Ranked Feature Fusion Models for Ad Hoc Retrieval

Jeremy Pickens, Gene Golovchinsky
FX Palo Alto Laboratory, Inc.
3400 Hillview Ave, Building 4
Palo Alto, California 94304 USA
{jeremy, gene}@fxpal.com

ABSTRACT

We introduce the Ranked Feature Fusion framework for information retrieval system design. Typical information retrieval formalisms such as the vector space model, the best-match model and the language model first combine features (such as term frequency and document length) into a unified representation, and then use the representation to rank documents. We take the opposite approach: Documents are first ranked by the relevance of a single feature value and are assigned scores based on their relative ordering within the collection. A separate ranked list is created for every feature value and these lists are then fused to produce a final document scoring. This new "rank then combine" approach is extensively evaluated and is shown to be as effective as traditional "combine then rank" approaches. The model is easy to understand and contains fewer parameters than other approaches. Finally, the model is easy to extend (integration of new features is trivial) and modify. This advantage includes but is not limited to relevance feedback and distribution flattening.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Retrieval Models

General Terms

Algorithms, Theory

Keywords

Feature Ranking, Metasearch, Score Normalization

1. INTRODUCTION

The modeling of the ad hoc information retrieval process using principled formalisms has a long history. Over the years, many models have been proposed, from the vector space model [12], to best-match $tf.IDF$ models [9], to inference networks [16] and language models [8] and even more recent "learning to rank" approaches [17]. These models combine raw statistical features from the collection (e.g. term frequencies, document lengths) into a single score that determines the retrieval rank of the document. Indeed, it is in the cleverness with which a model is able to combine the various features to produce a unified document score (e.g. by using document length as a normalization factor for term frequency) that a model succeeds or fails.

We take a different view. Rather than integrating raw statistical features directly, we first rank documents by the relevance of each feature independently, e.g. the frequency of a single query term, or the length of a document that contains a query term. We then combine the ranked lists that each feature produced. We name this general approach "ranked-feature fusion." The model is not defined by any particular ranking mechanism. The fusion strategy is also not a defining characteristic, as strategies other than the one we adopt in this paper are possible. What distinguishes the current approach is that multiple ranked lists of documents are created, one for each feature. These ranked lists are then combined to produce the final ranking. This work introduces an important notion: *The value of a single feature is not relative to other features in the document in which it resides; it is relative to its relevance-ranked position in a list of documents that contain the same feature.* We will show that this approach not only yields results that are as good as a well-known retrieval model, but does so with no parameter tuning and in a manner that is easier to understand and modify. We argue that this model offers an interesting and fertile new foundation for retrieval algorithm design.

2. MOTIVATION

Consider a two-word query: q_1 q_2. When these query terms appear in a document, a *combine then rank* approach produces a single score for that document based on the combination of all features (e.g. term frequency tf and document length dl) from the whole query. To slightly oversimplify, this typically translates to the sum or the product of the term frequencies, normalized by the document length:

$$score(d) = \frac{tf(q_1) + tf(q_2)}{dl(d)} \quad or \quad \frac{tf(q_1) \cdot tf(q_2)}{dl(d)}$$

Our *rank then combine* approach, on the other hand, treats each of these features (both tf and dl for terms q_1 and q_2) as independent retrieval subsystems. Each subsystem completes its own ranking of all the documents in the collection by that feature. The query above produces four ranked lists:

CIKM'08, October 26–30, 2008, Napa Valley, California, USA.

$R_{tf(q_1)}$, $R_{tf(q_2)}$, $R_{dl(q_1)}$ and $R_{dl(q_2)}$. $R_{tf(q_1)}$ is a ranked list of documents sorted by $tf(q_1)$. $R_{dl(q_2)}$ is a ranked list of documents sorted by $dl(q_2)$, the lengths of all the documents that contain q_2. These ranked lists are then combined, or fused, to produce the final document ranking R, using a fusion strategy such as CombSUM [1]. Let $R_{tf(q_1)}(d)$ be the normalized rank at which document d is found in $R_{tf(q_1)}$. The final score for d is:

$$score(d) = R_{tf(q_1)}(d) + R_{tf(q_2)}(d) + R_{dl(q_1)}(d) + R_{dl(q_2)}(d)$$

Some research has hinted at using term frequency [7] for fusion, but we are not aware of any experiments on such low level features. In this section we perform one such experiment to motivate the problem, and in later sections we expand this approach into a full retrieval model.

We compare sum_{tf} and $borda_{tf}$ that use only one class of feature, tf, the term frequency in a document. However, the first system uses raw term frequency, whereas the second system uses ranked term frequency. While better results can be obtained by using document lengths and document frequencies, comparing two systems using only the tf feature should give us insight into the rank-then-combine approach. (In Section 5 we will compare full-strength models.)

The first system, sum_{tf}, is a combine-then-rank approach. The overall score for a document is the combination of the sum of term frequencies. Documents are ranked by this combined score:

$$score(d) = \sum_{q \in Q} tf(q,d) \quad (1)$$

The second system, $borda_{tf}$, is a rank-then-combine approach. Documents are ranked by term frequency; the final score for a document is the sum of the borda counts, rather than the sum of the raw tf. Let $borda_{tf(t)}$ be a list of documents sorted by $tf(t)$ and normalized by Borda counts [1]. Borda assigns a score to each document based on its relative position in the list: $max(1000 - r, 1)$, where r is the original rank. The top ranked tf receives a score of 1000, the second ranked frequency a score of 999 and so on. If there are more than 1000 unique frequencies, a minimal score of 1 is assigned to all ranks past 1000. $borda_{tf(t)}(d)$ is the rank-based score for document d in that list:

$$score(d) = \sum_{q \in Q} borda_{tf(q)}(d) \quad (2)$$

We did a full-scale evaluation of $borda_{tf}$ versus sum_{tf}, using 400 queries from TREC1 through TREC8 (topics 51-450) on their appropriate full volumes. The queries are title-only, except for TREC4, which has a short description field. The results are found in Table 1. The sum of the tf ranks (rank-then-combine) far exceeds the performance of the sum of the raw tf (combine-then-rank) only.

Naturally, in full-strength retrieval systems various forms of document length normalization are used to account for the differences in tf magnitudes. The purpose of this example is simply to show that there are advantages to the ranked feature fusion approach.

	sum_{tf}	$borda_{tf}$	
			%Chg
Total number of documents over all queries			
Retrieved:	400000	400000	
Relevant:	63876	63876	
Rel|ret:	11729	20017	**+70.66**⋆
Precision:			
At 5 docs:	0.0880	0.2840	**+222.7**⋆
At 10 docs:	0.0780	0.2620	**+235.9**⋆
At 15 docs:	0.0720	0.2493	**+246.3**⋆
At 20 docs:	0.0691	0.2371	**+243.0**⋆
At 30 docs:	0.0652	0.2192	**+236.4**⋆
At 100 docs:	0.0549	0.1598	**+191.2**⋆
At 200 docs:	0.0487	0.1205	**+147.4**⋆
At 500 docs:	0.0380	0.0750	**+97.2**⋆
At 1000 docs:	0.0293	0.0500	**+70.7**⋆
Average precision (non-interpolated)			
	0.0351	0.1032	**+193.91**⋆

Table 1: $borda_{tf}$ compared against sum_{tf}. p<0.0001 (t-test) is indicated with ⋆

3. RELATED WORK

There are many ways in which ranked list fusion has been used to combine multiple sources of relevance evidence. For this discussion we refer to Figure 1 that illustrates the three conceptual levels at which fusion can take place. Typically ranked list fusion has been used as a foundation for metasearch [13, 1]: the same query is fed to multiple search engines that make different estimates of relevance. The combination of ranked lists from these engines increases both the precision and coverage of the resulting list. Metasearch fusion is sometimes also known as external fusion because the entire retrieval process is completed by the engines before their outputs are combined.

Internal fusion operates within single search engine [1, 7]. Documents contain various types of information such as bibliographic data, hyperlinks or citations, images, and unstructured text (see Figure 1). The output of separate, specialized search engines for each of these document subcontent types is fused into a single ranked representation.

Ranked feature fusion takes this notion a step further. It is possible, within a single subsystem, to rank the features that comprise that subsystem. Such approaches are not uncommon in multimedia retrieval domains, where the large state space often makes it easier to first classify (rank) lower-level features and then combine, rather than inventing a formal probabilistic model that unified all features [15]. However,

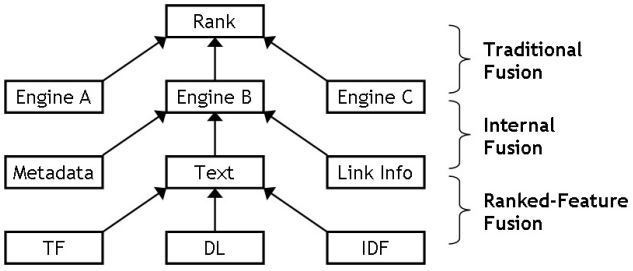

Figure 1: Traditional (external) fusion merges the ranked lists of multiple search engines. Internal fusion combines various subsystems, typically involving different data types, into a single search engine. Ranked Feature fusion combines raw features from a single data type (e.g. text)

in standard text retrieval there is a trend is in the opposite direction, toward more complex modeling or combination functions (e.g. probabilistic graphical models, support vector kernels). Some research does hint at pushing the fusion deeper into the retrieval process by using term frequencies as a retrieval subsystem [7] but we are not aware of any studies implementing that or similar sub-internal, ranked feature, retrieval subsystems. The goal of this work is to use the same raw features as other systems, namely *tf*, *dl* and *IDF*, and show that a simple yet effective retrieval algorithm can be composed from feature ranking and fusion.

4. RANKED FEATURE FUSION MODEL

Section 2 contains an interesting example of rank-then-combine effectiveness using term frequencies only. In this section we show how a complete system can be built from fusion components.

Aslam and Montague [1] distinguish between two different types of unsupervised fusion: fusion schemes that operate on ranked list information only (e.g. Borda count) and fusion schemes that make use of the magnitude of the relevance score returned from each engine (e.g. CombSUM); CombSUM was shown to outperform Borda count, and so we chose it as a foundation upon which the retrieval system is built [13, 1, 7]. The purpose of this work is not to introduce a new fusion model. It is to show how fusion provides a solid foundation for retrieval using only the basic features *tf*, *dl* and *IDF*, and to show how feature ranking creates new opportunities for system design. Three stages are necessary to describe the model: feature relevance ranking, ranked list normalization, and fusion.

4.1 Feature Relevance Ranking

In external and internal fusion (see Figure 1) a retrieval subsystem ranks documents using whatever information is available. Ranked feature fusion does the same, but only uses a single feature, such as $tf(q)$ or $dl(q)$.

To rank documents by relevance using only one feature, that feature needs to have different values in different documents. A feature that has the same value in every document does not discriminate between documents and cannot be used for ranking. Both *tf* and *dl* are suitable for relevance ranking: Query terms have differing term frequencies in different documents, and documents containing those query terms have different lengths. However, *IDF* alone cannot be used for ranking. Rather, it appears in the fusion model as a weighting factor (Section 4.3).

4.1.1 Term Frequency

Relevance ranking by *tf* is straightforward: documents with a higher frequency for a query term are ranked above documents with a lower frequency. Most of modern IR literature is based upon the notion that term frequency is positively correlated with relevance and we do not justify it further. Table 2 contains an example, TREC topic#63: *machine translation*. Columns 1 and 5 show *tf*, sorted by relevance. In the interest of space, multiple documents with the same value/rank are removed only the top and bottom twelve values are shown.

4.1.2 Document Length

Relevance ranking by *dl*, though on the surface not immediately obvious, is just as straightforward. One need only realize that most, if not all, combine-then-rank models (e.g. BM25 or language models) use document length as a *tf* normalization factor, i.e. $\frac{1}{dl}$. Document length is often inversely correlated with relevance, which means that shorter documents rank higher than long documents. Columns 9 and 13 in Table 2 show *dl*, sorted by relevance.

Using separate *dl*-based rankings for every query term means that one document has non-equal rank values in different lists. In our example, both terms appear in a document of length 21, but the document with that length is ranked 12^{th} in the *machine* list and 7^{th} in the *translation* list. The corresponding ranked-based document length scores are therefore different. *To our knowledge, this is the first information retrieval model in which a single document length score is given multiple values, depending on which term in that document is being matched.* It is beyond the scope of this paper to investigate this further, but this non-intuitive–and potentially useful–situation arises directly from the ranked-feature approach.

While in many cases short documents are preferred to longer ones, there exist collections for which longer documents contain better results. Whereas combine-then-rank approaches have to accommodate for this in complex ways (see [5], for example), our formalism requires only that the documents be re-ranked in order of ascending, rather than descending, document length. The document length preference can be specified by the user at query time to support exploratory search.

4.2 Ranked List Normalization

Now that we have subsystems for *tf*- and *dl*-based relevance ranking, we need a way of normalizing the ranks to fuse the lists [6]. In CombSUM fusion normalization is done via linear transformation [4, 6]; the underlying relevance scores are shifted and scaled to the min-max [0..1] range. The min-max score for document ranked by feature f is given by the following equation, where $f(d)$ is the original value of the feature in the document and $rank_f[1]$ and $rank_f[n]$ are original feature values in the first and last documents in the relevance-ordered list:

$$minmax_f(d) = \frac{f(d) - rank_f[n]}{rank_f[1] - rank_f[n]} \quad (3)$$

In our implementation, rather than shifting and scaling to [0..1] range, we shift and scale to the [1..1000] range. We set the minimum to 1.0 because even the lowest ranked item in a *tf* or *dl* or any other relevance-ranked retrieval list should have a non-zero value.[1] For easy comparison of these (M)inmax values in Table 2 the maximum is set to 1000 to bring it onto the same scale as Borda count. This scaling is irrelevant, however: every ranked list is scaled by the same factor and therefore has no effect on the final rankings. Thus for term frequencies of query term q the score is:

$$minmax_{tf(q)}(d) = \left(\frac{tf(q) - rank_{tf(q)}[n]}{rank_{tf(q)}[1] - rank_{tf(q)}[n]} * 999 \right) + 1$$

[1]This is not smoothing in a language modeling sense. Shifting the minimum to 1.0 still does not account for estimates of zero probability, i.e. a term not being found in a document. How a system deals with missing query terms in the ranked-feature fusion approach is an interesting question but beyond our current scope.

Term Frequency *tf*								Document Length *dl*							
Machine				Translation				Machine				Translation			
O	B	M	F	O	B	M	F	O	B	M	F	O	B	M	F
438	1000	1000.0	1000.0	**60**	1000	1000.0	1000.0	**8**	1000	1000.0	1000.0	**4**	1000	1000.0	1000.0
432	999	986.3	1000.0	**44**	999	729.1	1000.0	**11**	999	999.6	1000.0	**7**	999	999.6	1000.0
228	998	519.9	1000.0	**35**	998	576.7	1000.0	**12**	998	999.5	1000.0	**12**	998	998.9	1000.0
57	997	129.0	1000.0	**26**	997	424.3	1000.0	**13**	997	999.4	1000.0	**16**	997	998.4	1000.0
48	996	108.4	1000.0	**23**	996	373.5	1000.0	**14**	996	999.3	1000.0	**18**	996	998.2	1000.0
43	995	97.0	893.7	**22**	995	356.6	954.6	**15**	995	999.1	999.9	**19**	995	998.0	999.9
41	994	92.4	851.2	**20**	994	322.7	863.8	**16**	994	999.0	999.8	**21**	994	997.8	999.6
39	993	87.9	808.7	**19**	993	305.8	818.4	**17**	993	998.9	999.6	**22**	993	997.6	999.5
37	992	83.3	766.2	**18**	992	288.8	773.0	**18**	992	998.8	999.5	**23**	992	997.5	999.3
34	991	76.4	702.4	**15**	991	238.1	636.7	**19**	991	998.6	999.4	**24**	991	997.4	999.2
32	990	71.9	659.9	**14**	990	221.1	591.3	**20**	990	998.5	999.3	**25**	990	997.2	999.1
30	989	67.3	617.4	**13**	989	204.2	545.9	**21**	989	998.4	999.1	**26**	989	997.1	998.9
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12	972	26.1	234.8	**12**	988	187.3	500.5	**6403**	1	206.7	206.8	**5462**	1	281.7	282.2
11	971	23.9	213.6	**11**	987	170.3	455.1	**6420**	1	204.6	204.7	**5567**	1	267.9	268.4
10	970	21.6	192.3	**10**	986	153.4	409.7	**6468**	1	198.6	198.8	**5568**	1	267.8	268.3
9	969	19.3	171.0	**9**	985	136.5	364.3	**6539**	1	189.8	189.9	**5600**	1	263.5	264.0
8	968	17.0	149.8	**8**	984	119.5	318.9	**6563**	1	186.8	187.0	**5650**	1	257.0	257.4
7	967	14.7	128.5	**7**	983	102.6	273.5	**6656**	1	175.3	175.4	**6023**	1	207.9	208.3
6	966	12.4	107.3	**6**	982	85.7	228.0	**6970**	1	136.3	136.4	**6061**	1	202.9	203.3
5	965	10.1	86.0	**5**	981	68.7	182.6	**7196**	1	108.3	108.4	**6333**	1	167.1	167.4
4	964	7.9	64.8	**4**	980	51.8	137.2	**7213**	1	106.2	106.3	**6539**	1	140.0	140.2
3	963	5.6	43.5	**3**	979	34.9	91.8	**7261**	1	100.2	100.3	**6563**	1	136.8	137.1
2	962	3.3	22.3	**2**	978	17.9	46.4	**7595**	1	58.8	58.9	**7261**	1	45.0	45.0
1	961	1.0	1.0	**1**	977	1.0	1.0	**8061**	1	1.0	1.0	**7595**	1	1.0	1.0

Figure 2: (O)riginal, (B)orda count, (M)in-Max normalized, and (F)lattened values for various term frequency and document length scores for the terms *machine* and *translation* on TREC volumes 1&2

Similarly, for document lengths:

$$minmax_{dl(q)}(d) = \left(\frac{dl(q) - rank_{dl(q)}[n]}{rank_{dl(q)}[1] - rank_{dl(q)}[n]} * 999 \right) + 1$$

4.3 Fusion

The general form of CombSUM is based on a set of retrieval subsystems S and subsystem mixture weights λ_s. $minmax_s$ is the ranked list produced by subsystem s and $minmax_s(d)$ is the rank-normalized score of document d in that list.

$$score(d) = \sum_{s \in S} \lambda_s minmax_s(d) \quad (4)$$

For each term q in a query Q there are two retrieval subsystems, $minmax_{tf(q)}$ and $minmax_{dl(q)}$ In external fusion applications of CombSUM, the mixture weights λ_s are typically set through supervised training. Recall from Section 4.1 that we cannot rank documents by *IDF* because it varies across terms, rather than across documents. Therefore, it is used as an *unsupervised* mixture weight of the ranked lists produced by the various term-based features. For more direct comparison with our baseline (BM25) in the next section, we use *IDF* [11], where N is the total number of documents in the collection, and n is the number of documents in which the query term is found (which is also equivalent to the length of a ranked list for that feature, from Equation 3 above):

$$IDF(q) = \log \frac{(N - n + 0.5)}{(n + 0.5)} \quad (5)$$

To give this score a mixture weight interpretation, we normalize by the sum of IDF scores used for fusion, i.e. *two* per query term (one for the *tf* ranked list, one for the *dl* list):

$$\lambda_{idf(q)} = \frac{IDF(q)}{2 \cdot \sum_{q_i \in Q} IDF(q_i)} \quad (6)$$

Finally, the core retrieval algorithm, weighted CombSUM, is simply λ-weighted combination of all ranked lists, $minmax_{tf(q)}$ and $minmax_{dl(q)}$:

$$score(d) = \sum_{q \in Q} minmax_{tf(q)}(d) \cdot \lambda_{idf(q)} + \sum_{q \in Q} minmax_{dl(q)}(d) \cdot \lambda_{idf(q)} \quad (7)$$

Other than the +0.5 value inside of the *IDF* score no external parameters are set or tuned, or even needed. Everything about the model, from the relevance-ordered ranking to the min and max values for each subsystem, are completely specified by statistics from the collection.

5. EVALUATION

As in Section 2, the evaluation was done on 400 queries, TREC1 through TREC8 (topics 51-450). Stopwords were removed, but no stemming was done on either system. For our baseline, we chose Okapi BM25. Naturally, tuning the parameters k_1 and b affects performance. We use the values of $k_1 = 2.0$ and $b = 0.75$, deemed by Robertson and Spärck Jones [10] as effective values on TREC collections:

$$BM25(d) = \sum_{q \in Q} \frac{tf(q,d)(k_1 + 1)}{tf(q,d) + k_1((1-b) + b\frac{dl}{avgdl})} \cdot IDF(q) \quad (8)$$

We compare against BM25 for a specific reason. The the *IDF* function used in BM25 is the same function, Equation 5, that we use as a mixture for weighted fusion. In

	BM25	Ranked-Feature Fusion Model (RFM)					
			%Chg	+/±	p-value (sign)	p-value (t-test)	p-value (wilcoxon)
Total number of documents over all queries							
Retrieved:	400000	400000					
Relevant:	63876	63876					
Rel|ret:	22690	22274	-1.83	180/318	0.9907	0.3181	0.8644
Precision:							
At 5 docs:	0.3930	0.3870	-1.5	107/223	0.2734	0.6165	0.3254
At 10 docs:	0.3580	0.3592	+0.3	136/249	0.0725	0.8885	0.4153
At 15 docs:	0.3310	0.3320	+0.3	139/271	0.3353	0.8947	0.3021
At 20 docs:	0.3113	0.3114	+0.0	147/281	0.2190	0.9849	0.2781
At 30 docs:	0.2814	0.2845	+1.1	159/297	0.1115	0.5899	0.1237
At 100 docs:	0.1946	0.1928	-0.9	171/332	0.7084	0.6331	0.6944
At 200 docs:	0.1452	0.1413	-2.7	171/330	0.7456	0.1718	0.5275
At 500 docs:	0.0889	0.0876	-1.5	185/334	0.9756	0.4778	0.8135
At 1000 docs:	0.0567	0.0557	-1.8	180/318	0.9907	0.3181	0.8644
Average precision (non-interpolated							
	0.1330	0.1388	+4.39	201/392	0.3068	0.1119	0.1642

Table 2: Ranked-feature Fusion Models (RFM) compared with BM25. We give the actual signs +/± and p-values of three statistical tests (sign, t-test and wilcoxon sign-rank) for comparison. There are no statistically-significant differences between the two systems.

fact, the denominator in Equation 6 is somewhat unnecessary because it is the same for every ranked list being fused. Replacing $\lambda_{idf(q)}$ with $IDF(q)$ and refactoring Equation 7 yields:

$$score(d) = \sum_{q \in Q} \left(minmax_{tf(q)}(d) + minmax_{dl(q)}(d) \right) \cdot IDF(q)$$

Comparing Equations 7 and 8 shows that what distinguishes the BM25 model and our implementation of ranked-feature fusion is the treatment of *tf* and *dl*. This allows us to compare directly a "combine-then-rank" approach against a "rank-then-combine" approach, i.e.:

$$\frac{tf(q,d)(k_1+1)}{tf(q,d)+k_1((1-b)+b\frac{dl}{avgdl})} \quad vs. \quad \begin{array}{l} minmax_{tf(q)}(d)+ \\ minmax_{dl(q)}(d) \end{array}$$

Table 2 contains the results of this experiment plus the p-values for a sign test, t-test and wilcoxon signed-rank test. While MAP of ranked-feature fusion is slightly higher than BM25, none of the differences are statistically significant. The two systems perform equally well, on average. This is interesting, almost surprising. Standard CombSUM fusion of *tf* and *dl* retrieval subsystems performs just as well as the more complicated BM25.

Sometimes the value of an information retrieval model comes not through increased system performance, but from providing a way to think about the problem in a new manner [8]. Ranked-feature fusion offers this possibility. Rather than basing the retrieval score on probabilities, vector spaces, or other combine-then-rank models, documents are ranked by the relevance of single features. The final ranking then is nothing more than fusion of these ranked lists.

This model is easy to understand and conceptually straightforward, and is easy to extend and modify. Adding a feature to the model is primarily a matter of creating a relevance-ranking subsystem for that feature. Furthermore, the rank-centric approach more easily allows altering the effects of individual features by changing how that feature is ranked. In the next section we will show one method for modifying the model. The model can also be modified by changing the manner in which fusion is done. We propose and evaluate an alternate form of fusion based on the hedge algorithm as a method for implementing relevance feedback. In Section 7 we suggest many more extensions.

6. EXTENSIONS

We have shown that our model produces search results that are comparable to a well-regarded model. While these results are encouraging, they are not sufficient to base a research agenda on a formalism. We must also show that the formalism can be extended fruitfully to support useful operations. In this section, we evaluate two such extensions.

6.1 Outlier Flattening

Term frequency distributions are Zipfian, which means that there is a small number of documents in which a term occurs quite frequently and a long tail of documents in which the term occurs less frequently. Term frequency is a valuable indicator of relevance, but researchers have long observed that this contribution should not be linear; the marginal utility of term frequency as a diagnostic of document relevance declines as term frequency increases. Some systems utilize $log(tf)$ to model this decreasing utility, whereas BM25 uses more of an $\frac{tf}{tf+c}$ approach, which also exhibits asymptotic behavior at higher *tf* values.

These existing methods all introduce non-linearity as function of the raw data. How these ad hoc functions are chosen, and with what parameter values, are not intuitively clear. Our ranked feature approach offers an alternative: selection of an order-statistic. Recall from Equation 3 that a relevance-ranked feature list is normalized using the values at the first and last positions in the ranked list. To remove high *tf* outliers, we instead take the approach of selecting the k^{th} item in the ranked feature list, and setting every value that is ranked higher than this value to the maximum value in the normalized range [1..1000], i.e. we "clip" the top k items in the list. We normalize the remainder of the ranked list using the k^{th} value as the new maximum.

$$flattened_f(d) = \begin{cases} max & : \quad rank(d) < k \\ \frac{f(d) - rank_f[n]}{rank_f[k] - rank_f[n]} & : \quad rank(d) \geq k \end{cases} \qquad (9)$$

Figure 3 plots this effect for one term's *tf* values and Table 2 gives the scores for the ongoing *machine translation* query example. Order statistic-based transformations of ranked lists have the following properties:

1. The dominance of high-frequency terms is reduced.

2. k^{th} value renormalization increases the dynamic range, or boosts the contrast, of the renormalized rank values. As high-frequency outliers are discounted, lower values spread out to fill the [1..1000] normalization range. This creates more discriminatory power between middle and low-level feature values.

3. It is easier to choose order statistics than to than to create *ad hoc* functional forms for raw value renormalization, especially when raw values are tied to other features (e.g., document length in BM25).

4. Different features have different normalization values. For example, a term with tf=4.0 will always have the same renormalized value as a different term with tf=4.0 using either $log(tf)$ or BM25 non-linearity functions. However, as was mentioned in Sections 1 and 4.1.2, because the document containing the terms likely occupies a different rank in the respective *tf* ranked list for each term, order statistic flattening will produce different values, thus reflecting the relative importance of that term to each document.

5. This clipping level can be set interactively, allowing the user to discount high-frequency terms differently in different situations.

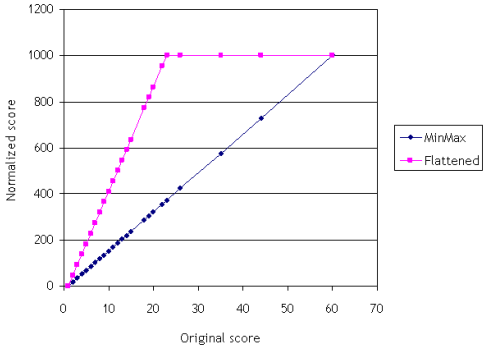

Figure 3: Ranked lists for *tf* of *translation*. Comparison of Min-max versus Flattened normalization.

Therefore, outlier flattening (Table 2) is done not only on term frequency features but also on document length features, or on any ranked feature list. For our experiments, we chose a threshold k of 5 to reflect the common Precision@5 metric and repeated the retrieval evaluation described in the previous section. This simple transformation resulted in improvements in overall system performance, as shown in Table 3 in the (RFM+Flat) column.

On average, the transformation resulted in a 4.41% improvement (p<0.01) in recall with no statistically-significant decrease in *MAP*. The transformation showed statistically-significant (p<0.01) improvements in precision at 100 documents and higher, which is likely the result of the discriminatory spreading mentioned above. These results suggest that the approach has some merit; simply using the k^{th} order statistic from a ranked-feature list as a normalizing value improves retrieval.

6.2 Online Relevance Feedback

Relevance feedback is another extension that suggests the robustness of a model. Our goal in this experiment is not to demonstrate that relevance feedback works; this fact is well-studied in the literature. Our goal is to show that the ranked-feature fusion model is capable of supporting relevance feedback, and furthermore does so in a non-traditional manner that opens up new ways of thinking about retrieval algorithm design.

This approach is similar to the online hedge algorithm for ranked list fusion [2]. In external fusion of multiple search engine lists, [2] dynamically alters the weight assigned to each engine's list in accordance with the number of relevant documents found while traversing that list. As certain ranked lists show themselves to be more "trustworthy" by containing larger numbers of relevant documents, a higher weight is given to the documents coming from those lists. This hedge algorithm uses loss functions and learning rates to update the mixture weights.

We adapt this intuition to the ranked feature fusion model in the same online manner. As the ranked list for a particular feature (e.g. $minmax_{dl(q)}$) finds more relevant documents than another feature's list, weights change to favor that list. Documents that have already been seen cannot be re-ordered, but remaining unseen documents in the subsystem queues can. This produces a final ranking, the overall order in which the documents have been traversed.

However, instead of using loss functions and learning rates, there is another alternative. Recall from Equation 5 in Section 4.3 that the mixture weights for ranked feature subsystems are set using $\lambda_{idf(q)} = IDF(q)$. This can be adapted when relevance information is available. A well-known approach is [9]. R is the number of documents thus far known to be relevant to a topic and r_q is the number of documents that contain query term q:

$$\lambda_{idf(q,R)} = \log \frac{\frac{r_q+0.5}{R-r_q+0.5}}{\frac{n_q-r_q+0.5}{N-n_q-R+r_q+0.5}} \qquad (10)$$

We use Equation 10 as the mixture weight for the $minmax_{tf(q)}$ and $minmax_{dl(q)}$ by reinterpreting r_q as $r_{tf(q)}$ and $r_{dl(q)}$. These are the numbers of relevant documents in the ranked lists for feature $tf(q)$ and $dl(q)$, respectively. Table 3 in the (RFM+Rel) column shows these improvements over the RFM baseline. Unsurprisingly, it works quite well; there is a 10-11% statistically significant (p<0.01) improvement in recall, MAP, and precision at various recall points. And, likely due to the online nature of the fusion, the more relevant documents found during the traversal of fused lists the better the fusion becomes. Again, the point here is not to demonstrate that relevance feedback works; it is to demonstrate that it works within the structure and form of ranked-feature fusion models.

6.3 Flattening with Relevance Feedback

Finally, we can combine the online relevance feedback of the previous section with the outlier flattening transformation described in Section 6.1. One can be performed independently of the other; we now evaluate whether the effects are cumulative.

The results in the (RFM+Flat+Rel) column of Table 3 show that the combination of the flattening with relevance feedback outperforms either extension alone, and is better than a linear combination of improvements. This makes sense: One of the effects of flattening was to increase the discriminatory power between terms in a subsystem ranked list. Relevance feedback alters the mixture weight on a ranked list. A small shift in mixture weights therefore has a larger effect on the documents within a list.

The analyses in this section demonstrate two of the many possible extensions of the basic model. They also demonstrate how easily the extensions can be combined. These extensions improve the performance of the model and lead us to believe that the formalism is more than a clever hack that yields good results for a specific problem. It offers a new framework in which to think about the retrieval problem.

7. FUTURE WORK

There are many possibilities for model extensions and improvements. The previous section suggested a few of these and we now offer an overview of three directions future work may develop. These are alternative rankings, improved fusion, and additional features.

7.1 Alternative Rankings

In this work documents were ranked by their literal term frequencies and document lengths. Even when normalized to the [1..1000] range, this ranking is still linear with respect to the underlying feature. Flattening was one way of introducing non-linearity into the ranking function. Naturally, a more principled specification of the flattening function, e.g. learning the best threshold t, will need to be derived and tested experimentally. The point is simply that ranked-feature fusion offers a new way of thinking about retrieval algorithm design. Improvements to precision and recall are made not by tweaking a term weight; they are made by selecting an order statistic. In addition to flattening, it may also be possible to derive a continuous function based on feature-specific information-theoretic considerations.

We also note the abstract similarity of outlier flattening (Section 6.1) to pivoted document length normalization [14]. The act of choosing an order statistic as a renormalization point has the effect of pivoting not only document lengths $minmax_{dl(q)}$ but term frequencies as well $minmax_{tf(q)}$. There is a lot of work around pivoted document length normalization that could be applied to the problem of choosing effective order statistic renormalization points.

7.2 Improved Fusion

This work used weighted CombSUM as a fusion mechanism. There are many more possibilities, such as Bayes Optimal fusion, Condorcet fusion, CombMNZ, etc. Although we apply fusion at a different conceptual level than metasearch engines, existing research is applicable to this domain and better fusion strategies are likely available. At the very least, one next step using the current approach is to decouple the $\lambda_{idf(q)}$ parameter. Currently both $minmax_{tf(q)}$ and $minmax_{dl(q)}$ retrieval subsystems use the same $\lambda_{idf(q)}$. It may be possible to decouple these parameters and use different weights on each list.

7.3 Additional Features

Many low-level features can be computed on documents, not just the *tf* and *dl* features used in this work. In general, any feature that discriminates between documents can be used for ranking. For example, the *dl* feature used in this paper was a function of the number of terms in a document. It is possible to use multiple forms of *dl*, e.g. length as a function of sentences, paragraphs, text tiles [3], etc., to create separate ranked lists for each. The same can be done for *tf*: term frequency can be a function not of the number of words in that "contain" t, but the number of sentences, paragraphs and text tiles that contain t. A document that ranks highly by all four $tf(q)$ subsystems, i.e. a document that outranks most other documents for term frequency, sentence frequency, paragraph frequency and text tile frequency is more likely to be relevant than a document that ranks high on term frequency but low on paragraph frequency. Multiple term frequency and document length metrics can improve overall rankings. The feature-rank fusion model can incorporate multiple types of *tf*, seamlessly and simultaneously.

Stemming, synonyms, hypernyms and hyponyms are also natural extensions of the feature list fusion model. For morphological variants (e.g. *giraffe* versus *giraffes*) in a combine-then-rank approach there is no difference between $tf(giraffe) + tf(giraffes)$ and $tf(giraffe \vee giraffes)$. However, in a rank-then-combine model, separate ranked lists with very different distributional properties exist for $tf(giraffe)$, for $tf(giraffes)$ and for $tf(giraffe \vee giraffes)$. The same document will not necessarily rank as high in one list as in another. Documents that rank high in *all* three lists are more likely to be relevant than documents that only rank high in a single list.

Proximity and co-occurrence operators can be integrated as well. Documents can be ranked by the number of times query words co-occur within a given window. Alternatively, documents can be ranked on the proximity or density of query terms. Documents with a low spread will rank higher than those with a higher spread.

8. CONCLUSIONS

We described an information retrieval model based on fusing ranked lists computed from primitive features of documents. We evaluated the effectiveness of this algorithm by comparing it with BM25: we replaced BM25's *dl*-normalized *tf* score with the fusion of *tf* and *dl* ranked lists and showed that there was no difference in system performance. We showed that term reweighting can be expressed as a function of rank list renormalization based on order statistics, and that relevance feedback may be expressed as hedge fusion. Both extensions improved system performance, and the combination was better than a sum of individual improvements.

The model we describe is conceptually simple, and unlike other models that use parameters to tweak system behavior, it relies on features whose values get relevance-ranked and fused. The value of a single feature is not relative to other features in the document in which it resides; it is relative to its relevance-ranked position in a list of documents that

	RFM	RFM+Flat		RFM+Rel		RFM+Flat+Rel		
			%Chg (RFM)		%Chg (RFM)		%Chg (RFM)	%Chg (RFM+Rel)
Total number of documents over all queries								
Retrieved:	400000	400000		400000		400000		
Relevant:	63876	63876		63876		63876		
Rel|ret:	22274	23256	**+4.41⋆**	24763	**+11.17⋆**	25851	**+16.06⋆**	**+4.39⋆**
Precision:								
At 5 docs:	0.3870	0.3985	+3.0	0.3945	+1.9	0.4075	**+5.3⋆**	+3.3
At 10 docs:	0.3592	0.3608	+0.4	0.3675	**+2.3⋆**	0.3740	**+4.1⋆**	+1.8
At 15 docs:	0.3320	0.3330	+0.3	0.3428	**+3.3⋆**	0.3478	**+4.8⋆**	+1.5
At 20 docs:	0.3114	0.3136	+0.7	0.3269	**+5.0⋆**	0.3333	**+7.0⋆**	+2.0
At 30 docs:	0.2845	0.2901	+2.0	0.2958	**+4.0⋆**	0.3059	**+7.5⋆**	**+3.4⋆**
At 100 docs:	0.1928	0.1984	**+2.9⋆**	0.2058	**+6.8⋆**	0.2187	**+13.4⋆**	**+6.2⋆**
At 200 docs:	0.1413	0.1483	**+4.9⋆**	0.1543	**+9.2⋆**	0.1655	**+17.1⋆**	**+7.3⋆**
At 500 docs:	0.0876	0.0913	**+4.2⋆**	0.0966	**+10.3⋆**	0.1016	**+16.1⋆**	**+5.3⋆**
At 1000 docs:	0.0557	0.0581	**+4.4⋆**	0.0619	**+11.2⋆**	0.0646	**+16.1⋆**	**+4.4⋆**
Average precision (non-interpolated) over all rel docs								
	0.1388	0.1418	+2.18	0.1534	**+10.51⋆**	0.1591	**+14.62⋆**	**+3.72⋆**

Table 3: All % differences are given with RFM as the baseline, except for the last column in which RFM+Rel is the baseline. Statistical significance at p<0.01 (t-test) is indicated with ⋆.

contain the same feature. This has two advantages: retrieval logic is separated from document structure and features can be customized to documents or collections. This model offers a flexible and fertile new foundation for retrieval algorithm design.

9. REFERENCES

[1] J. A. Aslam and M. Montague. Models for metasearch. In W. B. Croft, D. J. Harper, D. H. Kraft, and J. Zobel, editors, *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 276–284. ACM Press, September 2001.

[2] J. A. Aslam, V. Pavlu, and R. Savell. A unified model for metasearch and the efficient evaluation of retrieval systems via the hedge algorithm. In *Proc. 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 393–394, July 2003.

[3] M. Hearst. Texttiling: Segmenting text into multi-paragraph subtopic passages. *Computational Linguistics*, 23(1):33–64, 1997.

[4] J.-H. Lee. Analyses of multiple evidence combination. In *20th annual SIGIR*, pages 267–276, 1997.

[5] D. Losada and L. Azzopardi. An analysis on document length retrieval trends in language modeling smoothing. *Springer Information Retrieval Journal*, 11:109–138, 2008.

[6] M. Montague and J. A. Aslam. Relevance score normalization for metasearch. In H. Paques, L. Liu, and D. Grossman, editors, *Proceedings of the Tenth International Conference on Information and Knowledge Management*, pages 427–433. ACM Press, November 2001.

[7] M. Montague and J. A. Aslam. Condorcet fusion for improved retrieval. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management (CIKM)*, pages 538–548, November 2002.

[8] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proceedings of the ACM SIGIR Conference*, pages 275–281, 1998.

[9] S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. In *3rd annual Text REtrieval Conference*, NIST - Gaithersburg, MD, 1994.

[10] S. E. Robertson and K. S. Jones. Simple, proven approaches to text retrieval. Technical Report UCAM-CL-TR-356, University of Cambridge, Computer Laboratory, University of Cambridge, 1997.

[11] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24:513–523, 1988.

[12] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[13] J. A. Shaw and E. A. Fox. Combination of multiple searches. In *Text (REtrieval) Conference*, pages 105–108, 1994.

[14] A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *SIGIR '96*, pages 21–29, New York, NY, USA, 1996. ACM.

[15] C. G. M. Snoek, M. Worring, and A. W. M. Smeulders. Early versus late fusion in semantic video analysis. In *Proceedings of the ACM International Conference on Multimedia*, page 399402, Singapore, November 2005.

[16] H. Turtle and W. B. Croft. Evaluation of an inference network-based retrieval model. *ACM Transactions on Information Systems*, 9(3):187–222, 1991.

[17] J. Xu and H. Li. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 391–398, Amsterdam, The Netherlands, 2007. ACM Press.