Probabilistic Learning Approaches for Indexing and Retrieval with the TREC-2 Collection

Norbert Fuhr, Ulrich Pfeifer, Christoph Bremkamp, Michael Pollmann
University of Dortmund, Germany
Chris Buckley
Cornell University

Abstract

In this paper, we describe the application of probabilistic models for indexing and retrieval with the TREC-2 collection. This database consists of about a million documents (2 gigabytes of data) and 100 queries (50 routing and 50 adhoc topics). For document indexing, we use a description-oriented approach which exploits relevance feedback data in order to produce a probabilistic indexing with single terms as well as with phrases. With the adhoc queries, we present a new query term weighting method based on a training sample of other queries. For the routing queries, the RPI model is applied which combines probabilistic indexing with query term weighting based on query-specific feedback data. The experimental results of our approach show very good performance for both types of queries.

1 Introduction

The good TREC-1 results of our group described in [Fuhr & Buckley 93] have confirmed the general concept of probabilistic retrieval as a learning approach. In this paper, we describe some improvements of the indexing and retrieval procedures. For that, we first give a brief outline of the document indexing procedure which is based on description-oriented indexing in combination with polynomial regression. Section 3 describes query term weighting for adhoc queries, where we have developed a new learning method based on a training sample of other queries and corresponding relevance judgements. In section 4, the construction of the routing queries is presented, which is based on the probabilistic RPI retrieval model for query-specific feedback data. In the final conclusions, we suggest some further improvements of our method.

2 Document indexing

The task of probabilistic document indexing can be described as follows (see [Fuhr & Buckley 91] for more details): Let d_m denote a document, t_i a term and R the fact that a query-document pair is judged relevant, then $P(R|t_i, d_m)$ denotes the probability that document d_m will be judged relevant w.r.t. an arbitrary query that contains term t_i. Since these weights can hardly be estimated directly, we use the description-oriented indexing approach. Here term-document pairs (t_i, d_m) are mapped onto so-called *relevance descriptions* $\vec{x}(t_i, d_m)$. The elements x_i of the relevance description contain values of features of t_i, d_m and their relationship, like e.g.

tf	within-document frequency (wdf) of t_i,
logidf	= log(inverse document frequency),
lognumterms	= log(number of different terms in d_m),
imaxtf	= 1/(maximum wdf of a term in d_m)
is_single	=1, if term is a single word, =0 otherwise
is_phrase	=1, if term is a phrase, =0 otherwise.

(As phrases, we considered all adjacent non-stopwords that occurred at least 25 times in the $D_1 + D_2$ (training) document set.)

Based on these relevance descriptions, we estimate the probability $P(R|\vec{x}(t_i, d_m))$ that an arbitrary term-document pair having relevance description $\vec{x}$ will be involved in a relevant query-document relationship. This probability is estimated by a so-called *indexing function* $u(\vec{x})$. Different regression methods or probabilistic classification algorithms can serve as indexing function.

For our retrieval runs submitted to TREC-2, we used polynomial regression for developing an indexing function of the form

$$u(\vec{x}) = \vec{b} \cdot \vec{v}(\vec{x}), \tag{1}$$

where the components of $\vec{v}(\vec{x})$ are products of elements of $\vec{x}$.

The indexing function actually used has the form

$$
\begin{array}{lllll}
u(\vec{x}) = & & & & \\
\quad b_0 & & & & + \\
\quad b_1 \cdot \mathit{is\_single} & \cdot \mathit{tf} & \cdot \mathit{logidf} & \cdot \mathit{imaxtf} & + \\
\quad b_2 \cdot \mathit{is\_single} & \cdot \mathit{tf} & \cdot \mathit{imaxtf} & & + \\
\quad b_3 \cdot \mathit{is\_single} & \cdot \mathit{logidf} & & & + \\
\quad b_4 \cdot \mathit{lognumterms} & \cdot \mathit{imaxtf} & & & + \\
\quad b_5 \cdot \mathit{is\_phrase} & \cdot \mathit{tf} & \cdot \mathit{logidf} & \cdot \mathit{imaxtf} & + \\
\quad b_6 \cdot \mathit{is\_phrase} & \cdot \mathit{tf} & \cdot \mathit{imaxtf} & & + \\
\quad b_7 \cdot \mathit{is\_phrase} & \cdot \mathit{logidf}. & & &
\end{array}
$$

The coefficient vector $\vec{b}$ is computed based on a training sample of query-document-pairs with relevance judgements.

Since polynomial functions may yield results outside the interval $[0, 1]$, these values were mapped onto the corresponding boundaries of this interval.

For each phrase occurring in a document, indexing weights for the phrase as well as for its two components (as single words) were computed.

There are two major problems with this approach which we are investigating currently:

1. Which factors should be used for defining the indexing functions? We are developing a tool that supports a statistical analysis of single factors for this purpose.
2. What is the best type of indexing function? Previous experiments have suggested that regression methods outperform other probabilistic classification methods. As a reasonable alternative to polynomial regression, logistic regression seems to offer some advantages (see also [Fuhr & Pfeifer 94]). As a major benefit, logistic functions yield only values between 0 and 1, so there is no problem with outliers. We are performing experiments with logistic regression and compare the results to those based on polynomial regression.

3 Query term weighting for ad-hoc queries

3.1 Theoretical background

The basis of our query term weighting scheme for ad-hoc queries is the linear utility-theoretic retrieval function described in [Wong & Yao 89]. Let q_k^T denote the set of terms occurring in the query, and u_{im} the indexing weight $u(\vec{x}(t_i, d_m))$ (with $u_{im} = 0$ for terms t_i not occurring in d_m). If c_{ik} gives the utility of term t_i for the actual query q_k, then the utility of document d_m w.r.t. query q_k can be computed by the retrieval function

$$\varrho(q_k, d_m) = \sum_{t_i \in q_k^T} c_{ik} \cdot u_{im}. \tag{2}$$

For the estimation of the utility weights c_{ik}, we applied two different methods.

As a heuristic approach, we used tf weights (the number of occurrences of the term t_i in the query), which had shown good results in the experiments described in [Fuhr & Buckley 91].

As a second method, we applied linear regression to this problem. Based on the concept of polynomial retrieval functions as described in [Fuhr 89b], one can estimate the probability of relevance of q_k w.r.t. d_m by the formula

$$P(R|q_k, d_m) \approx \sum_{t_i \in q_k^T} c_{ik} \cdot u_{im}. \tag{3}$$

If we had relevance feedback data for the specific query (as is the case for the routing queries), this function could be used directly for regression. For the ad-hoc-queries, however, we have only feedback information about other queries. For this reason, we regard *query features* instead of specific queries. This can be done by considering for each query term the same features as described before in the context of document indexing. Assume that we have a set of features $\{f_0, f_1, \ldots, f_l\}$ and that x_{ji} denotes the value of feature f_j for term t_i. Then we assume that query term weight c_{ik} can be estimated by linear regression according to the formula

$$c_{ik} = \frac{1}{|q_k^T|} \sum_{j=0}^{l} a_j x_{ji}. \tag{4}$$

Here the factor $|q_k^T|$ serves for the purpose of normalization across different queries, since queries with a larger number of terms tend to yield higher retrieval status values with formula 2. The factors a_j are the coefficients that are to be derived by means of regression. Now we have the problem that regression cannot be applied to eqn 4, since we do not observe c_{ik} values directly. Instead, we observe relevance judgements. This leads us back to the polynomial retrieval function 3, where we substitute eqn 4 for c_{ik}:

$$
\begin{aligned}
P(R|q_k, d_m) &\approx \sum_{t_i \in q_k^T} \frac{1}{|q_k^T|} \left(\sum_{j=0}^{l} a_j x_{ji} \right) u_{im} \\
&= \sum_{j=0}^{l} a_j \sum_{t_i \in q_k^T} \frac{1}{|q_k^T|} x_{ji} u_{im} \\
&= \sum_{j=0}^{l} a_j y_j \qquad (5)
\end{aligned}
$$

with

$$y_j = \sum_{t_i \in q_k^T} \frac{1}{|q_k^T|} x_{ji} u_{im} \tag{6}$$

Equation 5 shows that we can apply linear regression of the form $P(R|q_k, d_m) \approx \vec{a} \cdot \vec{y}$ to a training sample

of query-document pairs with relevance judgements in order to determine the coefficients a_j. The values y_j can be computed from the number of query terms, the values of the query term features and the document indexing weights.
For the experiments, the following parameters were considered as query term features:
x_0 =1 (constant)
$x_1 = tf$ (within-query-frequency)
$x_2 = \log tf$
$x_3 = tf \cdot idf$
$x_4 = is\_phrase$
$x_5 = in\_title$ (=1, if term occurs in query title, =0 otherwise)
For most of our experiments, we only used the parameter vector $\vec{x} = (x_0, \ldots, x_4)^T$. The full vector is denoted as $\vec{x}'$. Below, we call this query term weighting method *reg*.
This method is compared with the standard SMART weighting schemes:
nnn: $c_{ik} = tf$
ntc: $c_{ik} = tf \cdot idf$
lnc: $c_{ik} = (1 + \log tf)$
ltc: $c_{ik} = (1 + \log tf) \cdot idf$

3.2 Experiments

In order to have three different samples for learning and/or testing purposes, we used the following combinations of query sets and document sets as samples: Q3/D12 was used as training sample for the *reg* method, and both Q1/D12 and Q2/D3 were used for testing. As evaluation measure, we consider the 11-point average of precision (i.e., the average of the precision values at $0.0, 0.1, \ldots, 1.0$ recall).

	sample	
QTW	Q1/D12	Q2/D3
nnn	0.2303	
ntc	0.2754	
lnc	0.2291	0.2601
ltc	0.2826	0.2783
reg	0.2698	0.2678

Table 1: Global results for single words

First, we considered single words only. Table 1 shows the results of the different query term weighting (QTW) methods. First, it should be noted that the *ntc* and *ltc* methods perform better than *nnn* and *lnc*. This finding is somewhat different from the results presented in [Fuhr & Buckley 91], where the *nnn* weighting scheme gave us better results than the *ntc* method for lsp indexing. However, in the earlier experiments, we used only fairly small databases, and the queries also were much shorter than in the TREC collection. These facts may account for the different results.

		test sample		
run	learning sample	features	Q1/D12	Q2/D3
1	every doc.	$\vec{x}$	0.2698	0.2678
2	every 100. doc.	$\vec{x}$	0.2700	0.2678
3	every 1000. doc.	$\vec{x}$	0.2662	
4	judged docs only	$\vec{x}$	0.2635	0.2677
5	every doc.	$\vec{x}'$	0.2654	
6	every 100. doc.	$\vec{x}'$	0.2677	

Table 2: Variations of the *reg* learning sample and the query features

	run					
factor	1	2	3	4	5	6
constant	-3.05	-2.96	1.04	-20.80	2.47	-1.71
tf	-7.54	-10.67	-2.40	14.58	-4.32	14.11
$\log tf$	-9.01	-5.69	-5.08	-81.58	-1.48	-0.70
$tf \cdot idf$	5.84	7.00	1.63	8.72	1.81	7.70
is_phrase	-8.06	-19.23	-2.73	19.81	-2.97	20.39
in_title					-1.70	3.44

Table 3: Coefficients of query regression

In a second series of experiments, we varied the sample size and the set of features of the regression method (table 2). Besides using every document from the learning sample, we only considered every 100th and every 1000th document from the database, as well as only those documents for which there were explicit relevance judgements available. As the results show almost no differences, it seems to be sufficient to use only a small portion of the database as training sample in order to save computation time. The additional consideration of the occurrence of a term in the query title also did not effect the results. So query titles seem to be not very significant. It is an open question whether or not other parts of the queries are more significant, so that the consideration as an additional feature would affect retrieval quality.
The coefficients computed by the regression process for the second series of experiments are shown in table 3. It is obvious that the coeffients depend heavily on the choice of the training sample — so it is quite surprising that retrieval quality is not affected by this factor. The only coefficient which does not change its sign through all the runs is the one for the $tf \cdot idf$ factor. This seems to confirm the power of this factor. The other factors can be regarded as being only minor modifications of the $tf \cdot idf$ query term weight.
Overall, it must be noted that the regression method does not yield an improvement over the *ntc* and *ltc* methods. This seems to be surprising, since the regression is based on the same factors which also go into the

ntc and *ltc* formulas. However, a possible explanation could be the fact that the regression method tries to minimize the quadratic error for all the documents in the learning sample, but our evaluation measure considers at most the top ranking 1000 documents for each query; so regression might perform well for most of the documents from the database, but not for the top of the ranking list. There is some indication for this explanation, since regression yields always slightly better results at the high recall end.

α	result
0.00	0.3199
0.10	0.3707
0.15	0.3734
0.20	0.3700
0.25	0.3656
0.30	0.3610
0.50	0.3451
1.00	0.3147

Table 4: Effect of downweighting of phrases (sample Q2/D12)

As described before, in our indexing process, we consider phrases in addition to single words. This leads to the problem that when a phrase occurs in a document, we index the phrase in addition to the two single words forming the phrase. As a heuristic method for overcoming this problem, we introduced a factor for downweighting query term weights for phrases. That is, the actual query term weight of a phrase is $c'_{ik} = \alpha c_{ik}$, where c_{ik} is the result of the regression process. In order to derive a value for α, we performed a number of test runs with varying values (see table 4). Obviously, weighting factors between 0.1 and 0.3 gave the best results. For the official runs, we choose $\alpha = 0.15$.

		sample	
QTW	α	Q1/D12	Q2/D3
ltc	0.15	0.3192	0.3131
ltc	0.2	0.3220	0.3056
reg	0.15	0.3080	0.3062

Table 5: Results for single words and phrases

In table 5, this method is compared with the *ltc* formula, where we also choose a weighting factor for phrases which gave the best results. One can see that with the sample Q2/D3, the differences between the methods are smaller than on sample Q1/D12, but still *ltc* seems to perform slightly better.

Finally, we investigated another method for coping with phrases. For that, let us assume that we have binary query weights only. Now as an example, the single words t_1 and t_2 form a phrase t_3. For a query with phrase t_3 (and thus also with t_1 and t_2), a document d_m containig the phrase would yield $u_{1m} + u_{2m} + u_{3m}$ as value of the retrieval function, where the weights u_{im} are computed by the lsp method described before. In order to avoid the effect of counting the single words in addition to the phrase, we modified the original phrase weight as follows:

$$u'_{3m} = u_{3m} - u_{1m} - u_{2m}$$

and stored this value as phrase weight. Queries with the single words t_1 or t_2 are not affected by this modification. For the query with phrase t_3, however, the retrieval function now would yield the value $u_{1m} + u_{2m} + u'_{3m} = u_{3m}$, which is what we would like to get.

QTW	α	result
reg	0.00	0.2724
reg	1.00	0.2596
ntc	0.00	0.2754
ntc	0.15	0.3110
ntc	1.00	0.2524

Table 6: Results for the subtraction method (sample Q1/D12)

Table 6 shows the corresponding results ($\alpha = 0$ means that single words only are considered). In contrast to what we expected, we do not get an improvement over single words only when phrases are considered fully. The result for the *ntc* method shows that still phrases should be downweighted. Possibly, there may be an improvement with this method when we would use binary query term weights, but it is clear that other query term weighting methods mostly give better results.

3.3 Official runs

As document indexing method, we applied the description-oriented approach as described in section 2. In order to estimate the coefficients of the indexing function, we used the training sample Q12/D12, i.e. the query sets Q_1 and Q_2 in combination with the documents from D_1 and D_2.

Two runs with different query term weights were submitted. Run `dortL2` is based on the *nnn* method, i.e. *tf* weights. Run `dortQ2` uses *reg* query term weights. For performing the regression, we used the query sets Q_1 and Q_2 and a sample of 400,000 documents from D_1.

Table 7 shows the results for the two runs (Numbers in parentheses denote figures close to the best/worst results.). As expected, `dortQ2` yields better results than `dortL2`. The recall-precision curves (see figure 1) show that there is an improvement throughout the whole recall range. For precision average and precision at 1000 documents retrieved, run `dortQ2` performs very well, while precision at 100 documents retrieved is less good. This confirms our interpretation from above, saying that

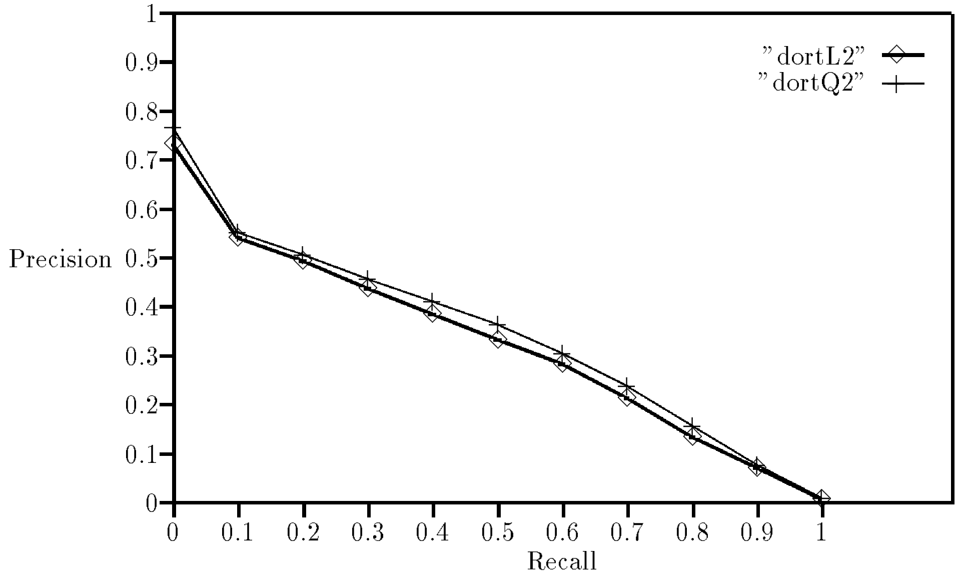

Figure 1: Recall-precision curves of ad-hoc runs

regression optimizes the overall performance, but not necessarily the retrieval quality when only the top ranking documents are considered. With regard to the moderate results for the *reg* query term weighting method, the good performance of `dortQ2` obviously stems from the quality of our document indexing method.

run	`dortL2`	`dortQ2`
query term weighting	*nnn*	*reg*
average precision:		
Prec. Avg.	0.3151	0.3340
query-wise comparison with median:		
Prec. Avg.	37:2	45:4
Prec. @ 100 docs	35:11	34:7
Prec. @ 1000 docs	37:10	45:2
Best/worst results:		
Prec. Avg.	3/0	3(1)/0
Prec. @ 100 docs	3(2)/1	4(1)/0(2)
Prec. @ 1000 docs	6(1)/0	9(1)/0
`dortL2` vs. `dortQ2`:		
Prec. Avg.	21:29	
Prec. @ 100 docs	22:24	
Prec. @ 1000 docs	17:29	

Table 7: Results for adhoc queries

4 Query term weighting for routing queries

4.1 Theoretical background

For the routing queries, the retrieval-with-probabilistic-indexing (RPI) model described in [Fuhr 89a] was applied. The corresponding retrieval function is based on the following parameters:

u_{im} indexing weight of term t_i in document d_m
D_k^R set of documents judged relevant for query q_k,
p_{ik} expectation of the indexing weight of term t_i in D_k^R,
D_k^N set of documents judged nonrelevant for query q_k,
r_{ik} expectation of the indexing weight of t_i in D_k^N.

The parameters p_{ik} and r_{ik} can be estimated based on relevance feedback data as follows:

$$p_{ik} = \frac{1}{D_k^R} \sum_{d_m \in D_k^R} u_{im}$$

$$q_{ik} = \frac{1}{D_k^N} \sum_{d_m \in D_k^N} u_{im}$$

Then the query term weight is computed by the formula

$$c_{ik} = \frac{p_{ik}(1 - r_{ik})}{r_{ik}(1 - p_{ik})} - 1$$

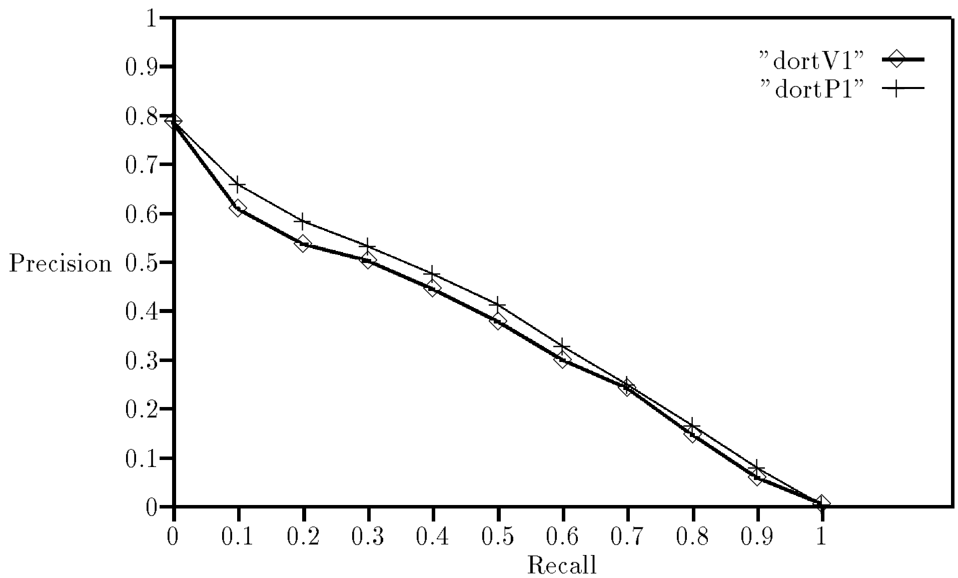

Figure 2: Recall-precision curves of routing runs

and the RPI retrieval function yields

$$\varrho(q_k, d_m) = \sum_{t_i \in q_k^T} \log(c_{ik} u_{im} + 1). \tag{7}$$

4.2 Experiments

In principle, the RPI formula can be applied with or without query expansion. For our experiments in TREC1, we did not use any query expansion. The final results showed that this was reasonable, mainly with respect to the small amount of relevance feedback data available. In contrast, for TREC2 there were about 2000 relevance judgements per query, so there was clearly enough training data for applying query expansion methods.
As basic criterion for selecting the expansion terms, we considered the number of relevant documents in which a term ocurs, which gave us a ranking of candidates; document indexing weights were considered for tie-breaking. Then we varied the the number of terms which are added to the original query.

expansion	result
0	0.2909
10	0.3047
30	0.3035
50	0.3002
100	0.2832

Table 8: Effect of number of expansion terms

In a first series of experiments, we considered single word only. We used Q2/D1 (*lsp* document indexing) as training sample and Q2/D2 (*ltc* indexing) as test sample. As can be seen from table 8, query expansion clearly improves retrieval quality, but only for a limited number of expansion terms. For larger numbers, we get worse results. This effect seems to be due to parameter estimation problems.

	expansion		
phraseweight	single w.	phrases	result
0.5	0	0	0.3476
0.5	20	0	0.3713
1.0	20	0	0.3730
0.5	20	10	0.3728
1.0	20	10	0.3605
0.5	30	10	0.3729
1.0	30	10	0.3626

Table 9: Query expansion with phrases

In a second series of experiments, we looked at the combination of single words and phrases. These experiments were performed as retrospective runs, with Q2/D12 as training sample and Q2/D2 as test sample (both with *ltc* document indexing). For the number of expansion terms, we treated single words and phrases separately. Furthermore, similar to the adhoc runs, we used an additional factor for downweighting the query term weights of phrases. The different parameter combinations tested and the corresponding results are given in table 9. Obviously, phrases as expansion terms gave no improvement, so we decided to have only single words as expansion terms (but the phrases from the original query still are used for retrieval). Furthermore, the retrieval quality reaches its optimum at about 20 terms.

4.3 Official runs

Two different runs were submitted for the routing queries, both based on the RPI model.
Run `dortP1` uses the same document indexing function as for the adhoc queries. Query terms were weighted according to the RPI formula. In addition, each query was expanded by 20 single words. Phrases were not downweighted.
Run `dortV1` is based on *ltc* document indexing. Here no query expansion took place.

run	`dortV1`	`dortP1`
document indexing	*ltc*	*lsp*
query expansion	none	20 terms
average precision:		
Prec. Avg.	0.3516	0.3800
query-wise comparison with median:		
Prec. Avg.	38:10	46:4
Prec. @ 100 docs	31:11	40:5
Prec. @ 1000 docs	32:9	37:7
Best/worst results:		
Prec. Avg.	1/0	4(2)/0
Prec. @ 100 docs	3(3)/1(1)	7(5)/1(1)
Prec. @ 1000 docs	6(2)/0(1)	10(2)/0(1)
`dortV1` vs. `dortP1`:		
Prec. Avg.	10:39	
Prec. @ 100 docs	9:27	
Prec. @ 1000 docs	7:33	

Table 10: Results for routing queries

Table 10 shows the results for the two runs. The recall-precision curves are given in figure 2. Again, the results confirm our expectations that LSP indexing and query expansion yields better results.

5 Conclusions and outlook

The experiments described in this paper have shown that probabilistic learning approaches can be applied successfully to different types of indexing and retrieval. For the ad-hoc queries, there seems to be still room for further improvement in the low recall range. In order to increase precision, a passage-wise comparison of query and document text should be performed. For this purpose, polynomial retrieval functions could be applied.
In the case of the routing queries, we first have to investigate methods for parameter estimation in combination with query expansion. However, with the large number of feedback documents given for this task, other types of retrieval models may be more suitable, e.g. query-specific polynomial retrieval functions.
Finally, it should be emphasized that we still use rather simple forms of text analysis. Since our methods are flexible enough to work with more sophisticated analysis procedures, this combination seems to be a prospective area of research.

A Operational details of runs

A.1 Basic Algorithms

The algorithm A to find the coefficient vector $\vec{a}$ for the ad-hoc query term weights can be given as follows:

Algorithm A

1 For each query document pair $(q_k, d_m) \in (Q_1 \cup Q_2) \times D_s$ with D_s being a sample from $(D_1 \cup D_2)$ do
 1.1 determine the relevance value r_{km} of the document d_m with respect to the query q_k.
 1.2 For each term t_i occuring in q_k do
  1.2.1 determine the feature vector $\vec{x}_i$ and the indexing weight u_{im} of the term t_i w.r.t. to document d_m.
 1.3 For each feature j of the feature vectors $\vec{x}$ compute the value of y_j looping over the terms of the query.
 1.4 Add vector $\vec{x}$ and relevance value r_{km} to the least squares matrix.
2 Solve the least squares matrix to find the coefficient vector $\vec{a}$

The algorithm B to find the coefficient vector $\vec{b}$ for the document indexing is sketched here:

Algorithm B

1 Index $D_1 \cup D_2$ (the learning document set) and $Q_1 \cup Q_2$ (the learning query set).

2 For each document $d \in D_1 \cup D_2$
 2.1 For each $q \in Q_1 \cup Q_2$
  2.1.1 Determine the relevance value r of d to q
  2.1.2 For each term t in common between q^T (set of query terms) and d^T (set of document terms)
   2.1.2.1 Find values of the elements of the relevance description involved in this run and add values plus relevance information to the least squares matrix being constructed
3 Solve the least squares matrix to find the coefficient vector $\vec{b}$

The algorithm C to index a document set D can now be given as:

Algorithm C

1 For each document $d \in D$
  1.1 For each term $t \in d^T$
    1.1.1 Find values of the relevance description $\vec{x}(t,d)$ involved in run.
    1.1.2 Give t the weight $\vec{b} \cdot \vec{v}(\vec{x}(t,d))$
  1.1 Add d to the inverted file.

A.2 Ad-hoc runs

The algorithm D is used for indexing and retrieval for the ad-hoc runs. Steps numbered with a trailing "A" apply only for run `dortQ2`, steps with trailing "B" only to run `dortL2`.

Algorithm D

1 Run algorithm B to determine the coefficient vector $\vec{b}$ for document indexing.
1A Run algorithm A to determine the coefficient vector $\vec{a}$ for query indexing.
2 Call algorithm C for document set $D_1 \cup D_2$
3 For each query $q_k \in Q_3$ do
  3.1 For each term t_i occuring in q_k do
    3.1.1A Determine the feature vector x_{ik} and compute the query term weight c_{ik} by multiplying it to $\vec{a}$.
    3.1.1B Weight t_i w.r.t. q_k (test query set) with tf weights (nnn variant). Phrases where downweighted by multiplying the weights with $\alpha = 0.15$.
  3.2 Run an inner product inverted file similarity match of $\vec{c_k}$ against the inverted file formed in step 2, retrieving the top 1000.

A.3 Routing runs

Algorithm E is used for indexing and retrieval for the routing runs. Steps numbered with a trailing "A" apply only for run `dortP1`, steps with trailing "B" only to run `dortV1`..

Algorithm E

1A Index query set Q_2 and document set $D_1 \cup D_2$ with $tf \cdot idf$ weights.
1B Index query set Q_2 and document set $D_1 \cup D_2$. by calling algorithm C
2 For each query $q \in Q_2$
  2.1 For each term $t \in q^T$ (set of query terms)
    2.1.1 Reweight term t using the RPI relevance weighting formula andthe relevance information supplied.
3A Index document set D_3 by calling algorithm C.
3B Index document set D_3 with $tf \cdot idf$ weights. *Note that the collection frequency information used was derived from occurrences in $D_1 \cup D_2$ only (in actual routing the collection frequencies within D_3 would not be known).*
4 Run the reweighted queries of Q_2 (step 2) against the inverted file (step 3), returning the top 1000 documents for each query.

References

Fuhr, N.; Buckley, C. (1991). A Probabilistic Learning Approach for Document Indexing. *ACM Transactions on Information Systems 9(3)*, pages 223–248.

Fuhr, N.; Buckley, C. (1993). Optimizing Document Indexing and Search Term Weighting Based on Probabilistic Models. In: Harman, D. (ed.): *The First Text REtrieval Conference (TREC-1)*, pages 89–100. National Institute of Standards and Technology Special Publication 500-207, Gaithersburg, Md. 20899.

Fuhr, N.; Pfeifer, U. (1994). Probabilistic Information Retrieval as Combination of Abstraction, Inductive Learning and Probabilistic Assumptions. *ACM Transactions on Information Systems 12(1)*.

Fuhr, N. (1989a). Models for Retrieval with Probabilistic Indexing. *Information Processing and Management 25(1)*, pages 55–72.

Fuhr, N. (1989b). Optimum Polynomial Retrieval Functions Based on the Probability Ranking Principle. *ACM Transactions on Information Systems 7(3)*, pages 183–204.

Wong, S.; Yao, Y. (1989). A Probability Distribution Model for Information Retrieval. *Information Processing and Management 25(1)*, pages 39–53.