	68	0.7667	69	0.7667	69	0.8000	70	0.9000	59	0.9333	73	0.9333	73
314	45	0.0333	13	0.2333	11	0.2333	11	0.1333	34	0.1000	33	0.1333	27	0.1333	27
315	67	0.0000	0	0.0000	1	0.0000	1	0.0667	29	0.0667	31	0.0667	30	0.0667	29
316	35	0.0667	6	0.0000	4	0.0000	4	0.1000	12	0.1000	12	0.1000	15	0.1000	12
317	14	0.1000	8	0.0333	3	0.0333	3	0.3333	11	0.3333	11	0.3333	10	0.3333	10
318	128	0.0000	15	0.0000	15	0.0000	15	0.0000	21	0.1333	25	0.1000	22	0.0667	20
319	187	0.2000	54	0.1000	49	0.1000	49	0.2333	78	0.2000	68	0.3333	91	0.2667	85
320	6	0.0000	5	0.0000	6	0.0000	6	0.0333	6	0.1333	6	0.0333	6	0.0333	6
321	234	0.0667	29	0.1333	32	0.1333	40	0.1333	36	0.1667	20	0.2333	27	0.1000	21
322	34	0.0333	4	0.0000	7	0.0667	12	0.0333	19	0.0333	19	0.0333	16	0.0333	16
323	63	0.0333	9	0.3333	19	0.3333	19	0.1333	13	0.1000	12	0.1000	12	0.1000	12
324	162	0.2333	87	0.2333	89	0.2333	89	0.2333	102	0.2000	104	0.8333	125	0.8333	125
325	24	0.0667	11	0.0667	11	0.1333	13	0.2333	20	0.2333	20	0.2333	20	0.2333	20
326	48	0.0333	9	0.0333	1	0.0333	1	0.7333	44	0.7667	43	0.8000	46	0.8000	46
327	18	0.0000	6	0.0333	4	0.0333	6	0.0333	14	0.0667	12	0.1000	12	0.1000	12
328	9	0.0333	5	0.0667	7	0.1333	8	0.2333	8	0.2333	8	0.2667	8	0.2667	8
329	50	0.1333	15	0.1333	11	0.1333	11	0.1000	23	0.0667	21	0.0667	20	0.0667	20
330	60	0.0333	4	0.0000	15	0.0000	15	0.0000	7	0.0333	8	0.0333	19	0.0333	15
331	222	0.1667	24	0.3333	35	0.3333	35	0.5667	96	0.6000	119	0.6333	153	0.6333	146
332	278	0.1333	51	0.1667	56	0.1667	56	0.3000	89	0.4667	85	0.3667	87	0.3000	85
333	72	0.0667	18	0.0333	17	0.1333	21	0.2000	62	0.3333	61	0.4333	63	0.3333	62
334	18	0.0000	9	0.0333	11	0.1333	12	0.3667	18	0.4000	18	0.4000	18	0.4000	18
335	70	0.0333	13	0.1000	25	0.1667	26	0.3333	38	0.5333	38	0.6000	47	0.6000	47
336	12	0.0000	1	0.0000	0	0.0000	0	0.0000	3	0.0333	3	0.0333	3	0.0333	3
337	98	0.2000	16	0.3000	23	0.3000	23	0.3667	65	0.3667	63	0.3667	89	0.3667	89
338	5	0.0333	2	0.0333	2	0.0333	2	0.0333	2	0.0333	2	0.0333	5	0.0333	2
339	10	0.0333	3	0.0667	10	0.0667	10	0.1333	10	0.1333	10	0.1667	10	0.1667	10
340	81	0.1000	9	0.1000	22	0.1333	22	0.5333	38	0.5333	38	0.5667	38	0.5667	35
341	81	0.0333	13	0.1667	26	0.1667	26	0.2333	31	0.4000	34	0.4333	36	0.4333	38
342	23	0.0000	2	0.0000	2	0.0000	2	0.0000	4	0.0000	4	0.0000	5	0.0333	4
343	290	0.1000	13	0.1000	26	0.1000	40	0.1333	98	0.2667	107	0.4333	151	0.4333	155
344	5	0.0000	0	0.0000	1	0.0000	1	0.0000	4	0.0333	4	0.0333	4	0.0333	4
345	39	0.0333	3	0.0333	3	0.0333	3	0.0333	9	0.1667	14	0.1667	17	0.1667	17
346	106	0.0000	1	0.0000	2	0.0000	2	0.0000	7	0.0000	12	0.0000	13	0.0000	12
347	157	0.1000	14	0.1000	32	0.1000	32	0.1000	62	0.1333	78	0.0667	73	0.0667	73
348	5	0.0000	1	0.0333	2	0.0333	2	0.0333	4	0.0667	4	0.0333	4	0.0333	4
349	73	0.0000	5	0.0000	5	0.0000	6	0.0000	9	0.0333	13	0.0000	11	0.0000	12
350	69	0.0000	7	0.1000	15	0.1000	15	0.1667	16	0.2000	19	0.2333	25	0.2333	28
ALL	4611	0.0700	797	0.0987	1011	0.1087	1059	0.1727	1657	0.2207	1706	0.2473	1998	0.2333	1903
avg prec		0.0253		0.0459		0.0558		0.1292		0.1494		0.1703		0.1750	

Table 2: Per-query statistics for TREC-6. For each query we list the total number of relevant documents. For each algorithm, we record the precision at 30 documents and the recall at 1000 documents. We also report the average uninterpolated precision, averaged over all queries.

		base		+phrase		+syntax		+idf		+len		+feedback		+cluster	
Q#	rel	prec30	rel	prec30	rel	prec30	rel	prec30	rel	prec30	rel	prec30	rel	prec30	rel
251	579	0.3667	78	0.4333	65	0.1000	70	0.0667	69	0.0000	49	0.0000	31	0.0000	38
252	37	0.1000	9	0.1000	9	0.1000	9	0.1000	10	0.0000	8	0.0000	7	0.0333	12
253	10	0.0000	0	0.0000	0	0.0000	0	0.0667	2	0.0667	2	0.0667	6	0.0667	6
254	85	0.0000	1	0.0667	13	0.0667	23	0.0667	35	0.0667	36	0.1000	37	0.1333	41
255	109	0.0333	10	0.0333	4	0.0333	4	0.0000	3	0.0333	3	0.0000	3	0.0000	3
256	22	0.0000	3	0.0000	7	0.0000	7	0.0000	5	0.0667	11	0.0667	13	0.0667	13
257	135	0.1000	16	0.4000	33	0.4000	33	0.6000	76	0.6000	74	0.6000	92	0.6000	93
258	115	0.0000	11	0.0000	10	0.0333	21	0.1333	35	0.2000	37	0.3667	43	0.5000	53
259	36	0.0333	18	0.3333	28	0.4667	30	0.4333	33	0.4667	33	0.5000	35	0.5000	35
260	22	0.0000	4	0.0000	5	0.0000	10	0.0000	4	0.0000	2	0.0000	2	0.0000	2
261	87	0.1667	35	0.1667	53	0.1667	53	0.2000	54	0.3000	35	0.3333	35	0.3333	35
262	4	0.0667	4	0.1333	4	0.1333	4	0.1333	4	0.1333	4	0.1333	4	0.1333	4
263	15	0.0667	7	0.0000	11	0.0000	11	0.0000	11	0.0000	8	0.0000	8	0.0000	8
264	281	0.0333	14	0.0000	26	0.1000	48	0.3000	57	0.4000	60	0.5667	92	0.7333	118
265	147	0.0667	16	0.1667	31	0.2667	121	0.5000	127	0.7000	125	1.0000	128	1.0000	128
266	139	0.0000	2	0.0000	3	0.0000	3	0.0333	34	0.0333	32	0.0667	41	0.0667	41
267	4	0.0000	0	0.0000	0	0.0000	0	0.0000	1	0.0000	1	0.0000	1	0.0000	1
268	45	0.0000	8	0.0000	7	0.0000	4	0.0000	10	0.0000	14	0.0000	16	0.0000	16
269	594	0.1667	68	0.2000	66	0.2667	63	0.4667	80	0.4667	54	0.4333	78	0.4333	81
270	116	0.1333	43	0.3667	45	0.3667	45	0.5667	62	0.7667	59	0.9000	85	0.9000	85
271	86	0.0333	7	0.0667	12	0.0667	12	0.1333	24	0.1667	26	0.2000	30	0.2000	30
272	36	0.0333	14	0.2000	18	0.2000	18	0.4000	30	0.3667	28	0.3667	32	0.3667	32
273	513	0.2000	31	0.2000	37	0.3667	82	0.4333	113	0.5000	136	0.6667	298	0.7000	299
274	119	0.0000	38	0.1667	54	0.1667	54	0.3333	56	0.4000	50	0.7000	62	0.7000	62
275	19	0.0000	3	0.0000	0	0.0000	0	0.0000	5	0.1333	13	0.0667	15	0.0667	15
276	7	0.0667	7	0.2333	7	0.2333	7	0.2333	7	0.2333	7	0.2333	7	0.2333	7
277	74	0.0333	18	0.4667	45	0.4667	45	0.6667	51	0.6000	48	0.5667	51	0.5667	51
278	7	0.0000	1	0.0000	0	0.0000	2	0.0000	1	0.0000	1	0.0333	1	0.0333	1
279	2	0.0000	0	0.0000	1	0.0667	2	0.0000	2	0.0333	2	0.0333	2	0.0333	2
280	32	0.3000	21	0.4000	22	0.4000	22	0.3667	28	0.3667	25	0.4667	26	0.4667	26
281	1	0.0000	0	0.0000	0	0.0000	0	0.0000	1	0.0000	1	0.0000	1	0.0000	1
282	131	0.2000	36	0.0333	24	0.0333	24	0.2333	27	0.1333	28	0.2667	50	0.2667	50
283	84	0.0667	23	0.1333	14	0.1333	14	0.1333	18	0.0667	18	0.1667	44	0.1333	42
284	70	0.0000	4	0.0000	3	0.0000	6	0.1000	22	0.2333	28	0.3667	36	0.3667	37
285	261	0.4000	43	0.4000	52	0.4333	59	0.5000	143	0.5333	134	0.4667	139	0.4667	139
286	142	0.1000	65	0.7000	80	0.7000	80	0.7000	80	0.1667	16	0.2000	22	0.0333	22
287	40	0.1333	22	0.2333	26	0.2333	26	0.2000	28	0.2333	24	0.2333	25	0.2333	25
288	92	0.0000	10	0.1667	16	0.3667	27	0.4333	48	0.4667	53	0.4667	55	0.5000	55
289	141	0.0667	32	0.1333	32	0.1667	19	0.2333	58	0.3000	75	0.3000	73	0.2333	73
290	119	0.0000	14	0.0333	8	0.0333	8	0.0000	17	0.0667	29	0.0000	23	0.0000	23
291	407	0.0000	28	0.1667	24	0.1667	24	0.1000	13	0.1000	23	0.0667	16	0.0667	16
292	59	0.0000	3	0.0333	3	0.0333	4	0.0000	8	0.0333	5	0.0333	12	0.0000	3
293	41	0.0000	8	0.0000	4	0.0000	4	0.0667	8	0.0667	5	0.0667	5	0.0667	5
294	160	0.0000	9	0.0000	11	0.0000	13	0.0000	19	0.0000	23	0.0333	25	0.0000	26
295	15	0.1667	6	0.1333	6	0.1333	6	0.1667	9	0.1667	8	0.1667	8	0.1667	8
296	1	0.0000	0	0.0000	0	0.0000	0	0.0000	0	0.0000	0	0.0000	0	0.0000	0
297	86	0.0000	1	0.0333	19	0.0333	19	0.6333	23	0.6333	27	0.7333	39	0.7667	39
298	91	0.1333	13	0.1667	13	0.1667	13	0.2667	14	0.1667	26	0.3333	50	0.3000	49
299	62	0.1000	8	0.1333	10	0.1333	10	0.0667	11	0.0333	9	0.0333	11	0.0333	11
300	44	0.0000	1	0.0000	2	0.0000	2	0.0667	4	0.0667	18	0.0667	24	0.0667	24
ALL	5524	0.0673	813	0.1327	963	0.1447	1161	0.2027	1580	0.2113	1533	0.2493	1939	0.2513	1986
avg prec		0.0216		0.0618		0.0761		0.1192		0.1216		0.1489		0.1535	

Table 3: Per-query statistics for TREC-5. We used this data to tune the parameters for the algorithms we studied.

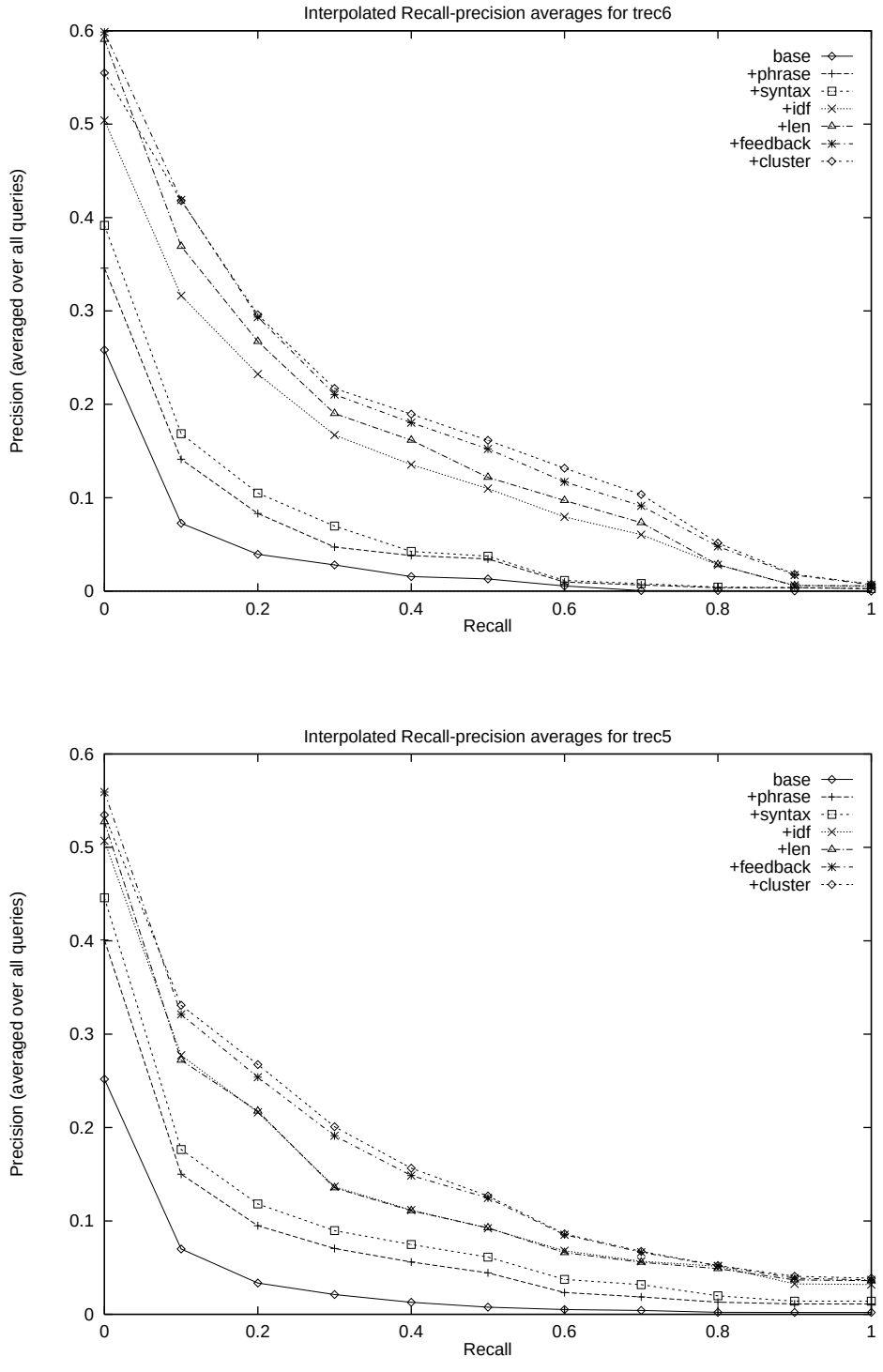

Figure 1: The interpolated recall-precision averages for the six algorithms discussed in this paper. The numbers are averaged over all 50 topics in TREC-6 (top figure) and TREC-5 (bottom figure).

3 The Routing Task

3.1 Method

Unlike our *ad hoc* submission, our *routing* submission focused on only using built-in functionality of the Verity search engine[4]. Since the routing task has training data in the form of positive and negative examples (documents), it made sense to use the built-in Verity Query By Example (QBE) functionality, which is manifested in Verity Query Language (VQL) using the `<LIKE>` operator. This operator takes several parameters:

- any number of positive examples,
- any number of negative examples, and
- the number of terms to include in the final query.

From these parameters, it then generates a query in VQL using only term frequency to perform term selection. It is important to remember that VQL does *not* have built-in support for IDF or document length normalization. Thus, compared to other TREC entries, the Verity QBE is at something of a disadvantage.

To train our system, we generated a classifier (a QBE query) for each routing query. The training process had to select which positive and negative examples to use, along with the number of query terms. Because of the large numbers of positive and negative examples, investigating all subsets of available examples was not tractable. Instead, we used a variation of "local region" training [Schütze et al., 1995], wherein training examples were first ranked using a QBE query with the actual TREC topic text as the single (positive) example. Selection of both positive and negative training examples was then based on the number of top ranked examples. Thus, training a classifier became a matter of selecting the number of top ranked positive examples, the number of top ranked negative examples, and the number of terms. Note that, except for determining the ranking of training examples, the original query text is not used.

We used average precision as a performance metric during training, since preliminary experiments indicated that it was more predictive of test set performance than precision at 30 documents (our preferred measure). Our preliminary experiments also indicated that as a function of the classifier parameters, average precision presented quite a "bumpy" optimization surface. Thus, more sophisticated optimization procedures did not seem helpful, and a version of grid-searching was used to select the parameters used for each query. Table 4 shows what values of parameters were used during training.

Two training regimens were used in conjunction with the grid-search. Both used only documents from "similar" sources as the test documents. The first regimen (regimen A) used training documents from the AP, WSJ, SJM, and FBIS corpora, with the best query parameters selected based on performance on all four corpora combined. These corpora were chosen because they seemed most similar in content and style to the test collection (more FBIS documents). Regimen B used a more aggressive source-sensitive approach. Here, training documents from AP, WSJ, SJM, and FBIS were used again, but all FBIS training examples were artificially ranked above documents from other collections. Furthermore, query parameters were selected based on performance on FBIS alone. Thus, regimen B was essentially just training on FBIS documents, but allowed the use of documents from the other collections when larger numbers of training examples were used.

[4]One slight change to the basic query construction procedure was made which theoretically should not significantly change performance, but which did improve the correlation between training and test set performance and also ameliorated the saturation problem. This change has since been incorporated into the next release of Verity's engine.

Parameter	Values Used for Training
# Positive Examples	5 10 30 80 160 all
# Negative Examples	0 5 10 20 40 160
# Terms	3 8 10 12 15 30

Table 4: Parameter Values Scanned to Train each QBE Query

Validation experiments on a portion of the training FBIS collection indicated that each regimen worked better for some queries than for others. Consequently, we decided to use a system selection approach, choosing a regimen individually for each query. Time constraints prevented investigation of different selection criteria, so we simply used whichever training regimen performed better in our validation runs.

In summary, our routing approach investigated several techniques based on using QBE straight "out-of-the-box":

1. training on a per-query basis,
2. local region training,
3. source specific training, and
4. system selection.

3.2 Results

A precision/recall graph of our submission (VrtyRT6) which used the system selection approach is shown in Figure 2. Also shown are graphs for both regimens and for an approach (labeled "fixed") which did not tune parameters on a per-query basis, but instead always used all positive examples, no negative, and 15 terms.

The system selection procedure is superior to regimen A but inferior to regimen B. Apparently our selection rule was not a good one, and did slightly worse than randomly choosing one of the two regimens. Our validation experiments showed the same behavior, but in those runs both regimens performed at about the same level, so the combined system showed neither improvement nor degradation. These results should not be surprising given that previous work on system selection has generally failed [Diamond, 1996].

Setting aside the system selection run, we note that both regimens perform better than the "fixed" run. Thus, it pays to tune the number of terms, the number of top positive examples and the number of top negative examples on a per-query basis. Furthermore, since regimen B does better than A, it also appears that it pays to emphasize training documents from the same source as the test set. Also, since about half of the trained queries resulting from regimen B made use of documents outside of the FBIS collection, it also seems important to make these extra documents available, instead of training exclusively on same-source documents. Our validation runs also support this conclusion. When the FBIS training data was split in half, with one half used for training and one for testing, regimen B consistently did better than only training on the FBIS documents (0.24 average precision versus 0.17-0.22, depending on which half was used for training). Finally, it appears that some queries were able to take advantage of the sorted order of training examples (the "local region" technique). About 33 of the 47 regimen A queries used less than the full number of positive examples.

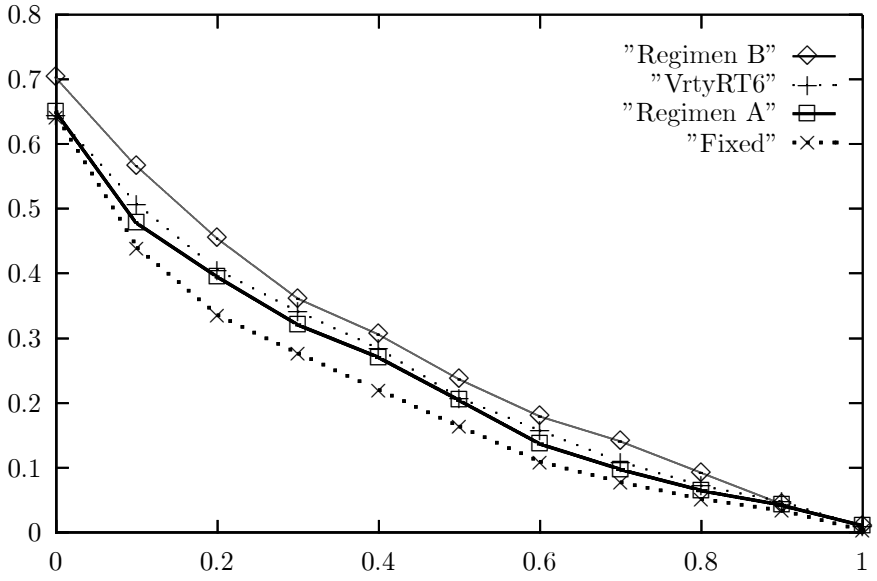

Figure 2: Precision vs. Recall for Routing Runs

Query	# Pos	# Neg	# Terms
3	all (561)	5	10
44	30	160	3
114	30	10	10
148	all (33)	40	3

Table 5: Parameters for Four Best Routing Queries

Figure 3 shows how the regimen B run compared to other routing entries. Overall, the performance is about average. However, only 16 queries exhibited above-median performance. Four were very significantly above the median, none were maximum, and only one was the worst. Over all queries, the mean average precision was approximately the same as the mean median score of all TREC entries.

Table 5 shows the parameters for the four best queries. No discernible pattern emerges, which supports the hypothesis that the parameter settings vary greatly according to the query.

Perhaps the most striking thing to note is that despite a lack of length normalization and IDF, our system was capable of producing reasonable results, on a par with the average TREC system. Judging by the boosts these techniques gave our *ad hoc* entry, it would not be unreasonable to assume the Verity's *routing* entry may have easily been more competitive.

4 Conclusions

Our results in the *routing* task indicate that despite a lack of standard techniques like IDF and length normalization, Verity QBE can perform on a par with other TREC entries, with appropriate selection of training examples and query length. Specifically, training on positive and negative examples which are "closest" to the original query text (the local region technique) provides a boost. Also, emphasizing training documents from the same source as the test collection helps.

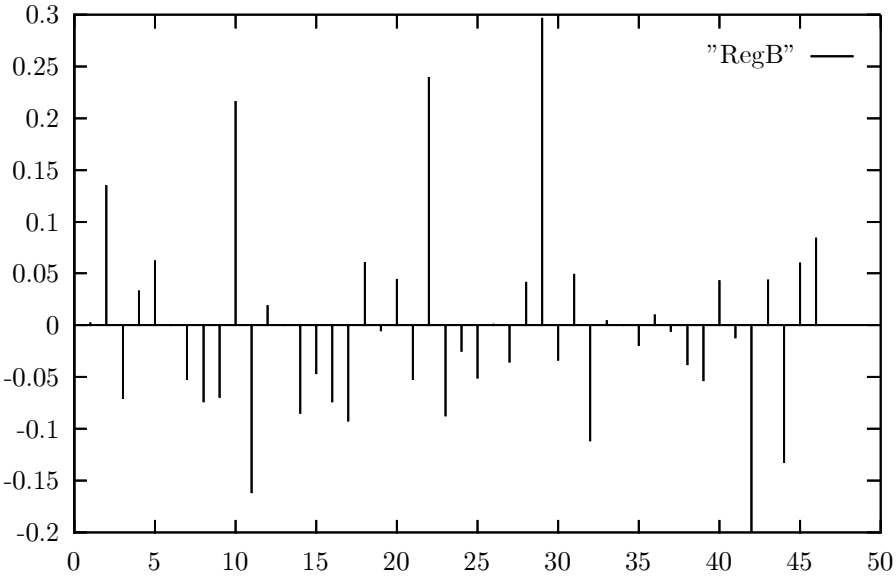

Figure 3: Routing Comparison to Median by Query for Regimen B

Both of these techniques presumably work by improving both term selection and term weighting. Also, by training a single classifier per query, and adjusting the number of terms in the resultant query, we were able to more accurately capitalize on the idiosyncrasies of each query.

The baseline *ad hoc* system is not nearly as impressive as the *routing*, presumably because it only has access to terms from the original TREC topic, and not from training examples. However, by selecting and weighting terms based on linguistic information and IDF, along with a novel length normalization approach and pseudo-feedback, we are able to significantly improve performance. By including these enhancements, we achieve performance on par with other TREC entries.

Most significantly, our experiments did not modify the well-established VQL extended boolean query language. As such, any of these enhancements could easily be incorporated as a built-in facility.

Acknowledgements

The authors would like to express special thanks to George Politowski who prepared the data, built Verity collections, and assisted the authors with the Verity toolkit (VDK).

References

[Diamond, 1996] Diamond, T. (1996). Information retrieval using dynamic evidence combination. PhD Dissertation Proposal. http://www.nwlink.com/~tgdiamon.

[Schütze et al., 1995] Schütze, H., Hull, D. A., and Pedersen, J. O. (1995). A comparison of classifiers and document representations for the routing problem. In Fox, E. A., Ingwersen, P., and Fidel, R., editors, *SIGIR 95: Proceedings of the Eighteenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 229–237, Seattle. Xerox PARC, ACM Press.